Table of Contents

Temporal Logics for Non-Regular Properties: Model Checking *(invited talk)* 1
Martin Lange

Is Tractable Reasoning in Extensions of the Description Logic $\mathcal{EL}$ Useful in Practice? 8
Franz Baader, Carsten Lutz, Boontawee Suntisrivaraporn

Query Answering Based on Modal Correspondence Theory 21
Evgeny Zolin

A tableau-based decision procedure for a branching-time interval temporal logic 38
Davide Bresolin, Angelo Montanari

Verifying qualitative and quantitative properties with LTL over concrete domains 54
Regis GASCON

Complexity of Hybrid Logics over Transitive Frames 62
Martin Mundhenk, Thomas Schneider, Thomas Schwentick, Volker Weber

Two Tableau-Based Decision Procedures for Hybrid Logic 79
Thomas Bolander, Torben Braner

Hybrid Logic with operations on nominals 97
Evangelos Tzanis

A Modal View on Polder Politics 111
Wouter van Atteveldt, Stefan Schlobach

Cut–Free Axiomatizations for Stratified Modal Fixed Point Logic 125
Mathis Kretz, Gerhard Jäger, Thomas Studer

Natural deduction for belief "at most" 144
Johan W. Klwer, Arild Waaler

Modal logic and navigational XPath: an experimental comparison 156
Massimo Franceschet, Enrico Zimuel

ATEL with Common and Distributed Knowledge is ExpTime-Complete . . 173
Dirk Walther

Automatic proof search in logic of justified common knowledge 187
Yegor Bryukhov

A Completeness Result for BAN Logic 202
Mika Cohen, Mads Dam

Extensions of Tableau calculi for preference-based conditional logics 220
Laura Giordano, Valentina Gliozzi, Nicola Olivetti, Camilla Schwind

KLMLean 1.0: a Theorem Prover for Logics of Default Reasoning 235
Nicola Olivetti, Gian Luca Pozzato

Description Logics and Disjunctive Datalog—More Than just a Fleeting Resemblance? *(invited talk)* 246
Boris Motik

Theory and practice of theorem proving for monodic first-order temporal logic *(invited talk)* 266
Boris Konev

The Typechecking Problem for XML Transformations: Methods and Formal Models *(invited talk)* 268
Wim Martens

Timed Modal Logics for Specifying and Verifying Real-Time Systems *(invited talk)* 293
Franois Laroussinie

Temporal Logics for Non-Regular Properties: Model Checking

Martin Lange

Institut für Informatik, Ludwig-Maximilians-Universität München, Germany
Martin.Lange@ifi.lmu.de

Abstract

The modal μ-calculus captures exactly the bisimulation-invariant regular tree languages. Hence, properties expressed in formulas of its fragments like LTL or CTL for instance are only regular. We present temporal logics whose expressive power reaches beyond that of the modal μ-calculus and survey known result about these with a focus on their model checking problems — due to the undecidability of satisfiability.

1 Introduction

Temporal logics or, more generally, modal logic with recursion mechanism are used in computer science for program verification [1,22]. Regardless of whether verification is done in terms of model or satisfiability checking, the focus has been on logics that are fragments of or can be embedded into Kozen's modal μ-calculus $\mathcal{L}_\mu$: LTL [19], CTL [5], CTL* [6], or even PDL [7], etc.

There are equivalence preserving translations forth and back between formulas of the modal μ-calculus and various types of finite tree automata or Monadic Second Order Logic [10,11,20]. Hence, all properties expressed in the aforementioned logics are regular in the sense that the set of tree or word models of these formulas form an ω-regular language.

In fact, the expressive power of these four logics for example is strictly below that of full ω-regularity: for instance LTL defines only the star-free properties on ω-words [8,21], and on infinite trees, CTL is weaker than CTL* which coincides with the bisimulation invariant fragment of Monadic Path Logic [17]. All of them, including PDL, can be embedded into the first level of the alternation hierarchy within the modal μ-calculus [2].

Nevertheless, much attention is being payed to these weak logics when it comes to automatic verification. This is (partially) explained by two observations: (1) Increasing expressive power naturally comes with increasing computational complexity and vice-versa. In automatic verification, where

input sizes are typically very large, weak logics may be the only choice. (2) Properties expressed in stronger logics of regular properties than those, i.e. in higher levels of $\mathcal{L}_\mu$'s alternation hierarchy are usually examples of the form: something must hold infinitely often unless something else holds infinitely unless something else ... (to be read right-associatively).

This does not mean that considering even larger classes of properties will result in even more pathological examples. In fact, there are properties that are inexpressible in $\mathcal{L}_\mu$ but which are interesting for the verification of transmission protocols for example: "never more `out`- than `in`-actions", unlimited counting, repetition of sequences of actions, etc.

In Section 2 we will briefly describe four temporal logics that are capable of expressing non-regular properties. As mentioned above, higher expressive power requires higher computational complexity. The model checking problems for these logics are, in this order, complete for PTIME, PSPACE, EXPTIME, and non-elementary. The price to pay w.r.t. satisfiability is even higher: all these logics are highly undecidable. Intuitively, in order to achieve non-regular effects one has to be able to model context-freeness. Furthermore, sensible specification logics feature a $\wedge$-construct. But the intersection problem for context-free languages is undecidable.

Undecidability also restricts the model checking problem to finite structures. Checking non-regular properties on finite structures is not an oxymoron. Note that on a structure of fixed size, mutli-modal logic even suffices to express every possible μ-calculus definable property. But the formula would depend on the structure and it is desirable to have properties formalised independently of the underlying structure to be tested. The same holds for non-regular properties. Such logics provide independence of the underlying models and in same cases even succinctness over logics for regular properties.

2 Temporal Logics for Non-Regular Properties

2.1 Transition Systems

Let $\mathcal{P} = \{p, q, \ldots\}$ be a finite set of atomic proposition, and let $\Sigma = \{a, b, \ldots\}$ be a finite set of atomic action names. A transition system is a tuple $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \Sigma\}, s_0, L)$ where $\mathcal{S}$ is the set of *states*, $\xrightarrow{a}$ for any $a \in \Sigma$ is a binary relation on states called the *transitions*, $s_0 \in \mathcal{S}$ is a designated *starting state*, and $L : \mathcal{S} \to 2^{\mathcal{P}}$ labels the states with propositions.

The four logics that are presented in the following are all interpreted over transition systems.

2.2 Non-Regular PDL

Propositional Dynamic Logic, as introduced by Fisher and Ladner [7] building on a proposal by Pratt is multi-modal logic over an infinite set of transition relations that form a Kleene Algebra. It was originally introduced to describe

properties of *regular* programs, i.e. programs built from atomic ones using the regular operations *union*, *sequential composition* and *Kleene star*.

Harel, Pnueli and Stavi extended this to *Propositional Dynamic Logic of Non-Regular Programs* (PDL[CFG]) [9] by allowing programs to be built from atomic ones using the full power of context-free grammars.

Definition 2.1 Formulas of PDL[CFG] are given by the following grammar.

$$\varphi \ ::= \ q \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle G \rangle \varphi$$

where $q \in \mathcal{P}$ and G is a context-free grammar over the alphabet Σ. As usual, we write $L(G)$ to denote the language generated by G.

The semantics of PDL[CFG] is defined as follows.

$$\begin{aligned} \mathcal{T}, s \models q &\text{ iff } q \in L(s) \\ \mathcal{T}, s \models \varphi \vee \psi &\text{ iff } \mathcal{T}, s \models \varphi \text{ or } \mathcal{T}, s \models \psi \\ \mathcal{T}, s \models \neg\varphi &\text{ iff } \mathcal{T}, s \not\models \varphi \\ \mathcal{T}, s \models \langle G \rangle \varphi &\text{ iff } \exists w \in L(G), \exists t \in \mathcal{S}, \text{ s.t. } s \xrightarrow{w} t \end{aligned}$$

where the transition relations are lifted to words in Σ^* in a natural way:

$$\begin{aligned} s \xrightarrow{\epsilon} t &\text{ iff } s = t \\ s \xrightarrow{aw} t &\text{ iff } \exists u, \text{ s.t. } s \xrightarrow{a} u \text{ and } u \xrightarrow{w} t \end{aligned}$$

Example 2.2 Consider the context-free grammar G over the alphabet $\Sigma = \{a, b\}$ given by the production rule

$$S \ \rightarrow \ b \mid aSS$$

Note that $L(G) = \{w \mid \forall u, v \in \Sigma^* : w = uv \Rightarrow |u|_a \leq |u|_b\}$. This is easily seen to be a non-regular language. Moreover, $L(G)$ describes the undesired runs of a buffer if, for example, $b = \texttt{out}$ and $a = \texttt{in}$. Hence, the PDL[CFG] formula $[G]\texttt{ff} := \neg\langle G \rangle(q \vee \neg q)$ specifies that on all paths in a transition system the number of b-transitions never exceeds the number of a-transitions.

Theorem 2.3 [12] *The model checking problem for PDL[CFG] is PTIME-complete.*

2.3 The Modal Iteration Calculus

Definition 2.4 Let $(V, \leq)$ be a complete lattice with bottom element $\bot$ and suprema $\sqcup$, and let $f : V \to V$ be any function an V, not necessarily monotone. For limit ordinals λ and arbitrary ordinals α define

$$f^0 := \bot \ , \quad f^{\alpha+1} := f^\alpha \sqcup f(f^\alpha) \ , \quad f^\lambda := \bigsqcup_{\alpha<\lambda} f^\alpha$$

Then the trans-finite sequence $f^0, f^1, \ldots, f^\omega, f^{\omega+1}, \ldots$ trivially forms in increasing chain of elements in V and eventually there is an α s.t. $f^\alpha = f^{\alpha+1}$. This is called the *inflationary fixpoint* of f, denoted $\mathtt{inf} f$, and it is easy to see that it is indeed a fixpoint of f: $f(\mathtt{inf} f) = \mathtt{inf} f$.

Inflationary fixpoints are as interesting from a computational point of view as least fixpoints are. They can serve as quantifiers whose definition already yields a method for computing them.

Inspired by the (perhaps surprising) result that *First-Order Logic with Least Fixpoints* LFP is equi-expressive to *First-Order Logic with Inflationary Fixpoints* IFP, Dawar, Grädel and Kreutzer defined the *Modal Iteration Calculus* MIC which extends the modal μ-calculus with inflationary fixpoint constructs [3].

Definition 2.5 Let $\mathcal{V} = \{X, Y, \ldots\}$ be a set of second-order variables. Formulas of MIC are given by the following grammar.

$$\varphi \;:=\; q \mid X \mid \varphi \vee \varphi \mid \neg\varphi \mid \langle a \rangle \varphi \mid \mathtt{inf}(X_1 \leftarrow \varphi_1, \ldots, X_n \leftarrow \varphi_n)$$

where $q \in \mathcal{P}$, $a \in \Sigma$, and $X, X_1, \ldots, X_n \in \mathcal{V}$. The semantics of a MIC formula is defined using environments ρ that interpret the free variables in a formula by a subset of $\mathcal{S}$, e.g.

$$\begin{aligned}
&[\![\langle a \rangle \varphi]\!]_\rho \;:=\; \{s \in \mathcal{S} \mid \exists t \in \mathcal{S}, s \xrightarrow{a} t \text{ and } t \in [\![\varphi]\!]_\rho\} \\
&[\![\mathtt{inf}(X_1 \leftarrow \varphi_1, \ldots, X_n \leftarrow \varphi_n)]\!]_\rho \;:= \\
&\qquad \pi_1\big(\mathtt{inf}(\lambda(S_1, \ldots, S_n).([\![\varphi_n]\!]_{\rho[S_1/X_1, \ldots, S_n/X_n]}, \ldots, [\![\varphi_n]\!]_{\rho[S_1/X_1, \ldots, S_n/X_n]})\big)
\end{aligned}$$

etc., where π_1 projects a tuple onto its first component.

A fragment of MIC worth mentioning is 1MIC. It is obtained by replacing the syntax rule for arbitrary inflationary fixpoint by a rule for non-simultaneous fixpoint quantifiers only: $\mathtt{inf} X \leftarrow \varphi$.

Example 2.6 Take the MIC formula

$$\mathtt{inf}(X \leftarrow q \vee (\langle - \rangle \mathtt{tt} \wedge [-](X \wedge \neg Y)), Y \leftarrow X \wedge \neg q)$$

This expresses *uniform inevitability*: the formula is satisfied in a state s of a transition system iff there is a $k \in \mathbb{N}$ s.t. on all paths beginning in s, q holds after k steps. No finite tree automaton recognises this language [4].

Theorem 2.7 [3] *The model checking problem for MIC is PSPACE-complete.*

2.4 Fixpoint Logic with Chop

Müller-Olm's *Fixpoint Logic with Chop* (FLC) extends the modal μ-calculus with a sequential composition operator [18]. This makes it possible to achieve context-free, and because of the presence of intersection, even certain context-sensitive effects.

Definition 2.8 Let $\mathcal{V} = \{X, Y, \ldots\}$ be a set of third-order variables for functions from sets of states to sets of states. Formulas of FLC are given by the following grammar.

$$\varphi \;::=\; q \mid \neg q \mid X \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi; \varphi \mid \langle a \rangle \mid [a] \mid \tau \mid \mu X.\varphi \mid \nu X.\varphi$$

where $q \in \mathcal{P}$, $a \in \Sigma$ and $X \in \mathcal{V}$. Note that the semantics of a modal μ-calculus formula is a predicate, a subset of the state space. In order to cope with sequential composition, the semantics of FLC lifts the modal μ-calculus semantics straight-forwardly to the space of monotone *predicate transformers*. For details we refer to [18]. A game-based characterisation of the FLC semantics has been given in [13].

Example 2.9 The following FLC formula expresses "all finite paths are of the form ww for some $w \in \Sigma^*$".

$$\varphi \;:=\; \bigwedge_{(a,b)\in\Sigma^2, a\neq b} \psi_a; \psi_b; \langle - \rangle; \mathtt{tt} \qquad \text{where} \quad \psi_x \;:=\; \nu X.[x] \wedge [-]; X; [-]$$

where $\mathtt{tt} := q \vee \neg q$ for some q. This is based on the usual trick which shows that the complement of the non-context-free language $\{ww \mid w \in \Sigma^*\}$ is context-free.

Theorem 2.10 [16,14] *The model checking problem for FLC is EXPTIME-complete.*

2.5 Higher-Order Fixpoint Logic

The idea of obtaining more expressiveness by employing higher-order constructs has been followed consequently by Viswanathan and Viswanathan in the introduction of *Higher Order Fixpoint Logic* (HFL) [23]. It is a hybrid of the modal μ-calculus and a simply typed λ-calculus.

Types of HFL formulas are given by the grammar

$$\tau \;::=\; \mathrm{Pr} \mid \tau \to \tau$$

where the atomic type Pr stands for predicates, and function types are understood canonically. Variables in HFL formulas occur either only positively, or only negatively, or without any constraint. Thus, a *variance* is a $\sigma \in \{+, -, ?\}$. The syntax of HFL is then given by the following grammar.

$$\varphi \;::=\; q \mid \varphi \vee \varphi \mid \neg\varphi \mid X \mid \langle a \rangle \varphi \mid \varphi\ \varphi \mid \lambda X^{\sigma}.\varphi \mid \mu X.\varphi$$

Typing rules ensure the well-formedness of formulas. For a detailed definition of HFL including its straight-forward typing rules see the introductory paper [23]. We also refer to this for the exact definition of the semantics which is given through liftings of predicates to monotone higher order functions on

some set of states of a transition system. Monotonicity is guaranteed by the typing rules: fixpoint variables may only occur positively in their defining fixpoint body.

Example 2.11 An example of a class of properties that can be formalised in HFL has been given by Viswanathan and Viswanathan []: *assume-guarantee properties*. A state satisfies the formula $\nu X.\varphi \triangleright \nu Y.\psi$ iff for all $\alpha \in \mathbb{O}rd$: if s satisfies $\nu^{\alpha} X.\varphi$ then s also satisfies $\nu^{\alpha+1} Y.\psi$. In other words, if we assume φ to hold α many times then we must guarantee ψ at least $\alpha + 1$ many times. This property is expressed by the HFL formula

$$(\nu Z.\lambda X.\lambda Y.(\neg X \vee Y) \wedge Z\ \varphi\ \psi)\ \mathtt{tt}\ \psi$$

where the greatest fixpoint quantifier ν is abbreviated as usual through least fixpoints and negation.

Theorem 2.12 [15] *The model checking problem for HFL is non-elementary but elementary for every HFL^k.*

References

[1] Clarke, E. M., E. A. Emerson and A. P. Sistla, *Automatic verification of finite state concurrent systems using temporal logic specifications*, in: *Proc. 10th Symp. on Principles of Programming Languages, POPL'83* (1983), pp. 117–126.

[2] Dam, M., *CTL* and ECTL* as fragments of the modal μ-calculus*, TCS **126** (1994), pp. 77–96.

[3] Dawar, A., E. Grädel and S. Kreutzer, *Inflationary fixed points in modal logic*, in: L. Fribourg, editor, *Proc. 15th Workshop on Computer Science Logic, CSL'01*, LNCS (2001), pp. 277–291.

[4] Emerson, E. A., *Uniform inevitability is tree automaton ineffable*, Information Processing Letters **24** (1987), pp. 77–79.

[5] Emerson, E. A. and J. Y. Halpern, *Decision procedures and expressiveness in the temporal logic of branching time*, Journal of Computer and System Sciences **30** (1985), pp. 1–24.

[6] Emerson, E. A. and J. Y. Halpern, *"Sometimes" and "not never" revisited: On branching versus linear time temporal logic*, Journal of the ACM **33** (1986), pp. 151–178.

[7] Fischer, M. J. and R. E. Ladner, *Propositional dynamic logic of regular programs*, Journal of Computer and System Sciences **18** (1979), pp. 194–211.

[8] Gabbay, D., A. Pnueli, S. Shelah and J. Stavi, *The temporal analysis of fairness*, in: *Proc. 7th Symp. on Principles of Programming Languages, POPL'80* (1980), pp. 163–173.

[9] Harel, D., A. Pnueli and J. Stavi, *Propositional dynamic logic of nonregular programs*, Journal of Computer and System Sciences **26** (1983), pp. 222–243.

[10] Janin, D. and I. Walukiewicz, *Automata for the μ-calculus and related results*, in: J. Wiedermann and P. Hájek, editors, *Proc. 20th Symp. on Math. Foundations of Computer Science, MFCS'95*, LNCS **969** (1995).

[11] Kaivola, R., *On modal μ-calculus and Büchi tree automata*, Information Processing Letters **54** (1995), pp. 17–22.

[12] Lange, M., *Model checking propositional dynamic logic with all extras*, Journal of Applied Logic **??**, pp. ??–??, accepted for publication.

[13] Lange, M., *Local model checking games for fixed point logic with chop*, in: L. Brim, P. Jančar, M. Křetínský and A. Kučera, editors, *Proc. 13th Conf. on Concurrency Theory, CONCUR'02*, LNCS **2421** (2002), pp. 240–254.

[14] Lange, M., *Three notes on the complexity of model checking fixpoint logic with chop* (2005), (submitted).

[15] Lange, M. and R. Somla, *The complexity of model checking higher order fixpoint logic*, in: J. Jedrzejowicz and A. Szepietowski, editors, *Proc. 30th Int. Symp. on Math. Foundations of Computer Science, MFCS'05*, LNCS **3618** (2005), pp. 640–651.

[16] Lange, M. and C. Stirling, *Model checking fixed point logic with chop*, in: M. Nielsen and U. H. Engberg, editors, *Proc. 5th Conf. on Foundations of Software Science and Computation Structures, FOSSACS'02*, LNCS **2303** (2002), pp. 250–263.

[17] Moller, F. and A. Rabinovich, *Counting on CTL*: On the expressive power of monadic path logic*, Information and Computation **184** (2003), pp. 147–159.

[18] Müller-Olm, M., *A modal fixpoint logic with chop*, in: C. Meinel and S. Tison, editors, *Proc. 16th Symp. on Theoretical Aspects of Computer Science, STACS'99*, LNCS **1563** (1999), pp. 510–520.

[19] Pnueli, A., *The temporal logic of programs*, in: *Proc. 18th Symp. on Foundations of Computer Science, FOCS'77* (1977), pp. 46–57.

[20] Rabin, M. O., *Decidability of second-order theories and automata on infinite trees*, Trans. of Amer. Math. Soc. **141** (1969), pp. 1–35.

[21] Thomas, W., *Star-free regular sets of ω-sequences*, Information and Control **42** (1979), pp. 148–156.

[22] Vardi, M. Y. and P. Wolper, *An automata-theoretic approach to automatic program verification (preliminary report)*, in: *Proc. 1st Symp. on Logic in Computer Science, LICS'86*, IEEE, Washington, DC, 1986 pp. 332–344.

[23] Viswanathan, M. and R. Viswanathan, *A higher order modal fixed point logic*, in: P. Gardner and N. Yoshida, editors, *Proc. 15th Int. Conf. on Concurrency Theory, CONCUR'04*, LNCS **3170** (2004), pp. 512–528.

Is Tractable Reasoning in Extensions of the Description Logic $\mathcal{EL}$ Useful in Practice?

Franz Baader, Carsten Lutz, Boontawee Suntisrivaraporn

Theoretical Computer Science, TU Dresden, Germany
{baader,lutz,meng}@tcs.inf.tu-dresden.de

Abstract. Extensions of the description logic $\mathcal{EL}$ have recently been proposed as lightweight ontology languages. The most important feature of these extensions is that, despite including powerful expressive means such as general concept inclusion axioms, reasoning can be carried out in polynomial time. In this paper, we consider one of these extensions, $\mathcal{EL}^+$, and introduce a refinement of the known polynomial-time classification algorithm for this logic, which was implemented in our CEL reasoner. We describe the results of several experiments with CEL on large ontologies from practice, which show that even a relatively straightforward implementation of the described algorithm outperforms highly optimized, state-of-the-art tableau reasoners for expressive description logics.

1 Introduction and Motivation

The quest for tractable (i.e., polynomial-time decidable) description logics (DLs), which started in the 1980s after the first intractability results for DLs were shown [5, 18], was until recently restricted to DLs extending the basic language $\mathcal{FL}_0$, which allows for conjunction ($\sqcap$) and value restrictions ($\forall r.C$). The main reason was that, when clarifying the logical status of property arcs in semantic networks and slots in frames, the decision was taken that arcs/slots should be read as value restrictions rather than existential restrictions ($\exists r.C$).

Unfortunately, as soon as terminologies (also called TBoxes or DL ontologies) were taken into consideration, tractability turned out to be unattainable in $\mathcal{FL}_0$: even classifying the simplest form of TBoxes that admit only acyclic concept definitions was shown to be coNP-hard [19]. If the most general form of TBoxes is admitted, which consists of general concept inclusion axioms (GCIs) supported by all modern DL systems, then classification in $\mathcal{FL}_0$ even becomes ExpTime-complete [2].

For these reasons, and also because of the need for expressive DLs in applications, from the mid 1990s on, the DL community has mainly given up on the quest of finding tractable DLs. Instead, it investigated more and more expressive DLs, for which reasoning is worst-case intractable. The goal was then to find practical reasoning procedures, i.e., algorithms that are easy to implement and optimize, and which—though worst-case exponential or even worse—behave well in practice (see, e.g., [17]). This line of research has resulted in the availability of

highly optimized DL systems for expressive DLs based on tableau algorithms [14, 10], and successful applications: most notably the recommendation by the W3C of the DL-based language OWL [16] as the ontology language for the Semantic Web.

Recently, the choice of value restrictions as a sine qua non of DLs has been reconsidered. On the one hand, it was shown that the DL $\mathcal{EL}$, which allows for conjunction and existential restrictions, has better algorithmic properties than $\mathcal{FL}_0$. Classification of both acyclic and cyclic $\mathcal{EL}$ TBoxes is tractable [1], and this remains so even if general TBoxes with GCIs are admitted [6]. On the other hand, there are applications where value restrictions are not needed, and where the expressive power of $\mathcal{EL}$ or small extensions thereof appear to be sufficient. In fact, SNOMED, the Systematized Nomenclature of Medicine, employs $\mathcal{EL}$ with an acyclic TBox [23]. Large parts of the Galen medical knowledge base can also be expressed in $\mathcal{EL}$ with GCIs and transitive roles [21]. Finally, the Gene Ontology [7] can be seen as an acyclic $\mathcal{EL}$ TBox with one transitive role.

The tractability results for $\mathcal{EL}$ together with the bio-medical applications mentioned above have motivated our research on extensions of $\mathcal{EL}$: the leitmotif for this research was to extend $\mathcal{EL}$ as far as possible by adding standard DL constructors available in ontology languages like OWL, while still retaining polynomial-time reasoning in the presence of GCIs. This has resulted in the tractable DL $\mathcal{EL}^{++}$ [2], which includes transitive roles, so-called right-identities [23] on roles, nominals (and thus ABoxes), and disjointness constraints on concepts. The purpose of the research presented in the present paper was to evaluate whether or not the polynomial-time algorithms for reasoning in $\mathcal{EL}$ and its extensions are suitable as a basis for implementing a DL reasoning system that can handle large bio-medical ontologies, and whether such a reasoner outperforms existing high-optimized DL reasoners for expressive DLs.

At first sight, one might think that a polynomial-time algorithm is always better suited for implementation than worst-case exponential-time algorithms such as the ones underlying modern DL reasoners. However, due to the plethora of sophisticated optimization techniques that have been developed for tableau algorithms over the last decade [15], it is far from obvious whether a straightforward implementation of the polynomial-time algorithm can compete with highly-optimized implementations of tableau algorithms. A case in point is our experience with implementing the polynomial-time classification algorithms for cyclic $\mathcal{EL}$ TBoxes introduced in [1]: direct implementations of both the algorithm for subsumption w.r.t. descriptive semantics (based on a reduction to satisfiability of propositional Horn formulae [9]) and the algorithm for subsumption w.r.t. greatest fixpoint semantics (based on computing the greatest simulation on a graph [11]) did not lead to satisfactory results on the Gene Ontology [24].

In this paper, we consider a restriction of the polynomial-time classification algorithm for $\mathcal{EL}^{++}$ [2] to the fragment $\mathcal{EL}^{+}$ of $\mathcal{EL}^{++}$. This fragment differs from $\mathcal{EL}^{++}$ in that nominals and the bottom concept are disallowed. The reason for considering this fragment was that none of the bio-medical ontologies mentioned above use nominals or the bottom concept. We describe a refined version

of this algorithm that is tailored toward implementation. The purpose of this refinement is to remove an obvious obstacle for efficient implementation of the algorithm as given in [2]: the uninformed, brute-force search for applicable completion rules. With (almost) no further optimizations, we have implemented the refined algorithm in our CEL (Classifier for EL) reasoner. We have performed several experiments to compare the performance of CEL with the performance of state-of-the-art DL systems based on tableau algorithms. It turns out that CEL can compete with modern DL systems and often outperforms them. We view these results as a serious encouragement for further research into *optimized* implementations of DL reasoners based on polynomial-time algorithms for the $\mathcal{EL}$ family of DLs.

2 The Description Logic $\mathcal{EL}^+$

In DLs, *concept descriptions* are inductively defined with the help of a set of *constructors*, starting with a set NC of *concept names* and a set NR of *role names*. $\mathcal{EL}^+$ concept descriptions are formed using the constructors shown in the upper part of Table 1. An $\mathcal{EL}^+$ *ontology* is a finite set of *general concept inclusions (GCIs)* and *role inclusions (RIs)*, whose syntax is shown in the lower part of Table 1.

The semantics of $\mathcal{EL}^+$ is defined in terms of *interpretations* $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where the domain $\Delta^\mathcal{I}$ is a non-empty set of individuals, and the interpretation function $\cdot^\mathcal{I}$ maps each concept name $A \in \mathsf{NC}$ to a subset $A^\mathcal{I}$ of $\Delta^\mathcal{I}$ and each role name $r \in \mathsf{NR}$ to a binary relation $r^\mathcal{I}$ on $\Delta^\mathcal{I}$. The extension of $\cdot^\mathcal{I}$ to arbitrary concept descriptions is inductively defined as shown in the semantics column of Table 1. An interpretation $\mathcal{I}$ is a *model* of an ontology $\mathcal{O}$ if, for each inclusion in $\mathcal{O}$, the conditions given in the semantics column of Table 1 are satisfied.

Name	Syntax	Semantics
top	$\top$	$\Delta^\mathcal{I}$
conjunction	$C \sqcap D$	$C^\mathcal{I} \cap D^\mathcal{I}$
existential restriction	$\exists r.C$	$\{x \in \Delta^\mathcal{I} \mid \exists y \in \Delta^\mathcal{I} : (x,y) \in r^\mathcal{I} \wedge y \in C^\mathcal{I}\}$
general concept inclusion	$C \sqsubseteq D$	$C^\mathcal{I} \subseteq D^\mathcal{I}$
role inclusion	$r_1 \circ \cdots \circ r_n \sqsubseteq s$	$r_1^\mathcal{I} \circ \cdots \circ r_n^\mathcal{I} \subseteq s^\mathcal{I}$

Table 1. Syntax and semantics of $\mathcal{EL}^+$.

One main use of GCIs in $\mathcal{EL}^+$ is to give definitions of concept names in terms of complex concept descriptions. Therefore, we introduce *concept definitions* $A \equiv C$, with A a concept name, as an abbreviation for the two GCIs $A \sqsubseteq C$ and $C \sqsubseteq A$. Intuitively, C describes the *necessary and sufficient* conditions for being an instance of A. GCIs of the form $A \sqsubseteq C$, with A a concept

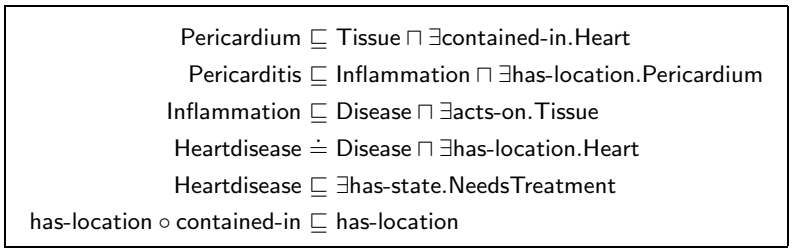


Fig. 1. An example $\mathcal{EL}^+$ ontology.

name, are called *primitive concept definitions*.[1] They give only necessary (but no sufficient) conditions for being an instance of A. In DL, a finite set of GCIs is commonly called a *general TBox*, and a finite set of (possibly primitive) concept definitions with unique left-hand sides is called a *TBox*. We call a TBox *primitive* if it contains only primitive concept definitions and *acyclic* if there are no concept names $A_0, \ldots, A_{n-1}$ such that $A_{(i+1) \bmod n}$ occurs on the right hand of the (possibly primitive) concept definition of A_i, for all $i < n$.

It is worthwhile to note that the role inclusions available in $\mathcal{EL}^+$ generalize a number of standard expressive means: role inclusions of the form $r \sqsubseteq s$ are commonly called *role hierarchies*; *transitivity* of a role r can be expressed by writing $r \circ r \sqsubseteq r$; finally, RIs can express *right-identity rules* $r \circ s \sqsubseteq r$, which play an important role in medical ontologies [23]. An example ontology formulated in $\mathcal{EL}^+$ can be found in Figure 1, where all uppercase words are concept names, and all lowercase words are role names.

The basic inference problem for DL concept descriptions is *concept subsumption*: a concept C is subsumed by a concept D w.r.t. an ontology $\mathcal{O}$ (written $C \sqsubseteq_{\mathcal{O}} D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in every model $\mathcal{I}$ of $\mathcal{O}$. The basic inference problem for DL ontologies is *classification*: compute the subsumption hierarchy of all concept names occurring in the ontology $\mathcal{O}$. In our example ontology, it is not hard to see that Pericarditis is classified as Heartdisease (i.e., Pericarditis $\sqsubseteq_{\mathcal{O}}$ Heartdisease), and thus needs treatment.

3 Classifying an $\mathcal{EL}^+$ Ontology

A polynomial-time algorithm for classification in $\mathcal{EL}$ with GCIs and role hierarchies has been proposed in [6], and this algorithm was extended to the more powerful DL $\mathcal{EL}^{++}$ in [2]. We introduce the restriction of the algorithm from [2] to $\mathcal{EL}^+$, and then propose a refined version for implementation purposes.

Both in tableau-based DL systems and in earlier DL systems based on structural subsumption algorithms, the subsumption hierarchy is computed by performing multiple subsumption tests. In addition to optimizing the single subsumption tests, such systems can also be optimized by trying to minimize the

[1] despite not actually defining anything.

number of subsumption tests needed to compute the whole hierarchy [3]. In contrast, the classification algorithm in [2] simultaneously computes the subsumption relationships between *all* pairs of concept names in the input ontology.

To classify an ontology, the algorithm first transforms it into *normal form*, which requires that all GCIs and RIs are of one of the forms shown in the left part of Figure 2. By introducing new concept and role names and applying a number of straightforward rewriting rules, any $\mathcal{EL}^+$ ontology $\mathcal{O}$ can be transformed into a normalized one such that subsumption between the concept names occurring in $\mathcal{O}$ is preserved. The normalization can be carried out in linear time, yielding an ontology whose size is linear in the size of the original one [2, 24].

Normal Form	Completion Rules
$A \sqsubseteq B$	**CR1** If $A \in S(X)$, $A \sqsubseteq B \in \mathcal{O}$, and $B \notin S(X)$ then $S(X) := S(X) \cup \{B\}$
$A_1 \sqcap A_2 \sqsubseteq B$	**CR2** If $A_1, A_2 \in S(X)$, $A_1 \sqcap A_2 \sqsubseteq B \in \mathcal{O}$, and $B \notin S(X)$ then $S(X) := S(X) \cup \{B\}$
$A \sqsubseteq \exists r.B$	**CR3** If $A \in S(X)$, $A \sqsubseteq \exists r.B \in \mathcal{O}$, and $(X, B) \notin R(r)$ then $R(r) := R(r) \cup \{(X, B)\}$
$\exists r.A \sqsubseteq B$	**CR4** If $(X, Y) \in R(r)$, $A \in S(Y)$, $\exists r.A \sqsubseteq B \in \mathcal{O}$, and $B \notin S(X)$ then $S(X) := S(X) \cup \{B\}$
$r \sqsubseteq s$	**CR5** If $(X, Y) \in R(r)$, $r \sqsubseteq s \in \mathcal{O}$, and $(X, Y) \notin R(s)$ then $R(s) := R(s) \cup \{(X, Y)\}$
$r \circ s \sqsubseteq t$	**CR6** If $(X, Y) \in R(r)$, $(Y, Z) \in R(s)$, $r \circ s \sqsubseteq t \in \mathcal{O}$, and $(X, Z) \notin R(t)$ then $R(t) := R(t) \cup \{(X, Z)\}$

Fig. 2. Normal Form and Completion Rules

For the rest of this section, we assume without loss of generality that the input ontology $\mathcal{O}$ is in normal form. Let $\mathsf{NC}_\mathcal{O}$ be the set of all concept names occurring in $\mathcal{O}$ including $\top$, and $\mathsf{NR}_\mathcal{O}$ be the set of all role names occurring in $\mathcal{O}$. The algorithm computes

- a mapping S assigning to each element of $\mathsf{NC}_\mathcal{O}$ a subset of $\mathsf{NC}_\mathcal{O}$, and
- a mapping R assigning to each element of $\mathsf{NR}_\mathcal{O}$ a binary relation on $\mathsf{NC}_\mathcal{O}$.

The intuition is that these mappings make implicit subsumption relationships explicit in the sense that $B \in S(A)$ implies $A \sqsubseteq_\mathcal{O} B$, and $(A, B) \in R(r)$ implies $A \sqsubseteq_\mathcal{O} \exists r.B$. The mappings are initialized by setting $S(A) := \{A, \top\}$ for each $A \in \mathsf{NC}_\mathcal{O}$ and $R(r) := \emptyset$ for each $r \in \mathsf{NR}_\mathcal{O}$. Then the sets $S(A)$ and $R(r)$ are extended by applying the completion rules shown in the right part of Figure 2 until no more rule applies. The algorithm has been proved sound and complete in [2], i.e., after termination we have $B \in S(A)$ iff $A \sqsubseteq_\mathcal{O} B$, for all concept

names A, B occurring in $\mathcal{O}$. It has also been proved that the algorithm always terminates in time polynomial in the size of the input ontology.

One of the main problems to be solved when implementing the described algorithm is to develop a good approach for finding the next rule to be applied. If this is realized by a naïve brute-force search for applicable completion rules, then one cannot expect an acceptable runtime behavior on large inputs. As a solution to this problem, we propose a refined version of the algorithm, which is inspired by the linear-time algorithm for satisfiability of propositional Horn formulas proposed in [9]. This version uses a set of queues, one for each concept name appearing in the input ontology, to guide the application of completion rules. Intuitively, the queues list modifications to the data structure (i.e. to the sets $S(A)$ and $R(r)$) that still have to be carried out. The possible entries of the queues are of the form

$$B, \quad B \to B', \quad \text{and} \quad \exists r.B$$

with B and B' concept names, and r a role name. The entry $B \in \mathsf{queue}(A)$ means that the concept name B has to be added to $S(A)$. Similarly, $B \to B' \in \mathsf{queue}(A)$ means that B' has to be added to $S(A)$ if $S(A)$ already contains B, and $\exists r.B \in \mathsf{queue}(A)$ means that (A, B) has to be added to $R(r)$. The fact that such an addition triggers other rules will be taken into account by appropriately extending the queues when the addition is performed.

To facilitate describing the manipulation of the queues, we view the (normalized) input ontology $\mathcal{O}$ as a mapping $\widehat{\mathcal{O}}$ from concepts to sets of queue entries as follows: for each concept name A (including the case $A = \top$), $\widehat{\mathcal{O}}(A)$ is the minimal set of queue entries such that

- if $A \sqsubseteq B \in \mathcal{O}$, then $B \in \widehat{\mathcal{O}}(A)$;
- if $A \sqcap A' \sqsubseteq B \in \mathcal{O}$ or $A' \sqcap A \sqsubseteq B \in \mathcal{O}$, then $A' \to B \in \widehat{\mathcal{O}}(A)$;
- if $A \sqsubseteq \exists r.B \in \mathcal{O}$, then $\exists r.B \in \widehat{\mathcal{O}}(A)$.

Likewise, for each concept $\exists r.A$, $\widehat{\mathcal{O}}(\exists r.A)$ is the minimal set of queue entries such that, if $\exists r.A \sqsubseteq B \in \mathcal{O}$, then $B \in \widehat{\mathcal{O}}(\exists r.A)$.

Now, we can describe how the queues are used: since the sets $S(A)$ are initialized with $\{A, \top\}$, we initialize $\mathsf{queue}(A)$ with $\widehat{\mathcal{O}}(A) \cup \widehat{\mathcal{O}}(\top)$, i.e., we add to the queues the *immediate* consequences of being an instance of A and $\top$. Then, we repeatedly fetch (and thereby remove) entries from the non-empty queues and process them using the procedure process displayed in Figure 3. To be more precise, $\mathsf{process}(A, X)$ is called when we are currently treating the concept name A, and X is the next element on $\mathsf{queue}(A)$. Observe that the second if-clause implements **CR1** and (part of) **CR4**, the first if-clause is a pre-processing step addressing the **CR2** rule and delegating the real work to the second if-clause, and the third if-clause implements **CR3**, (the other part of) **CR4**, as well as **CR5** and **CR6**. The procedure $\mathsf{process\text{-}new\text{-}edge}(A, r, B)$ handles the effects of adding a new pair (A, B) to $R(r)$. The notation $\sqsubseteq^*_{\mathcal{O}}$ used in its top-most **for**-loop stands for the reflexive-transitive closure of the role hierarchy statements.

```
procedure process(A, X)
begin
    if X = B → B′ then
        if B ∈ S(A) then
            continue with X := B′;
        else return; {do nothing}
    if X is a concept name and X ∉ S(A) then
        S(A) := S(A) ∪ {X};
        queue(A) := queue(A) ∪ Ô(X);
        for all concept names B and role names r with (B, A) ∈ R(r) do
            queue(B) = queue(B) ∪ Ô(∃r.X);
    if X is an existential restriction ∃r.B and (A, B) ∉ R(r) then
        process-new-edge(A, r, B)
end;
procedure process-new-edge(A, r, B)
begin
    for all role names s with r ⊑*_O s do
        R(s) := R(s) ∪ {(A, B)};
        queue(A) := queue(A) ∪ ⋃_{B′|B′∈S(B)} Ô(∃s.B′);
        for all concept names A′ and role names t, u with
            t ∘ s ⊑ u ∈ O and (A′, A) ∈ R(t) and (A′, B) ∉ R(u) do
            process-new-edge(A′, u, B);
        for all concept names B′ and role names t, u with
            s ∘ t ⊑ u ∈ O and (B, B′) ∈ R(t) and (A, B′) ∉ R(u) do
            process-new-edge(A, u, B′);
end;
```

Fig. 3. Processing the queue

Queue processing is continued until all queues are empty. Observe that the refined algorithm need not perform *any* search to check which completion rules are applicable. It can be proved that the refined algorithm is still sound and complete, and that it terminates in polynomial time.

4 Implementation and Evaluation

Modern DL reasoners are usually based on tableau-based subsumption algorithms [4]. Although such algorithms are exponential in the worst case, the development of a whole plethora of sophisticated optimization techniques has led to a quite good runtime behavior in practice. In this section we will show that, nevertheless, even a relatively naïve implementation of the refined algorithm described above can compete with, and even outperform, modern tableau-based DL systems.

We have implemented the refined algorithm described in the previous section in the CEL reasoner. CEL is written in Common LISP and accepts input based

on a small extension of the KRSS syntax [20]. For details about the use of the system, refer to the CEL manual.[2] To test whether CEL can compete with modern tableau-based reasoners, we have conducted a number of experiments based on three important bio-medical ontologies: the Gene Ontology (GO) [7], the Galen medical knowledge base (GALEN) [21], and the Systematized Nomenclature of Medicine (SNOMED) [8].

GO. The Gene Ontology provides a controlled vocabulary to describe gene and gene product attributes in any organism. It currently consists of 16,803 concepts and a single, transitive role "part-of". The original distribution of GO used a frame-like formalism without formal semantics. For example, the concept Polarisome is described as follows:

```
[Term]
id: GO:0000133
name: polarisome
namespace: cellular_component
def: "Protein complex playing a role in determining cell polarity"
is_a: GO:0043234 ! protein complex
relationship: part_of GO:0005938 ! cell cortex
relationship: part_of GO:0030427 ! site of polarized growth
```

The most natural approach to translate GO concept definitions into an $\mathcal{EL}^+$ ontology is to use primitive concept definitions. For example, the above GO concept would be defined as

$$\mathsf{GO0000133} \sqsubseteq \mathsf{GO0043234} \sqcap \exists \mathsf{part\_of}.\mathsf{GO0005938} \sqcap \exists \mathsf{part\_of}.\mathsf{GO0030427}.$$

This translation gives us $\mathcal{O}^{\mathrm{GO}}_{\mathsf{prim}}$, an acyclic, primitive TBox with one role inclusion $\mathsf{part\_of} \circ \mathsf{part\_of} \sqsubseteq \mathsf{part\_of}$. It coincides with the OWL version of GO contained in recent distributions of the gene ontology. However, as an ontology for testing DL systems, $\mathcal{O}^{\mathrm{GO}}_{\mathsf{prim}}$ is not well-suited since it is too simple. Actually, computing subsumption in an acyclic, primitive TBox is relatively easy: it simply means computing the transitive closure over the explicitly told subsumption relationships. In the above example, $\mathsf{GO0000133} \sqsubseteq \mathsf{GO0043234}$ is such an explicit relationship. It should be noted however, that neither tableau-based systems nor our implementation directly use this fact.

To obtain an ontology based on GO that is harder to classify, we also consider the translation of GO into non-primitive concept definitions, i.e., the above GO concept would be defined as

$$\mathsf{GO0000133} \equiv \mathsf{GO0043234} \sqcap \exists \mathsf{part\_of}.\mathsf{GO0005938} \sqcap \exists \mathsf{part\_of}.\mathsf{GO0030427}.$$

The result $\mathcal{O}^{\mathrm{GO}}_{\mathsf{def}}$ is an acyclic TBox with the same additional role inclusion as above. In $\mathcal{O}^{\mathrm{GO}}_{\mathsf{def}}$, subsumption cannot be computed as the transitive closure of told subsumptions. However, $\mathcal{O}^{\mathrm{GO}}_{\mathsf{def}}$ is a relatively unusual ontology: due to the

[2] The CEL reasoner and the manual are available at http://lat.inf.tu-dresden.de/systems/cel

assumption that concept definitions also provide sufficient conditions, which has not been intended by the GO developers, there are large groups of concepts that turn out to be equivalent (i.e., subsume each other).

GALEN. This ontology aims to promote the sharing and re-use of medical data. It was originally formulated in the language GRAIL and has been translated into description logic by Ian Horrocks [12]. In that translation, GALEN is formulated in $\mathcal{EL}$ extended with GCIs, role hierarchies, transitive roles, functional roles, and inverse roles. Since $\mathcal{EL}^+$ and CEL do not support inverse roles and functional roles, we have removed inverse role axioms and treat functional roles as ordinary ones. In this way, we obtain the $\mathcal{EL}^+$ ontology $\mathcal{O}^{\text{GALEN}}$ that contains 1,214 GCIs as well as 2,041 primitive and 699 non-primitive concept definitions. It refers to 413 roles, of which 26 are declared transitive. Moreover, there are 412 role hierarchy axioms.

SNOMED is the systematized nomenclature of medicine, a standardization of medical terminology used in the health systems of the US and the UK. The current version of SNOMED comprises 379,691 concept and 52 role names. SNOMED is formulated as an acyclic $\mathcal{EL}$ TBox that contains 38,719 concept definitions and 340,972 primitive concept definitions. There are no transitive roles,[3] but 11 role hierarchy statements and one right-identity rule of the form $r \circ s \sqsubseteq r$. This gives us the ontology $\mathcal{O}^{\text{SNOMED}}$. To get a smaller version of SNOMED that can be handled by standard DL reasoners, we also consider the fragment that is obtained by keeping only the non-primitive concept definitions. We call the resulting ontology $\mathcal{O}^{\text{SNOMED}}_{\mathsf{core}}$.

The CEL reasoner reads the input ontology, converts each concept definition (if any) into a pair of GCIs, normalizes the resulting ontology, and then starts the refined algorithm described in the previous section. When performing experiments with the initial version of CEL, we found that it could classify only the ontologies $\mathcal{O}^{\text{GO}}_{\mathsf{prim}}$, $\mathcal{O}^{\text{GALEN}}$, and $\mathcal{O}^{\text{SNOMED}}_{\mathsf{core}}$, but not $\mathcal{O}^{\text{GO}}_{\mathsf{def}}$ and $\mathcal{O}^{\text{SNOMED}}$. In the latter two cases, the problem is that the data structures become too big due to too many computed subsumption relationships. A more careful analysis revealed that the problem has different origins for the two ontologies. In the case of $\mathcal{O}^{\text{GO}}_{\mathsf{def}}$, the many subsumption relationships are due to the presence of large groups of equivalent concepts: this is problematic as n equivalent concepts give rise to n^2 subsumption relationships. In addition, if there is a subsumption between two equivalence classes of size n and m, then we must store $n \cdot m$ subsumption relationships. In the case of $\mathcal{O}^{\text{SNOMED}}$, many additional concept names are introduced during the normalization phase, for which we also compute all subsumption relationships. These observations led us to adopting the following improvements of the original implementation.

Synonyms identification. Told synonyms are concept names A, B that are explicitly stated to be equivalent by a concept definition $A \equiv B$. We improved

[3] Actually, SNOMED needs transitive roles, but since the commercial reasoner used by the developers cannot handle transitivity, they have simulated it in an incomplete way using the approach described in [22].

the algorithm by choosing and keeping only a single representative for each equivalence class of told synonyms. All the dropped synonyms are stored in a cost-effective *union-find* data structure to allow the identification of their representative. This is needed for answering subsumption queries that involve dropped synonyms. The removal of synonyms reduces the number of concept names by more than half in the case of $\mathcal{O}^{\text{GO}}_{\mathsf{def}}$ and enables classification of this ontology by CEL.

Improved normal form. Here, the goal is to reduce the number of additional concept names introduced through normalization. This is achieved in two ways. First, the purpose of additional concept names introduced during normalization is to replace complex concept descriptions in GCIs. The general pattern is that we introduce a concept name A as replacement for a complex concept C, and then add one of $A \sqsubseteq C$ and $A \sqsupseteq C$ to the ontology, depending on whether C was replaced on the left-hand side or on the right-hand side of a GCI. In the improved normalization, we keep track of the concept descriptions C for which an abbreviation has already been introduced, and avoid the introduction of multiple concept names for the same concept description.

Second, in the improved normal form we admit n-ary conjunction on the left-hand side of GCIs, thus avoiding the introduction of $n-1$ concept names for each left-hand side n-ary conjunction. Of course, the Completion Rule **CR2** has to be generalized accordingly:

CR2 If $A_1, \ldots, A_n \in S(X)$, $A_1 \sqcap \cdots \sqcap A_n \sqsubseteq B \in \mathcal{O}$, and $B \notin S(X)$
then $S(X) := S(X) \cup \{B\}$

Also, queue entries of the form $B \rightarrow B'$ are now replaced by the more general entries $B_1, \cdots, B_n \rightarrow B'$ with the obvious meaning. It is straightforward to generalize the process function given in Figure 3 to the new type of queue entries. These two modifications had a drastic effect when normalizing $\mathcal{O}^{\text{SNOMED}}$: whereas the original normalization approach introduced 401,830 new concepts, the improved one introduced "only" 114,658. We have compared the perfor-

	$\mathcal{O}^{\text{GO}}_{\mathsf{prim}}$	$\mathcal{O}^{\text{GO}}_{\mathsf{def}}$	$\mathcal{O}^{\text{GALEN}}$	$\mathcal{O}^{\text{SNOMED}}_{\mathsf{core}}$	$\mathcal{O}^{\text{SNOMED}}$
No. of CDefs.	0	16,803	699	38,719	38,719
No. of PCDefs.	16,803	0	2041	0	340,972
No. of GCIs	0	0	1214	0	0
No. of role axioms	1	1	438	0	12
$\lvert\mathsf{CN}_{\mathcal{O}}\rvert$	16,806	16,806	2,740	53,234	379,691
$\lvert\mathsf{RN}_{\mathcal{O}}\rvert$	1	1	413	52	52
CEL	25	10,833	16	1,360	12,772
FaCT	117	11	19	*unattainable*	*unattainable*
RACER	36	*unattainable*	25	84,917	*unattainable*

Table 2. Benchmarks and Evaluation Results

mance of CEL with two of the most advanced tableau-based reasoning systems: FaCT (v2.32.17) and RACER (v1.7.6). Both systems implement expressive DLs in which subsumption is ExpTime-complete. All experiments have been performed on a PC with 2.4GHz Intel Pentium 4 processor and 2GB memory running RedHat Linux 7.2. In the case of Galen, for the sake of fairness also FaCT and RACER have been used with the restricted version of Galen that includes neither functional nor inverse roles. In the case of Snomed, the right-identity rule was passed to CEL, but not to FaCT and RACER, as the latter do not support right identities.

The results of our experiments are summarized in the lower part of Table 2, where *unattainable* means that the reasoner failed due to exhaustion of memory. Notably, CEL outperforms both FaCT and RACER in all benchmarks except $\mathcal{O}_{\mathsf{def}}^{\mathrm{Go}}$, where FaCT needs only 11 seconds (we currently have no explanation for the good behavior of FaCT on $\mathcal{O}_{\mathsf{def}}^{\mathrm{Go}}$). Moreover, CEL was the only reasoner that was able to classify the whole Snomed ontology.[4]

5 Conclusion

We have proposed a refined classification algorithm for the description logic $\mathcal{EL}^+$, implemented it in the CEL reasoner, and then used CEL to show that even a relatively straightforward implementation of this algorithm can compete with, and usually even outperform, highly optimized tableau-based DL reasoners. We view this result as a strong argument for the use of tractable DLs, provided that their expressive power is sufficient for the application in question. We also believe that, when more powerful optimization techniques are developed, reasoners based on the refined algorithm presented in this paper can be made much more efficient than our current implementation CEL. Such additional optimizations may include the following:

Currently, we explicitly compute and store all the subsumption relationships between concepts names, rather than just computing and storing the Hasse-diagram of the subsumption hierarchy, as done by the other DL systems. Storing only the Hasse-diagram will definitely save space, but it is not yet clear how much time overhead (for computing subsumption relationships from this representation) this will cause.

Closely related to the previous point is the question whether we can use the well-known techniques for avoiding calls to the subsumption algorithm used by other DL systems while computing the Hasse-diagram (see [3]). With the current algorithm, this does not really make sense since a single call to our subsumption algorithm already computes the whole hierarchy. Thus, one must first modify our classification algorithm into a goal-directed subsumption algorithm that checks, with minimum efforts, only whether two given concept names are in a subsumption relationship.

[4] We have recently heard that the optimizations described in [13] have the effect that FaCT^{++} is now also capable of classifying $\mathcal{O}^{\mathrm{Snomed}}$. We were, however, not yet able to verify this claim with the version of FaCT^{++} available for download.

Another technique used by other DL systems is to determine some obvious (non-)subsumption relationships (such as told subsumers) before calling the actual subsumption algorithm. It is not clear yet how this kind of information can be used in our type of algorithm, but one could try to adopt the approach described in [13].

As shown by our experiments, the reduction of the number of concept names is a crucial issue for the efficiency of our algorithm. Therefore, any techniques that make such a reduction is promising candidates for further optimizations. It is clear that the two simple optimizations described in this paper can further be improved, but there may also be other, more sophisticated approaches to reduce the number of concepts to be considered.

Finally, we plan to extend our implementation CEL to the description logic $\mathcal{EL}^{++}$ [2].

References

1. Franz Baader. Terminological cycles in a description logic with existential restrictions. In *Proc. IJCAI'03*, 2003.
2. Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proc. IJCAI'05*, 2005.
3. Franz Baader, Enrico Franconi, Bernhard Hollunder, Bernhard Nebel, and Hans-Jürgen Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994.
4. Franz Baader and Ulrike Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.
5. Ronald J. Brachman and Hector J. Levesque. The tractability of subsumption in frame-based description languages. In *Proc. AAAI'84*, 1984.
6. Sebastian Brandt. Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and—what else? In *Proc. ECAI'04*, 2004.
7. The Gene Ontology Consortium. Gene Ontology: Tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.
8. R.A. Cote, D.J. Rothwell, J.L. Palotay, R.S. Beckett, and L. Brochu. The systematized nomenclature of human and veterinary medicine. Technical report, SNOMED International, Northfield, IL: College of American Pathologists, 1993.
9. William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *J. Logic Programming*, 1(3):267–284, 1984.
10. Volker Haarslev and Ralf Möller. RACER system description. In *Proc. IJCAR'01*, 2001.
11. Monika R. Henzinger, Thomas A. Henzinger, and Peter W. Kopke. Computing simulations on finite and infinite graphs. In *36th Annual Symposium on Foundations of Computer Science*, 1995.
12. Ian Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.
13. Dmitry Tsarkov and Ian Horrocks. Optimised classification for taxonomic knowledge bases. In *Proc. DL'05*, 2005.

14. Ian Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. KR'98*, 1998.
15. Ian Horrocks. Implementation and optimization techniques. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press, 2003.
16. Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
17. Ian Horrocks, Ulrike Sattler, and Stefan Tobies. Practical reasoning for very expressive description logics. *J. of the IGPL*, 8(3):239–264, 2000.
18. Bernhard Nebel. Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, 34(3):371–383, 1988.
19. Bernhard Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.
20. Peter F. Patel-Schneider and Bill Swartout. Description-logic knowledge representation system specification from the KRSS group of the ARPA knowledge sharing effort. Technical report, DARPA Knowledge Representation System Specification (KRSS) Group of the Knowledge Sharing Initiative, 1993.
21. Alan Rector and Ian Horrocks. Experience building a large, re-usable medical ontology using a description logic with transitivity and concept inclusions. In *Proceedings of the Workshop on Ontological Engineering, AAAI Spring Symposium (AAAI'97)*, Stanford, CA, 1997. AAAI Press.
22. Stefan Schulz, Martin Romacker, and Udo Hahn. Part-whole reasoning in medical ontologies revisited: Introducing SEP triplets into classification-based description logics. *Journal of the American Medical Informatics Association (JAMIA)*, pages 830–834, 1998.
23. Kent A. Spackman. Managing clinical terminology hierarchies using algorithmic calculation of subsumption: Experience with SNOMED-RT. *J. of the American Medical Informatics Association*, 2000. Fall Symposium Special Issue.
24. Boontawee Suntisrivaraporn. Optimization and implementation of subsumption algorithms for the description logic $\mathcal{EL}$ with cyclic TBoxes and general concept inclusion axioms. Master thesis, TU Dresden, Germany, 2005.

Query Answering Based on Modal Correspondence Theory

Evgeny Zolin

School of Computer Science
University of Manchester
Manchester, UK
ezolin@cs.man.ac.uk

Abstract. We propose a query answering technique applicable to a wide family of cyclic conjunctive queries with one distinguished variable. It exploits the fact that, given such a query, one can build in linear time a concept of some Description Logic (DL) such that to answer the query w.r.t. any knowledge base (KB) is the same as to find all instances of that concept. Notably, the method is uniform: the concept does not depend on a KB and even on a DL in which the KB is formulated. Thus, for these conjunctive queries, the problem of query answering is reduced to concept instance retrieval, which in turn is reducible to KB (un)satisfiability. The technique presented here is based on the modal correspondence theory (and close relationship between modal and description logics) and can be regarded as a practical application of that theory.

1 Introduction

Description Logics are family of knowledge representation formalisms closely related to first-order and modal logics. Most of them can be viewed as decidable fragments of the first-order logic. Their complexity was thoroughly investigated, and practical reasoning algorithms were designed (e.g., [9, 11, 14]) and implemented as highly optimised reasoning systems, e.g., Racer [7], FaCT [8], and Pellet [18]. Typical reasoning services provided by these systems are checking KB satisfiability, concept subsumption (and building a taxonomy, i.e., a computing subsumption relationships between all concept names occurring in a KB), concept instances retrieval etc. (see [1, Chap. 8] for an overview).

Among reasoning tasks, querying is a fundamental mechanism for extracting information from a KB. Two most important reasoning services involving queries—query answering and query containment, also called subsumption—were extensively explored [3–6, 10, 12, 17], but have not got a complete solution. A typical obstacle is the presence of cycles in a query (cf. [6]). In general, query containment and answering are reducible to each other and are at least as hard as concept subsumption or satisfiability problems (whence their lower complexity bounds). However, to the best of our knowledge, no tight upper bounds for their complexity are obtained so far. To illustrate the situation, recall that the Description Logic $\mathcal{SHIQ}$ is known to be ExpTime-complete [11]. At the same

time, the complexity of query answering over $\mathcal{SHIQ}$ KBs is shown in [5] to be 3coNExpTime, if a KB has no transitive roles, and 4coNExpTime otherwise.

In this paper, we concentrate on the query answering problem. We show that that for a wide family of conjunctive queries (with one distinguished variable) one can build in linear time a concept (usually, in a very simple DL) whose instances are exactly the answers to the query (w.r.t. any KB). As a consequence, within this family, the query answering problem is reduced to concept instance retrieval, which in turn is reducible to KB unsatisfiability. Note that the reduction is applicable to many cyclic queries, and the resulting concept is not in general equivalent to the query. The technique presented here is based on the modal correspondence theory; this theory explores the relation between the properties of frames expressible in modal and first-order languages.

To illustrate the method, consider the following instance retrieval request: $\mathcal{KB} \models a\colon (\neg X \sqcup \exists R.X)$, where the concept name X does not occur in the knowledge base $\mathcal{KB}$. The task is to find all individuals a (called constants in first-order logic) occurring in $\mathcal{KB}$ that belong to the concept $\neg X \sqcup \exists R.X$ in any model of $\mathcal{KB}$. It is not hard to show that exactly thos individuals a will be retrieved that satisfy (in all models of $\mathcal{KB}$) the condition aRa. Hence, the concept $\neg X \sqcup \exists R.X$ can be used for answering the cyclic query $q(x) \leftarrow xRx$. Similarly, the request $\mathcal{KB} \models a\colon (\neg\exists R.X \sqcup \exists S.X)$, where X does not occur in $\mathcal{KB}$, returns exactly those individuals a that are answers of the query $q(x) \leftarrow \exists y(xRy \wedge xSy)$. Thus, concepts involving *fresh* concept names can be used to compute the answers of even cyclic queries.

Now recall from modal logic that the formula $p \to \Diamond p$ is *valid at a point* e of a frame iff this point is reflexive, i.e., eRe holds. Similarly, the formula $\Box_R p \to \Diamond_S p$ is valid at a point e iff there is a point d such that eRd and eSd holds, i.e., e satisfies the first-order condition $\alpha(x) := \exists y(xRy \wedge xSy)$. Comparing this with the above observations, we can generalise it as follows: whenever the validity of a modal formula at a point of a frame is equivalent to some first-order condition (and this condition has the form of conjunctive query), we can use this fact for query answering. This is the main result of this paper (Theorem 7), and an open question is whether the converse implication holds. An important feature of this approach is its uniformity: a concept used to answers a query is not only independent of a KB against which the query is answered, but also independent of a DL in which a KB is formulated. Therefore, extending the expressivity of a DL does not destroy our query answering algorithm (in contrast to other approaches where, e.g., introducing transitive roles can invalidate an algorithm).

In the next section we recall some notions concerning DLs and query answering. Section 3 contains the main theorem of the paper, which enables us to transfer results from modal correspondence theory to query answering. As an application, Sect. 4 and 5 present two families of queries that can be handled by this approach. As a by-product, in Sect. 5 we obtain an (at least syntactic) extension to the well-known in modal logic Sahlqvist theorem. Numerous examples of queries captured by this technique are shown in Sect. 6. We give conclusions and an outlook in Sect. 7. An Appendix contains all the proofs.

$$\top^{\mathcal{I}} = \Delta^{\mathcal{I}} \quad \{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\} \qquad (C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$$
$$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \qquad (R^{-})^{\mathcal{I}} = \{\langle e, d\rangle \mid \langle d, e\rangle \in R^{\mathcal{I}}\}$$
$$(\geqslant n\, R.C)^{\mathcal{I}} = \{\ e \in \Delta^{\mathcal{I}} \ \mid\ \#\{d \in C^{\mathcal{I}} \colon\ \langle e, d\rangle \in R^{\mathcal{I}}\} \geqslant n\ \}$$

Fig. 1. Semantics for Description Logics.

2 Preliminaries

First let us recall the notions related to the DL $\mathcal{ALC}$ and its extensions.

Definition 1. (Syntax) The vocabulary consists of finite sets of *concept names* CN, *role names* RN, and *individual names* IN (also called as *constants*). Concepts of the logic $\mathcal{ALC}$ are defined by the following syntax:

$$C ::= \quad \top \quad | \quad A \quad | \quad \neg C \quad | \quad C \sqcap D \quad | \quad \exists R.C,$$

where A is a concept name, R a role name, and C and D are concepts. Other connectives are taken as customary abbreviations, e.g., $(C \sqcup D) := \neg(\neg C \sqcap \neg D)$, $(C \to D) := \neg(C \sqcap \neg D)$, $\bot := \neg\top$, $\forall R.C := \neg\exists R.\neg C$.

A *terminology* (or a TBox) $\mathcal{T}$ is a finite set of *axioms* of the form $C \sqsubseteq D$, where C, D are arbitrary concepts. An ABox $\mathcal{A}$ is a finite set of *assertions* of the form $a\colon C$ and aRb, where $a, b \in \mathsf{IN}$, C is a concept and R a role. Finally, a *knowledge base* $\mathcal{KB} = \langle \mathcal{T}, \mathcal{A}\rangle$ consists of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$.

Appending the letters $\mathcal{I}$, $\mathcal{Q}$, $\mathcal{O}$, or $\mathcal{H}$ to the name of the logic refers to the following syntax extensions:

$\mathcal{I}$: *inverse roles*: if R is a role, then R^{-} is a role;
$\mathcal{Q}$: *qualified number restrictions*: if R is a role, C a concept, and $n > 0$, then $(\geqslant n\, R.C)$ is a concept; then we can use an abbreviation $\exists R.C := (\geqslant 1\, R.C)$;
$\mathcal{O}$: *nominals*: if $a \in \mathsf{IN}$, then $\{a\}$ is a concept;
$\mathcal{H}$: *role hierarchy*: axioms of the form $R \sqsubseteq S$ are allowed in a TBox.

Replacing '$\mathcal{ALC}$' with '$\mathcal{S}$' in the name of a logic refers to allowing transitive roles, i.e., axioms of the form $\mathsf{Trans}(R)$ in a TBox. To maintain decidability, in presence of $\mathcal{S}$, $\mathcal{H}$, and $\mathcal{Q}$ together, the restriction is imposed on the concept syntax: expressions of the form $(\geqslant n\, R.C)$ are regarded as well-formed concepts only if the role R has no transitive subroles w.r.t. given TBox (cf. [9]).

Definition 2. (Semantics) An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of non-empty *domain* $\Delta^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$ that maps:

- each constant $a \in \mathsf{IN}$ to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$,
- each concept name $C \in \mathsf{CN}$ to a subset $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$,
- each role name $R \in \mathsf{RN}$ to a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$;

and is extended to all concepts and roles by inductive clauses shown in Fig. 1. The *unique name assumption* (UNA) is customary made, i.e., only interpretations mapping distinct constants to distinct elements of the domain $\Delta^{\mathcal{I}}$ are considered.

An interpretation $\mathcal{I}$ *satisfies* an axiom $C \sqsubseteq D$, $R \sqsubseteq S$, or $\mathsf{Trans}(R)$ resp., if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$, or the relation $R^{\mathcal{I}}$ is transitive, resp.; $\mathcal{I}$ satisfies an assertion $a\colon C$ or aRb if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ or $\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$, resp. An interpretation is a *model* of a knowledge base if it satisfies all its TBox axioms and ABox assertions. The fact '$\mathcal{I}$ satisfies Φ' is written as $\mathcal{I} \models \Phi$. A knowledge base $\mathcal{KB}$ *entails* Φ (written as $\mathcal{KB} \models \Phi$) if, for all models $\mathcal{I}$ of $\mathcal{KB}$, we have $\mathcal{I} \models \Phi$.

Definition 3. (Queries) A *conjunctive query* is an expression of the form[1]

$$q(\vec{x}) \;\leftarrow\; t_1(\vec{x}, \vec{y}) \wedge \ldots \wedge t_n(\vec{x}, \vec{y}),$$

where $\vec{x}, \vec{y}$ are tuples of (*distinguished*, resp., *non-distinguished*) variables, and each *atom* $t_i(\vec{x}, \vec{y})$ is of the form $w\colon C$ (*concept atom*) or wRz (*role atom*), where C is a concept, R a role, and w, z are either variables from $\vec{x}, \vec{y}$ or constants. The *arity* of a query $q(\vec{x})$ is the number of its distinguished variables: $\mathsf{ar}(q) := |\vec{x}|$. Queries of arity 1 are called *unary*. Given an interpretation $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}} \rangle$, a query q of arity m is interpreted as follows:

$$q^{\mathcal{I}} := \{\; \vec{e} \in \Delta^m \;\mid\; \mathcal{I} \models \exists \vec{y}\, (\, t_1(\vec{e}, \vec{y}) \wedge \ldots \wedge t_n(\vec{e}, \vec{y}) \,) \;\}.$$

The *answer set* of a query $q(\vec{x})$ w.r.t. a knowledge base $\mathcal{KB}$ is defined as the set of tuples of constants satisfying the query q in all models of $\mathcal{KB}$:

$$\mathsf{ans}_{\mathcal{KB}}(q) := \{\; \vec{a} \in \mathsf{IN} \;\mid\; \mathcal{KB} \models q(\vec{a}) \;\}.$$

Note that the answer set is always finite: $|\,\mathsf{ans}_{\mathcal{KB}}(q)| \leqslant |\mathsf{IN}|^{\mathsf{ar}(q)}$, and corresponds, in a model $\mathcal{I}$ of $\mathcal{KB}$, to a subset (a *named* part) of $q^{\mathcal{I}}$, i.e., the inclusion $\mathsf{ans}_{\mathcal{KB}}(q)^{\mathcal{I}} \subseteq q^{\mathcal{I}}$ always holds.

It is worth saying that the query answering problem is closely related to the query subsumption problem addressed by many researches [3, 10]. Recall that a query $q(\vec{x})$ *subsumes* a query $q'(\vec{x})$ w.r.t. a $\mathcal{KB}$ (written as $\mathcal{KB} \models q \sqsupseteq q'$) if $q^{\mathcal{I}} \supseteq q'^{\mathcal{I}}$ holds for every model $\mathcal{I}$ of $\mathcal{KB}$. In fact, these two reasoning problems are reducible to each other. Indeed, a tuple $\vec{a}$ belongs to the answer set of $q(\vec{x})$ iff the subsumption $\mathcal{KB} \models \top \sqsubseteq q(\vec{a})$ holds. Conversely, given a subsumption $\mathcal{KB} \models q(\vec{x}) \sqsubseteq q'(\vec{x})$ to be tested, let $\vec{y}$ be the list of non-distinguished variables in $q(\vec{x})$. Then we introduce tuples of *fresh* constants $\vec{a_x}$ and $\vec{a_y}$ corresponding to $\vec{x}$ and $\vec{y}$, resp., and define the *canonical* ABox $\mathcal{A}_q$ for q as the set of atoms of $q(\vec{x})$ with constants $\vec{a_x}$ and $\vec{a_y}$ substituted for variables $\vec{x}$ and $\vec{y}$:

$$\text{if}\;\; q(\vec{x}) = \textstyle\bigwedge_{i=1}^{n} t_i(\vec{x}, \vec{y}) \quad \text{then}\;\; \mathcal{A}_q := \{\; t_i(\vec{a_x}, \vec{a_y}) \;\mid\; 1 \leqslant i \leqslant n \;\}.$$

Then it is not hard to show that $\mathcal{KB} \models q \sqsubseteq q'$ iff the tuple $\vec{a_x}$ belongs to the answer set of $q'(\vec{x})$ w.r.t. the knowledge base $\mathcal{KB} \cup \mathcal{A}_q$, i.e., $\mathcal{KB} \cup \mathcal{A}_q \models q'(\vec{a_x})$.

[1] The existential quantifier $\exists \vec{y}$ in front of the r.h.s. of the query is customary omitted (and so we did), though implicitly assumed, as follows from the semantics below.

3 Queries answered by concepts

In the remainder of the paper, we will be concerned with unary queries only. The traditional query answering algorithms are usually based on the so called *rolling-up* technique (see [17] for definitions). In its basic form, when applied to a unary query $q(x)$, it can be viewed as a method of building a concept C that is equivalent to $q(x)$ (in the sense that $q^{\mathcal{I}} = C^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{KB}$), so that to answer the query $q(x)$ is the same as to retrieve all instances of the concept C. However, the equivalence of a concept C to a query $q(x)$ is sufficient, but not necessary for query answering purposes. Moreover, a rather simple query $q(x) \leftarrow xRx$ is not equivalent to any $\mathcal{ALC}$-concept; this can be easily shown using the tree model property for $\mathcal{ALC}$. Rolling-up techniques work particularly good for tree-like queries, but face certain problems when applied to cyclic ones (see, e.g., [6]). To overcome these difficulties, we introduce the following main notion of our paper.

Definition 4. A unary query $q(x)$ is *answered by* a concept C (written as $q(x) \approx C$) if $q(x)$ and C (considered as a query $x\colon C$) have the same answer sets w.r.t. any knowledge base $\mathcal{KB}$: $\mathsf{ans}_{\mathcal{KB}}(q) = \mathsf{ans}_{\mathcal{KB}}(x\colon C)$. In other words, $q(x) \approx C$ iff, for any $\mathcal{KB}$ and any $a \in \mathsf{IN}$, we have: $\mathcal{KB} \models q(a) \Leftrightarrow \mathcal{KB} \models a\colon C$.

Notice the quantification over *any* KB in this definition. Strictly speaking, we should formulate the definition as follows: a query $q(x)$ is answered by a concept C *over a description logic* $\mathcal{L}$ if $\mathsf{ans}_{\mathcal{KB}}(q) = \mathsf{ans}_{\mathcal{KB}}(x\colon C)$ for any $\mathcal{KB}$ formulated in the logic $\mathcal{L}$. However, we will not complicate the matters in this way, since all the results in our paper will have the form: a certain query $q(x)$ is answered by a concept C over *any* DL $\mathcal{L}$ (at least containing the syntax necessary for formulating C and q) and even over any first-order theory etc. The following lemma will be useful for us later.

Lemma 5. *If the queries $q_1(x) \leftarrow \alpha(x, \vec{y})$ and $q_2(x) \leftarrow \beta(x, \vec{z})$ are answered by concepts C and D, resp., and $\vec{y}, \vec{z}$ are disjoint lists of non-distinguished variables, then the query $q(x) \leftarrow \alpha(x, \vec{y}) \wedge \beta(x, \vec{z})$ is answered by the concept $C \sqcap D$.*

Proof. Follows from: $\exists \vec{y} \exists \vec{z}\ (\alpha(x, \vec{y}) \wedge \beta(x, \vec{z})) \ \equiv\ \exists \vec{y}\, \alpha(x, \vec{y}) \wedge \exists \vec{z}\, \beta(x, \vec{z})$. ⊣

Our task is to determine what kind of queries can be answered by concepts and when these concepts can be found efficiently (and preferably in the same language as the query). We will see that this can be done for a wide range of unary queries, provided that we are allowed to use "fresh" concept names not occurring in KBs and queries under consideration, i.e., specially reserved for query answering purposes. This is a rather customary assumption: in the rolling-up technique, new ("representative") concept names are also freely introduced (cf. [17]). But, unlike rolling-up, we do not need to add assertions involving these fresh concept names to a $\mathcal{KB}$. Fresh concept names will be denoted by capital letters from the end of alphabet (X, Y, Z, etc.), possibly sub- or superscripted.

The remainder of the section is devoted to formulation and proof of the main theorem of our paper. This theorem allows us to transfer results from the *modal*

correspondence theory to the query answering field. The background information on that theory can be found in [2, Chap. 3]; here we will recall its basic notions.

It is known (cf. [16]) that the DL $\mathcal{ALC}$ with role names $R_1, \dots, R_n$ is a notational variant of the propositional multi-modal language with modalities $\Box_1, \dots, \Box_n$, whose formulas are defined inductively: $\bot$ and propositional letters $p_0, p_1, \dots$ are formulas; if φ and ψ are formulas, then so are $\varphi \to \psi$ and $\Box_i \varphi$. Exploiting this fact, whenever φ is a modal formula, we denote by C_φ the concept obtained from φ by replacing $\Box_i$ with $\forall R_i$ and p_j with *fresh* concept name X_j. Kripke semantics for the modal language is introduced as usually (cf. [2]). A modal formula φ is *valid* at the point e of a frame F (written as $F, e \Vdash \varphi$) if, for any valuation ν on F, φ is true at e in a model $\langle F, \nu \rangle$.

Consider also the first-order language (with equality) having only binary predicate symbols $R_1, \dots, R_n$. Note that conjunctive queries containing no concept atoms (we will call them *relational* queries) belong to this first-order language. The following is the key notion in modal correspondence theory.

Definition 6. A first-order formula $\alpha(x)$ with a single free variable *locally corresponds* to a modal formula φ (written as $\alpha(x) \leftrightsquigarrow \varphi$) if, for any frame F and its point e, the equivalence holds: $F \models \alpha(e) \Leftrightarrow F, e \Vdash \varphi$.

Now we are ready to prove the theorem that establishes a connection between the notion of local correspondence and the query answering problem for relational queries. Later on, we will extend the family of queries covered by this approach to the ones containing concept atoms as well.

Theorem 7 (Reduction). *If a unary relational query $q(x)$ locally corresponds to a modal formula φ, then the query $q(x)$ is answered by the $\mathcal{ALC}$-concept C_φ. In symbols: $q(x) \leftrightsquigarrow \varphi \Longrightarrow q(x) \approx C_\varphi$.*

The proof can be found in the Appendix. This theorem allows us to reuse *positive* results from the modal correspondence theory for query answering purposes. For example, it implies that the query $q(x) \leftarrow xRx$ is answered by the concept $X \to \exists R.X$ (where X is a fresh concept name). Other applications will be demonstrated in the subsequent sections. A natural question is whether the converse of this theorem holds; it would enable also to transfer *negative* results. We have succeeded to prove only a half of the converse, in the following sense (the proof is again in the Appendix).

Lemma 8. *Suppose that a unary relational query $q(x)$ is answered by a concept C_φ, i.e., $q(x) \approx C_\varphi$, for some modal formula φ. Then for any frame F and its point e, the condition $F \models q(e)$ implies*[2] *$F, e \Vdash \varphi$.*

In what follows, we consider only queries with no constants in role atoms, since the general query answering problem is reducible to this case, at the price of introducing nominals in concept atoms: we can equivalently rewrite an atom yRa as $yRz \wedge z{:}\{a\}$, for a new non-distinguished variable z, and similarly for aRy.

[2] Replacing this implication by equivalence would yield the converse of Theorem 7. After this paper has been accepted, the author has succeeded to prove the converse implication, but only for *finitely branching* frames.

4 Application 1: Queries and the Sahlqvist fragment

The notion of local correspondence is extensively explored in modal logic. The most prominent in this direction is the result that a wide family of modal formulas (known as *Sahlqvist formulas*) locally correspond to first-order (FO) conditions, and moreover, there is an algorithm that takes a Sahlqvist formula and returns the corresponding FO condition (see [2, §3.5–3.6] for details; originally in [15]). We will use here a sort of converse of this result, obtained by M. Kracht (see [2, §3.7]; originally in [13]), who described syntactically the class of all those FO formulas (now called *Kracht formulas*) that locally correspond to Sahlqvist formulas, and provided the corresponding algorithm. We will apply this result to those Kracht formulas that have the form of a (unary relational) conjunctive query and then extend it to queries containing also concept atoms.

First, recall the definition of Kracht formulas. Consider a FO language having only binary relation symbols $R_1, \dots, R_n$. Formulas of the form $\exists y(xRy \land \alpha(y))$ and $\forall y(xRy \to \alpha(y))$ will be abbreviated[3] as $(\exists y\overline{R}x)\alpha(y)$ and $(\forall y\overline{R}x)\alpha(y)$, resp. A formula is called *clean* if no variable occurs both free and bound in it, and no two distinct occurrences of quantifiers bind the same variable. The formulas $\top$, $\bot$, and $xR_\sigma y$ are called *quasi-atomic*, where $\sigma = (i_1, \dots, i_r)$ is a sequence of indices from $\{1, \dots, n\}$, and $xR_\sigma y$ stands for $\exists z_2 \dots \exists z_r(xR_{i_1}z_2 \land z_2R_{i_2}z_3 \land \dots \land z_rR_{i_r}y)$.

Definition 9. A formula is *restrictedly positive* if it is built up from quasi-atomic ones using conjunction $\land$, disjunction $\lor$, and restricted quantifiers $(\exists y\overline{R}x)$ and $(\forall y\overline{R}x)$. A variable y in a *clean* formula is called *inherently universal* if either y is free, or y is bound by a restricted universal quantifier $(\forall y\overline{R}x)$ which is not in the scope of any existential quantifier. Finally, a *Kracht formula* is a clean, restrictedly positive formula such that every of its quasi-atomic subformulas contains at least one inherently universal variable.

For example, $(\forall y\overline{R}x)(\forall z\overline{R}x)(\exists w\overline{R}y)zRw$ is a Kracht formula, whereas the formula $(\exists y\overline{R}x)(\forall z\overline{R}x)yRz$ is not. Now we present a family of relational queries that belong to the Kracht fragment. We will not prove that this family is exhaustive, in the sense that it covers, modulo equivalence, exactly the intersection of the Kracht fragment and conjunctive queries (although this is likely to be true).

Lemma 10. *Any relational conjunctive query of the following form is (equivalent to a Kracht formula and hence) answered by an $\mathcal{ALC}$-concept:*

$$q(x) \;\leftarrow\; \exists \vec{y}\,\big(\,\mathsf{Tree}(x,\vec{y}) \;\land\; \textstyle\bigwedge_{i,j} xR_{\sigma_i}y_j \;\land\; \bigwedge_{k,\ell} y_kR_{\tau_\ell}x\,\big),$$

where $\mathsf{Tree}(x,\vec{y})$ is a conjunction of role atoms forming an oriented x-rooted tree.[4] *In other words, $q(x)$ states the existence of an x-rooted oriented tree with additional oriented chains linking x with other nodes (in any direction).*

[3] This is similar to designating a formula $\exists y(x \leqslant y \land \alpha(y))$ by $(\exists y \geqslant x)\alpha(y)$.

[4] An *oriented tree* is a connected oriented graph without cycles (and loops $\langle e, e\rangle$) s.t. each node has at most one incoming edge. Its *root* is a node with no incoming edges.

PROOF. The conjunct $\mathsf{Tree}(x, \vec{y})$ allows to equivalently rewrite the quantifier prefix $\exists \vec{y}$ into a restricted one:

$$\exists \vec{y}\,(\mathsf{Tree}(x, \vec{y}) \,\wedge\, \alpha(x, \vec{y})) \;\equiv\; (\exists y_1 \overline{R}_1 y_1') \ldots (\exists y_r \overline{R}_r y_r')\, \alpha(x, \vec{y}),$$

where the condition $y_i' \in \{x, y_1, \ldots, y_{i-1}\}$ holds, since the formula is clean and x is the only free variable. It is easily seen that the r.h.s. is a Kracht formula, and hence locally corresponds to a modal formula. Now we apply Theorem 7 to conclude that the query $q(x)$ is answered by some $\mathcal{ALC}$-concept. ⊣

Next we describe an algorithm for answering these queries; it also captures queries with concept atoms. By Lemma 5, we can consider queries $q(x)$ without atoms of the form $x\colon A$ and xRx, since adding these atoms is equivalent to adding the conjuncts A and $X \to \exists R.X$ to the concept that answers the query.

The algorithm for answering queries within the Kracht fragment

Assume that we are given a conjunctive query of the form

$$q(x) \;\leftarrow\; \exists \vec{y}\,\big(\,\mathsf{Tree}(x, \vec{y}) \;\wedge\; \underbrace{\textstyle\bigwedge_{i,j} x R_{\sigma_i} y_j}_{\alpha(x,\vec{y})} \;\wedge\; \underbrace{\textstyle\bigwedge_{k,\ell} y_k R_{\tau_\ell} x}_{\beta(x,\vec{y})} \;\wedge\; \underbrace{\textstyle\bigwedge_{t} y_t\colon B_t}_{\gamma(\vec{y})}\big), \qquad (\mathcal{K})$$

where $\mathsf{Tree}(x, \vec{y})$ is a conjunction of role atoms forming an oriented x-rooted tree; expressions of the form $yR_\sigma z$ are role chains linking y with z; the components of the query are denoted by α, β, γ for easy reference. We assume that the conjuncts in the above formula $(\mathcal{K})$ are distinct. Then we proceed as follows:

1) Introduce *fresh* concept names: $Y_{i,j}$ for each conjunct $xR_{\sigma_i}y_j$ in $\alpha(x, \vec{y})$, and $X_{k,\ell}$ for each conjunct $y_k R_{\tau_\ell} x$ in $\beta(x, \vec{y})$.

2) Build two conjunctions PRE and CON (for 'premise' and 'conclusion', resp.) by first putting PRE $:= x\colon \top$ and CON $:= y_1\colon \top \wedge \ldots \wedge y_r\colon \top$ and then proceed as follows (here $\forall R_\sigma.Y$ stands for $\forall R_{i_1}. \cdots \forall R_{i_r}.Y$, if $\sigma = (i_1, \ldots, i_r)$, and similarly for $\exists R_\tau.X$):
- for each $xR_{\sigma_i}y_j$ in α, add $x\colon \forall R_{\sigma_i}.Y_{i,j}$ to PRE and $y_j\colon Y_{i,j}$ to CON;
- for each $y_k R_{\tau_\ell} x$ in β, add $x\colon X_{k,\ell}$ to PRE and $y_k\colon \exists R_{\tau_\ell}.X_{k,\ell}$ to CON;
- for each $y_t\colon B_t$ in γ, add the conjunct $y_t\colon B_t$ to CON.

3) In PRE and CON obtained, merge conjuncts related to the same variables, i.e., replace $z\colon C \wedge z\colon D$ by $z\colon (C \sqcap D)$. After this, PRE will be of the form $x\colon C$ and CON of the form: $y_1\colon D_1 \wedge \ldots \wedge y_r\colon D_r$, for some concepts C and D_i.

4) Form the expression PRE $\to \exists \vec{y}\,(\mathsf{Tree}(x, \vec{y}) \wedge$ CON $\wedge\, x\colon \top)$, or explicitly:

$$x\colon C \to (\exists y_1 \overline{R}_1 y_1') \ldots (\exists y_r \overline{R}_r y_r')\,(\,y_1\colon D_1 \wedge \ldots \wedge y_r\colon D_r \wedge x\colon \top\,)$$

where $y_i' \in \{x, y_1, \ldots, y_{i-1}\}$. Now apply the standard rolling-up technique to turn the restricted quantifiers $(\exists y_i \overline{R}_i y_i')$ into DL-quantifiers, starting from the innermost one. For example, assuming for notational simplicity that $y_r' \equiv y_1$ (the case $y_r' \equiv x$ is similar), the variable y_r is eliminated as follows:

$$\begin{array}{l}(\exists y_r \overline{R}_r y_1)\,(\,y_1\colon D_1 \wedge \ldots \wedge y_r\colon D_r \wedge x\colon E\,) \;\equiv\\ y_1\colon (D_1 \sqcap \exists R_r.D_r) \wedge y_2\colon D_2 \wedge \ldots \wedge y_{r-1}\colon D_{r-1} \wedge x\colon E.\end{array}$$

After eliminating all the variables y_i, CON will be of the form $x\colon D$. Hence the expression considered at the beginning of Item 4 is equivalently transformed into $x\colon C \to x\colon D$. The desired concept answering the query $q(x)$ is $(C \to D)$, or, in the traditional DL notation, $\neg C \sqcup D$.

Note that the resulting concept belongs to the minimal DL containing the concepts B_t occurring in the original query. In particular, for a relational query we obtain an $\mathcal{ALC}$-concept. We will illustrate how this algorithm works in Ex. 5 (see Sect. 6). The algorithm described above is taken from the general Kracht's Theorem (see [2, §3.7]) and simplified for the case of FO formulas having the form of conjunctive queries (and slightly modified to capture concept atoms). One can follow the proof of Kracht's Theorem to show that the algorithm indeed returns a concept that answers the original query. We sum up with the following

Theorem 11. *The query answering problem for unary conjunctive queries from the family $\mathcal{K}$ is linearly reduced to the problem of concept instance retrieval (which in turn is reducible to knowledge base unsatisfiability).*

5 Application 2: Parallel-serial queries

It is known that the Sahlqvist fragment does not cover all modal formulas having FO correspondents, as well as the Kracht fragment does not contain all FO formulas that locally correspond to modal formulas. In this section we show that the family $\mathcal{K}$ of queries (which are equivalent to some Kracht formulas) does not cover all the conjunctive queries answered by $\mathcal{ALC}$-concepts. This result can be regarded as *syntactic* extension to Kracht's theorem. Since we did not prove that the family $\mathcal{K}$ is exactly the restriction of the Kracht fragment to the set of conjunctive queries, we cannot conclude that the queries considered below form a *semantically* proper extension to the Kracht fragment. At the end of this section we also describe a wider family of queries answered by $\mathcal{ALCI}$-concepts.

First, we define the family of *parallel-serial* queries (or ps-queries, for short) with two *poles* x and y (where x is the distinguished and y a non-distinguished variable) inductively. A query $q(x) \leftarrow xRy$ is an *atomic* ps-query with the poles x and y. If $q_1(x) \leftarrow \alpha(x, y, \vec{u})$ and $q_2(x) \leftarrow \beta(x, y, \vec{v})$ are ps-queries with the poles x and y and the tuples $\vec{u}$ and $\vec{v}$ are disjoint, then $q(x) \leftarrow \alpha(x, y, \vec{u}) \wedge \beta(x, y, \vec{v})$ is a ps-query with the same poles x and y called a *parallel connection* of q_1 and q_2 and denoted as $q_1 \| q_2$. If $q_1(x) \leftarrow \alpha(x, y, \vec{u})$ is a ps-query with the poles x and y, $q_2(y) \leftarrow \beta(y, z, \vec{v})$ is a ps-query with the poles y and z, and α and β have only y as a common variable, then $q(x) \leftarrow \alpha(x, y, \vec{u}) \wedge \beta(y, z, \vec{v})$ is a ps-query with the poles x and z called a *serial connection* of q_1 and q_2 and denoted as $q_1 \circ q_2$.

To a ps-query $q(x)$ we associate a role expression $\mathcal{R}(q)$ built from role names using intersection and composition: for an atomic query $q(x) \leftarrow xRy$, we put $\mathcal{R}(q) := R$; and inductive steps are: $\mathcal{R}(q_1 \| q_2) := \mathcal{R}(q_1) \sqcap \mathcal{R}(q_2)$, $\mathcal{R}(q_1 \circ q_2) := \mathcal{R}(q_1) \circ \mathcal{R}(q_2)$. By induction, it is straightforward to show that any ps-query $q(x)$ is answered by (and even equivalent to) the $\mathcal{ALC}(\sqcap, \circ)$-concept $\exists \mathcal{R}(q).\top$ (note that this concept contains no concept names).

Unfortunately, for most DLs used in practice, this solution to query answering is unsatisfactory, since adding role intersection and composition to a logic (together with arbitrary concept and role inclusion axioms in a TBox) leads to undecidability. Therefore, our next aim is to find conditions under which a ps-query is answered by an $\mathcal{ALC}$-concept. For convenience, we extend Definition 4 to the case of two concepts: we say that concepts C and D are *similar* (written as $C \approx D$) if they have the same instances w.r.t. any knowledge base $\mathcal{KB}$, i.e., $\mathcal{KB} \models a\colon C \Leftrightarrow \mathcal{KB} \models a\colon D$, for any $a \in \mathsf{IN}$.

Lemma 12 (Elimination Lemma). *The concept $\exists(R \sqcap S).C$ is similar to the concept $\forall R.Y \to \exists S.(Y \sqcap C)$ (where Y is a fresh concept name). In general:*

$$\exists(R_0 \sqcap \ldots \sqcap R_n).C \;\approx\; \big(\forall R_1.Y_1 \sqcap \ldots \sqcap \forall R_n.Y_n \;\to\; \exists R_0.(Y_1 \sqcap \ldots \sqcap Y_n \sqcap C)\big).$$

Proof. The concept $\exists(R \sqcap S).C$ is equivalent to the conjunctive query $q(x) \leftarrow (xRy \wedge xSy \wedge x\colon C)$, which belongs to the family $\mathcal{K}$ from Sect. 4. Applying the algorithm described there, we obtain that $q(x)$ is answered by the concept $\forall R.Y \to \exists S.(Y \sqcap C)$. The proof for several roles is completely analogous. ⊣

Now we introduce a new family $\mathcal{Z}$ of queries: *it consists of queries whose relational part is a ps-query built up from atomic ones using parallel connections and restricted serial connections: $q_1 \circ q_2$ is allowed only if q_2 is atomic.*

Theorem 13. *Any query from the family $\mathcal{Z}$ is answered by a concept that can be built in linear time. Relational queries from $\mathcal{Z}$ are answered by $\mathcal{ALC}$-concepts.*

The proof is in the Appendix. The families $\mathcal{K}$ and $\mathcal{Z}$ are incomparable w.r.t. inclusion: indeed, the following query belongs to $\mathcal{K} \setminus \mathcal{Z}$:

$$\begin{array}{ll} \text{Query:} & q(x) \;\leftarrow\; xRy \wedge yRz \wedge yRw \wedge xPz \wedge xQw \\ \text{Concept:} & \forall P.Z \sqcap \forall Q.W \to \exists R.(\exists R.Z \sqcap \exists R.W) \end{array}$$

whereas the following one belongs to $\mathcal{Z} \setminus \mathcal{K}$ (but we do not know whether it is equivalent to any Kracht formula beyond the family $\mathcal{K}$):

$$\begin{array}{ll} \text{Query:} & q(x) \;\leftarrow\; xRy \wedge xSy \wedge yPw \wedge xRz \wedge xSz \wedge zQw \\ \text{Concept:} & \forall R.(Y \sqcap Z) \sqcap \forall S.(Y \to \forall P.W) \to \exists S.(Z \sqcap \exists Q.W) \end{array}$$

It is straightforward to "lift" Theorem 13 to modal logic, thus obtaining the statement: *any first-order formula $q(x)$ from the family $\mathcal{Z}$ locally corresponds to a modal formula that can be obtained efficiently.* This is a *syntactic* extension to Kracht's theorem (as already pointed out, we do not know if this is a *semantically* proper extension). At the same time, this can be regarded as an argument in favour of the (unproved) converse of Theorem 7: the family $\mathcal{Z}$ did not bring us a counterexample to the converse of Theorem 7.

Notice that all relational queries considered so far were answered by $\mathcal{ALC}$-concepts. What if we allow inverse roles? If we "forget" the direction of edges in the queries from the families $\mathcal{K}$ and $\mathcal{Z}$, then we arrive to the following family $\mathcal{Y}$ of queries (formulated in terms of the underlying graph of a query, whose nodes are variables and edges correspond to role atoms): *$\mathcal{Y}$ is the family of connected*

queries $q(x)$ without cycles (even non-oriented) consisting of non-distinguished variables only, i.e., every cycle in $q(x)$ must contain the distinguished variable x. We conclude with the following theorem (its proof is in the Appendix).

Theorem 14. *Any query from the family $\mathcal{Y}$ is answered by a concept that can be built in linear time. Relational queries from $\mathcal{Y}$ are answered by $\mathcal{ALCI}$-concepts. The family $\mathcal{Y}$ includes both families $\mathcal{K}$ and $\mathcal{Z}$, i.e., $\mathcal{K} \cup \mathcal{Z} \subset \mathcal{Y}$.*

6 The harvest

Here we present concrete examples of the results obtained. As already mentioned above, we can consider queries $q(x)$ without an atom of the form $x\colon A$.

Example 1. Since reflexivity is expressible by the modal formula $p \to \Diamond p$, we conclude that the query $q(x) \leftarrow xRx$ is answered (over any DL knowledge base) by the concept $X \to \exists R.X$. Similarly, the results from Sect. 4 yield the following (cyclic) queries and corresponding concepts:

Query	Concept
$q_1(x) \leftarrow xRy \wedge ySx$	$X \to \exists R.\exists S.X$
$q_2(x) \leftarrow xRy \wedge xSy$	$\forall R.Y \to \exists S.Y$
$q_3(x) \leftarrow xRy \wedge ySx \wedge y\colon B$	$X \to \exists R.(B \sqcap \exists S.X)$
$q_4(x) \leftarrow xRy \wedge xSy \wedge y\colon B$	$\forall R.Y \to \exists S.(B \sqcap Y)$

The query $q_1(x)$ can be easily generalised to the case of an oriented cycle of the length n starting and ending at x, and iterating $\exists R$ in the corresponding concept. Similarly, one can generalise the query $q_2(x)$, replacing R and S by chains of the length n and m both going from x to y, as shown in our next example.

Example 2. For any $m \geqslant 0$ and $n \geqslant 0$ with $m + n > 0$, the query[5]

$$\begin{array}{l} q(x) \leftarrow xR_1y_2 \wedge y_2R_2y_3 \wedge \ldots \wedge y_mR_my \wedge \\ \qquad\quad xS_1z_2 \wedge z_2S_2z_3 \wedge \ldots \wedge z_nS_ny \end{array}$$

is answered by an $\mathcal{ALC}$-concept $\forall R_1.\cdots\forall R_m.Y \to \exists S_1.\cdots\exists S_n.Y$, and the query

$$\begin{array}{l} q(x) \leftarrow xR_1y_2 \wedge y_2R_2y_3 \wedge \ldots \wedge y_mR_my \;\wedge\; y_2\colon B_2 \wedge \ldots \wedge y_m\colon B_m \wedge \\ \qquad\quad xS_1z_2 \wedge z_2S_2z_3 \wedge \ldots \wedge z_nS_ny \quad\wedge\; z_2\colon C_2 \wedge \ldots \wedge z_n\colon C_n \;\wedge y\colon D \end{array}$$

(with B_i, C_j, D being concepts in a language $\mathcal{L}$) is answered by the $\mathcal{L}$-concept

$$\begin{array}{l} \exists R_1.(B_2 \sqcap \exists R_2.(B_3 \sqcap \ldots (B_m \sqcap \exists R_m.\neg Y)\ldots)) \sqcup \\ \exists S_1.(C_2 \sqcap \exists S_2.(C_3 \sqcap \ldots (C_n \sqcap \exists S_n.(Y \sqcap D))\ldots)). \end{array}$$

The query $q_2(x)$ in Example 1 can be generalised in another direction: instead of two roles linking x with y, we can take several ones. This leads to the following example; and it is the first time where we need more than one fresh concept name.

[5] For $m = 0$ the first line in q is dropped and y replaced by x; and similarly for $n = 0$.

Example 3. A query $q(x) \leftarrow xR_0y \wedge \ldots \wedge xR_ny$ is answered by the $\mathcal{ALC}$-concept

$$\forall R_1.Y_1 \sqcap \ldots \sqcap \forall R_n.Y_n \;\rightarrow\; \exists R_0.(Y_1 \sqcap \ldots \sqcap Y_n).$$

Now if we stretch each R_i into a chain linking x with y, we obtain the following.

Example 4. The query of the form $q(x) \leftarrow q_0(x) \wedge \ldots \wedge q_n(x)$, where each $q_i(x)$ is a chain of m_i edges linking x with y:

$$q_i(x) \leftarrow\; xR_1^iy_2^i \wedge y_2^iR_2^iy_3^i \wedge \ldots \wedge y_{m_i}^iR_{m_i}^iy,$$

(superscripts are indices not powers) is answered by the $\mathcal{ALC}$-concept

$$\forall R_1.Y_1 \sqcap \ldots \sqcap \forall R_n.Y_n \;\rightarrow\; \exists R_0.(Y_1 \sqcap \ldots \sqcap Y_n),$$

where $\forall R_i$ stands for $\forall R_1^i.\cdots\forall R_{m_i}^i$ and similarly for $\exists R_0$. We leave adding concept atoms to this query as an exercise.

All these queries belong to the family $\mathcal{K}$ from introduced in Sect. 4: in the last example, the chain $q_0(x)$ can be regarded as a $\mathsf{Tree}(x,\vec{y})$, while the other chains $q_i(x)$ together constitute the conjunct $\alpha(x,\vec{y})$. An example of the query from the family $\mathcal{Z}$ (to be more exact, from $\mathcal{Z} \setminus \mathcal{K}$) was given after Theorem 13. Now we will illustrate how the algorithm presented in Sect. 4 works.

Example 5. Consider the query $q(x) \leftarrow \mathsf{Tree}(x,\vec{y}) \wedge \alpha(x,\vec{y}) \wedge \beta(x,\vec{y})$, where

$$\begin{aligned}
\mathsf{Tree}(x,\vec{y}) &\equiv xR_1y_1 \wedge y_1R_2y_2 \wedge y_1R_3y_3 \wedge y_1R_4y_4 \wedge y_4R_5y_5 \wedge y_4R_6y_6,\\
\alpha(x,\vec{y}) &\equiv xS_1y_1 \wedge xS_4y_6,\\
\beta(x,\vec{y}) &\equiv y_2S_2x \wedge y_5S_3x.
\end{aligned}$$

We introduce fresh concept names Y_{11} and Y_{46} for atoms in $\alpha(x,\vec{y})$, and X_{22} and X_{53} for atoms in $\beta(x,\vec{y})$. Next we build the expressions PRE and CON; they contain conjuncts that correspond to atoms in $\alpha(x,\vec{y})$ and $\beta(x,\vec{y})$:

$$\begin{aligned}
&\textsc{pre}: && x\colon \forall S_1.Y_{11} \wedge x\colon \forall S_4.Y_{46} \wedge x\colon X_{22} \wedge x\colon X_{53},\\
&\textsc{con}: && y_1\colon Y_{11} \wedge y_6\colon Y_{46} \wedge y_2\colon \exists S_2.X_{22} \wedge y_5\colon \exists S_3.X_{53}.
\end{aligned}$$

Then the expression $\textsc{pre} \rightarrow \exists\vec{y}\,(\mathsf{Tree}(x,\vec{y}) \wedge \textsc{con})$ will look as follows:

$$\begin{aligned}
&x\colon \big(\, \forall S_1.Y_{11} \sqcap \forall S_4.Y_{46} \sqcap X_{22} \sqcap X_{53} \,\big) \;\rightarrow\\
&(\exists y_1\overline{R}_1x)(\exists y_2\overline{R}_2y_1)(\exists y_3\overline{R}_3y_1)(\exists y_4\overline{R}_4y_1)(\exists y_5\overline{R}_5y_4)(\exists y_6\overline{R}_6y_4)\\
&\big(\, y_1\colon Y_{11} \wedge y_6\colon Y_{46} \wedge y_2\colon \exists S_2.X_{22} \wedge y_5\colon \exists S_3.X_{53} \,\big).
\end{aligned}$$

Now we roll up the restricted quantifiers, starting from the innermost one. After rolling-up the quantifiers first over y_6 and then over y_5, we will obtain:

$$\begin{aligned}
&x\colon \big(\, \forall S_1.Y_{11} \sqcap \forall S_4.Y_{46} \sqcap X_{22} \sqcap X_{53} \,\big) \;\rightarrow\\
&(\exists y_1\overline{R}_1x)(\exists y_2\overline{R}_2y_1)(\exists y_3\overline{R}_3y_1)(\exists y_4\overline{R}_4y_1)\\
&\big(\, y_1\colon Y_{11} \wedge y_2\colon \exists S_2.X_{22} \wedge y_4\colon (\exists R_6.Y_{46} \sqcap \exists R_5.\exists S_3.X_{53}) \,\big).
\end{aligned}$$

When all quantifiers are rolled-up, we end up with an expression of the form $x\colon (C \rightarrow D)$, where the concept $(C \rightarrow D)$ answering our original query $q(x)$ is:

$$\begin{aligned}
&\big(\, \forall S_1.Y_{11} \sqcap \forall S_4.Y_{46} \sqcap X_{22} \sqcap X_{53} \,\big) \;\rightarrow\\
&\exists R_1.\big(\, Y_{11} \sqcap \exists R_2.\exists S_2.X_{22} \sqcap \exists R_3.\top \sqcap \exists R_4.(\exists R_6.Y_{46} \sqcap \exists R_5.\exists S_3.X_{53}) \,\big).
\end{aligned}$$

7 Conclusions and outlook

One of the achievements of this paper is the established relationship between the notion of local correspondence from modal logic and the notion 'a concept answers a query' (Theorem 7). As an application, this enabled us to reuse the results from modal correspondence theory for answering a wide range of conjunctive queries with one distinguished variable. We have also found a syntactic (and probably semantic) extension to Kracht's theorem. Please note that, although at the first glace, only $\mathcal{ALC}$ or $\mathcal{ALCI}$ occur explicitly in Theorems 11,13,14, these should not be misunderstood as giving a query answering method for $\mathcal{ALC}$ or $\mathcal{ALCI}$ knowledge bases only. In fact, as we pointed out earlier, the results obtained provide us with algorithms for query answering over KBs formulated in any DL (extending $\mathcal{ALC}$), using as minimal means as possible (i.e., the resulting concept belongs to a simple logic $\mathcal{ALC}$ or $\mathcal{ALCI}$).

Analysing these families of queries, one can observe that the inherent complexity of query answering lies in the relational structure (i.e., the underlying graph) of the query. This structure was captured by concepts of $\mathcal{ALC}$ (Theorems 11 and 13) or $\mathcal{ALCI}$ (Theorem 14). A natural extension would be to invoke into this framework the correspondence theory for richer modal logics. The usage of qualified number restrictions (i.e., $\mathcal{ALCQ}$-concepts) for answering relational queries would involve the correspondence theory for graded modal logic. However, to the best of our knowledge, no general results similar to Sahlqvist's theorem are known for graded modal logic. Another possible direction is to extend this technique to DLs with relations of arbitrary arity, thus involving the correspondence theory for polyadic modal logics (which is explored extensively).

The need to investigate extensions stems from the restricted applicability of the technique we developed so far. For example, we have not succeeded to find an $\mathcal{ALC}$-concept answering the query $q(x) \leftarrow xRy \wedge ySy$. Moreover, if the converse of Theorem 7 holds (which is still an open question), then it would imply that such an $\mathcal{ALC}$-concept does not exist, since the property of frames expressed by $q(x)$ is not modally definable (see [2, Chap. 3], discussion after Corollary 3.16). At the same time, this and other queries shown in the table below can be answered by concepts involving extra role operations (whether they can be answered by any $\mathcal{ALCQI}$-concept is unknown):

Query	Concept
$q(x) \leftarrow xRy \wedge ySy$	$\exists R.\exists(S \sqcap id(\top)).\top$
$q(x) \leftarrow xRy \wedge ySz \wedge yPz$	$\exists R.\exists(S \sqcap P).\top$
$q(x) \leftarrow xRy \wedge xSz \wedge yPz \wedge zQy$	$\forall S.Z \rightarrow \exists R.\exists(P \sqcap Q^{-}).Z$
$q(x) \leftarrow xRyR'w \wedge xSzS'w \wedge yPz$	$\forall S.Z \rightarrow \exists R.\exists(P \sqcap (R' \circ S'^{-})).Z$

Finally, it is interesting whether all conjunctive queries can be answered by some concepts. As an example, for the following "tetrahedron" query

$$q(x) \;\leftarrow\; xRyR'w \wedge xSzS'w \wedge yPz \wedge xQw$$

we have neither found a concept in any DL that would answer it, nor proved that such a concept (in a certain DL) does not exist.

Acknowledgements

The research has been supported by EPSRC, Grant No. GR/S63168/01. The author would like to thank Dr. Ulrike Sattler and Dr. Ian Horrocks for guiding him in Description Logic and the help during the research. The author is also grateful to Birte Glimm for fruitful discussions on the subject of query answering and some useful suggestions.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
2. P. Blackburn, M. de Rijke, and Y. Venema, *Modal Logic*. Cambridge University Press, Theoretical Tracts in Computer Science, 2001.
3. D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'98)*, pp. 149–158, 1998.
4. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data Complexity of Query Answering in Description Logics. In *Proc. of the Int. Workshop on Description Logic (DL'05)*, 2005. (`ceur-ws.org`)
5. D. Calvanese, M. Magdalena Ortiz de la Fuente, T. Eiter, and E. Franconi. Data complexity of answering conjunctive queries over $\mathcal{SHIQ}$ knowledge bases. Technical Report. 2005.
6. B. Glimm and I. Horrocks, Handling cyclic conjunctive queries, In *Proc. of the Int. Workshop on Description Logic (DL'05)*, p. 219, 2005. (`ceur-ws.org`)
7. V. Haarslev and R. Möller. Racer: An OWL Reasoning Agent for the Semantic Web. In *Proc. of the Int. Workshop on Applications, Products and Services of Web-based Support Systems, in conj. with the 2003 IEEE/WIC Int. Conf. on Web Intelligence*, Halifax, Canada, pages 91–95, 2003. `http://www.racer-systems.com/`
8. I. Horrocks. FaCT and iFaCT. In *Proc. of the Int. Workshop on Description Logics (DL'99)*, pages 133–135, 1999. (`ceur-ws.org`)
9. I. Horrocks and U. Sattler. A Tableaux Decision Procedure for $\mathcal{SHOIQ}$. In *Proc. of 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, 2005. Morgan-Kaufmann Publishers (to appear).
10. I. Horrocks, U. Sattler, S. Tessaris, and S. Tobies. How to decide query containment under constraints using a Description Logic. In *Proc. of the 7th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'2000)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2000. Accompanying technical report: Query containment using a $\mathcal{DLR}$ ABox. LTCS-Report 99-15, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 1999.
11. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000.
12. I. Horrocks and S. Tessaris. A conjunctive query language for description logic ABoxes. In *Proc. of the 17th Nat. Conf. on Artificial Intelligence (AAAI'2000)*, pp. 399–404, 2000.
13. M. Kracht. How completeness and correspondence theory got married. In de Rijke, editor, *Diamonds and Defaults*, pp. 175–214. Kluwer, 1993.
14. C. Lutz. *The Complexity of Description Logic with Concrete Domains.* PhD Thesis, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 2002.

15. H. Sahlqvist. Correspondence and completeness in the first- and second-order semantics for modal logic. In S. Kanger, editor, *Proc. of the 3rd Scand. Logic Symp.*, Uppsala, 1973. North-Holland Publishing Company, Amsterdam, 1975.
16. K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pp. 466–471, 1991.
17. S. Tessaris. *Questions and answers: reasoning and querying in Description Logic.* PhD thesis, University of Manchester, 2001.
18. University of Maryland. Pellet OWL reasoner, 2003. Institute for Advanced Computer Studies, MIND LAB, The Semantic Web Research Group. `http://www.mindswap.org/2003/pellet/`

Appendix A: Proofs

Theorem 7 (Reduction). *If a unary relational query $q(x)$ locally corresponds to a modal formula φ, then the query $q(x)$ is answered by the $\mathcal{ALC}$-concept C_φ. In symbols: $q(x) \leftrightsquigarrow \varphi \Longrightarrow q(x) \approx C_\varphi$.*

Proof. Suppose that $q(x) \leftrightsquigarrow \varphi$. Then, given a knowledge base $\mathcal{KB}$ (in any DL containing $\mathcal{ALC}$, see the discussion after Definition 4) and a constant $a \in \mathsf{IN}$, we will prove the following equivalence: $\mathcal{KB} \models q(a) \Leftrightarrow \mathcal{KB} \models a\colon C_\varphi$.

($\Rightarrow$) Take any model $\mathcal{I}$ of $\mathcal{KB}$. By assumption, $\mathcal{I} \models q(a)$. We need to show that $\mathcal{I} \models a\colon C_\varphi$ (independently of how the fresh concept names X_i occurring in C_φ are interpreted in $\mathcal{I}$). Let F be the frame underlying $\mathcal{I}$ and denote $e := a^\mathcal{I}$. By definition, from $q(x) \leftrightsquigarrow \varphi$ it follows that, for these F and e, we have: $F \models q(e) \Leftrightarrow F, e \Vdash \varphi$. But we also know that $F \models q(e)$, because $\mathcal{I} \models q(a)$ and $q(x)$ is relational. Hence we conclude that $F, e \Vdash \varphi$, i.e., $e \in \varphi^\nu$ for any valuation ν.

Now take the valuation ν that is "read-off" from our interpretation $\mathcal{I}$ by putting $p_i^\nu := X_i^\mathcal{I}$, for all propositional letters p_i occurring in the formula φ. It is easily seen that $\varphi^\nu = C_\varphi^\mathcal{I}$, since C_φ is just a notational variant of φ, whereas ν and $\mathcal{I}$ are essentially the same. As shown above, $e \in \varphi^\nu$ for this valuation ν and so $a^\mathcal{I} = e \in \varphi^\nu = C_\varphi^\mathcal{I}$. Thus we have proven that $\mathcal{I} \models a\colon C_\varphi$.

($\Leftarrow$) Take any model $\mathcal{I}$ of $\mathcal{KB}$. By assumption, $\mathcal{I} \models a\colon C_\varphi$. Let F be a frame underlying $\mathcal{I}$ and $e := a^\mathcal{I}$. We need to show that $\mathcal{I} \models q(a)$; since $q(x)$ is relational, it suffices to show that $F \models q(e)$. Due to the assumption that $q(x) \leftrightsquigarrow \varphi$, it remains to show that $F, e \Vdash \varphi$.

To this end, take any valuation ν and show that $e \in \varphi^\nu$. Let $\mathcal{J}$ be an interpretation differing from $\mathcal{I}$ only in how it interprets the fresh concept names occurring in C_φ, namely it is "read-off" from the valuation ν by putting $X_i^\mathcal{J} := p_i^\nu$. Observe that $C_\varphi^\mathcal{J} = \varphi^\nu$. Since $\mathcal{J}$ and $\mathcal{I}$ agree on concept, role and individual names occurring in $\mathcal{KB}$, we conclude that $\mathcal{J} \models \mathcal{KB}$. Now we use our assumption $\mathcal{KB} \models a\colon C_\varphi$ to infer that $\mathcal{J} \models a\colon C_\varphi$, from which it follows that $e = a^\mathcal{I} = a^\mathcal{J} \in C_\varphi^\mathcal{J} = \varphi^\nu$.

This completes the proof of Theorem 7. ⊣

Lemma 8. *Suppose that a unary relational query $q(x)$ is answered by a concept C_φ, i.e., $q(x) \approx C_\varphi$, for some modal formula φ. Then for any frame F and its point e, the condition $F \models q(e)$ implies $F, e \Vdash \varphi$.*

PROOF. Assume that $F \models q(e)$. The query $q(x)$ has the form $\exists \vec{y} \bigwedge_{i=1}^{m} t_i(x, \vec{y})$, where t_i are role atoms. Then there exist $\vec{o} \in \Delta$ such that $F \models t_i(e, \vec{o})$ for all $i \leqslant m$. Now, take the *canonical* ABox $\mathcal{A}_q$ for $q(x)$, i.e., introduce new constants a_x and $\vec{a_y}$ and put $\mathcal{A}_q := \{t_i(a_x, \vec{a_y}) \mid 1 \leqslant i \leqslant m\}$, and consider a knowledge base $\mathcal{KB}_q := \langle \varnothing, \mathcal{A}_q \rangle$. Since $\mathcal{KB}_q \models \bigwedge_{i=1}^{m} t_i(a_x, \vec{a_y})$, we have $\mathcal{KB}_q \models \exists \vec{y} \bigwedge_{i=1}^{m} t_i(a_x, \vec{y})$, and hence $\mathcal{KB}_q \models q(a_x)$. Applying the condition of our Lemma $q(x) \approx C_\varphi$, we obtain that $\mathcal{KB}_q \models a_x : C_\varphi$.

To prove that $F, e \Vdash \varphi$, take an arbitrary valuation ν on the frame F and show that $e \in \varphi^\nu$. Let $\mathcal{I}$ be an interpretation based on F such that it is "read-off" from ν by putting $X_i^{\mathcal{I}} := p_i^\nu$, for all fresh concept names X_i occurring in C_φ, and extended to the new constants by putting $a_x^{\mathcal{I}} := e$ and $\vec{a_y}^{\mathcal{I}} := \vec{o}$. Since $\mathcal{I}$ is based on F and $F \models t_i(o, \vec{e})$, we have $\mathcal{I} \models t_i(a_x, \vec{a_y})$, for all $i \leqslant m$, and hence $\mathcal{I} \models \mathcal{KB}_q$. As shown above, $\mathcal{KB}_q \models a_x : C_\varphi$. Therefore, we conclude that $\mathcal{I} \models a_x : C_\varphi$ and finally $e = a_x^{\mathcal{I}} \in C_\varphi^{\mathcal{I}} = \varphi^\nu$. ⊣

Theorem 13. *Any query from the family $\mathcal{Z}$ is answered by a concept that can be built in linear time. Relational queries from $\mathcal{Z}$ are answered by $\mathcal{ALC}$-concepts.*

PROOF. First assume that $q(x)$ is a relational query from $\mathcal{Z}$. Then:

1) Build the role expression $\mathcal{R}(q)$ (see Sect. 5). As already pointed out, the query $q(x)$ is equivalent to the $\mathcal{ALC}(\sqcap, \circ)$-concept $\exists \mathcal{R}(q).\top$.

2) Starting from the concept $\exists \mathcal{R}(q).\top$, repeatedly apply the following rewriting rules (each application of the second rule introduces a new concept name X), until all role operations are eliminated and we arrive to an $\mathcal{ALC}$-concept that answers the query $q(x)$:

$$\begin{array}{lcl} D \sqcup \exists (R \circ S).C & \equiv & D \sqcup \exists R.\exists S.C \\ D \sqcup \exists (R \sqcap S).C & \approx & D \sqcup \exists R.\neg X \sqcup \exists S.(X \sqcap C) \end{array}$$

The second rule follows from the Elimination Lemma and the following observation: *if $C \approx D$ then $(C \sqcup E) \approx (D \sqcup E)$*. Since, in the definition of the family $\mathcal{Z}$, a serial connection $q_1 \circ q_2$ is allowed only if q_2 is atomic, each time the first rule is applied, S will be a role name. Hence we will never obtain an expression of the form $\exists (R \sqcap S).C$ inside another quantifier, and so will be able to eliminate all occurrences of role intersection using the second rule. Therefore, the algorithm always terminates and produces an $\mathcal{ALC}$-concept C such that $C \approx \exists \mathcal{R}(q).\top$. It is easily seen that the procedure works in linear time in size of $q(x)$.

Now suppose that $q(x)$ belongs to $\mathcal{Z}$ and contains concept atoms. Let $y : C$ be one of them, where y is a non-distinguished variable (as already mentioned above, we can always consider queries $q(x)$ without atoms of the form $x : A$). Then we introduce a new non-distinguished variable y' and replace the atom $y : C$ by $y(id(C))y'$ and each atom of the form yRz by $y'Rz$ (recall that $id(C)$ is a notation that goes back to the propositional dynamic logic, and it denotes the role with the semantics: $id(C)^{\mathcal{I}} := \{\langle e, e \rangle \mid e \in C^{\mathcal{I}}\}$). This can be done for all concept atoms in $q(x)$ and results in an equivalent relational query $q'(x)$ (in an extended language) that belongs to $\mathcal{Z}$. Now we apply the above algorithm and finally replace in the resulting concept each occurrence of $\exists id(C).D$ by the equivalent expression $(C \sqcap D)$. ⊣

Theorem 14. *Any query from the family $\mathcal{Y}$ is answered by a concept that can be built in linear time. Relational queries from $\mathcal{Y}$ are answered by $\mathcal{ALCI}$-concepts. The family $\mathcal{Y}$ includes both families $\mathcal{K}$ and $\mathcal{Z}$, i.e., $\mathcal{K} \cup \mathcal{Z} \subset \mathcal{Y}$.*

PROOF. Every query $q(x)$ in $\mathcal{K} \cup \mathcal{Z}$ is connected. Furthermore, if we remove from its graph the node x together with the edges incident to x, we will obtain an acyclic graph. This shows that $q(x) \in \mathcal{Y}$, hence the inclusion $\mathcal{K} \cup \mathcal{Z} \subset \mathcal{Y}$.

Now suppose we are given a query $q(x) \in \mathcal{Y}$. We will give a sketch of the proof that $q(x)$ can be represented as a query from "non-oriented" version of the family $\mathcal{K}$ (i.e., with edges oriented arbitrarily). First remove from its graph the node x together with the edges incident to x. This yields an acyclic non-oriented graph, i.e., a forest (a finite set of trees). Since the original graph (before removing x) was connected, it had edges linking x with the trees in this forest; take one edge per tree. Then the node x together with this forest and edges linking x with trees in this forest form a non-oriented tree $\mathsf{Tree}(x, \vec{y})$. Other edges can be considered as chains linking x with some non-distinguished variables. Thus, we have obtained a representation of the query $q(x)$ in the "non-oriented" version of $(\mathcal{K})$. Applying the algorithm described in Sect. 4 yields an $\mathcal{ALCI}$-concept that answers our query. ⊣

A tableau-based decision procedure for a branching-time interval temporal logic

Davide Bresolin and Angelo Montanari

Department of Mathematics and Computer Science,
University of Udine, Italy
E-mail: {bresolin|montana}@dimi.uniud.it

Abstract

Propositional interval temporal logics are very expressive temporal logics, with simple syntax and semantics, which allow one to naturally express statements that refer to time intervals and continuous processes. Most of them feature temporal operators that only allow one to express properties of a single timeline. In this paper we develop a branching-time interval neighborhood logic that interleaves operators that quantify over possible timelines with operators that quantify over intervals belonging to a given timeline. We define its syntax and semantics, and we provide it with a doubly-exponential tableau-based decision procedure.

1 Introduction

Propositional interval temporal logics are very expressive temporal logics, with simple syntax and semantics, which allow one to naturally express statements that refer to time intervals and continuous processes. Among them, we mention Halpern and Shoham's Modal Logic of Time Intervals (HS) [9], Venema's CDT logic [13], Moszkowski's Propositional Interval Temporal Logic (PITL) [11], and Goranko, Montanari, and Sciavicco's Propositional Neighborhood Logic (PNL), which is the propositional counterpart of Chaochen and Hansen's first-order Neighborhood Logic [2,6] (an up-to-date survey of the field can be found in [7]). Most of them feature temporal operators that only allow one to express properties of a single timeline, with the exception of Paech's Branching Regular Logic (BRL) [12]. BRL is a *local* branching-time interval logic, whose operators quantify over different timelines [1]. In [12] the author provides a Gentzen-style system for BRL and she states some expressiveness and complexity results.

[1] Locality forces a propositional variable to be true over an interval if and only if it is true at its starting point [11].

In this paper we develop a branching-time interval neighborhood logic that interleaves operators that quantify over possible timelines with operators that quantify over intervals belonging to a given timeline. We define its syntax and semantics, and we provide it with a doubly-exponential tableau-based decision procedure. Unlike the case of BRL, we do not impose any semantic restriction, such as locality, to get decidability.

When defining a temporal logic, there are basically two possible choices for the underlying temporal structure. Either time is *linear* (at any time there is only one possible future) or it has a *branching*, tree-like structure (any time may have many different futures). In the case of point-based temporal logics, both these alternatives have been successfully explored, and several meaningful logics have been developed (we only mention the linear temporal logic LTL and its many variants, and the branching-time temporal logics CTL and CTL* [4]). On the contrary, interval-based temporal logics are usually interpreted over linear temporal structures. Even those interval logics which are interpreted over branching-time temporal structures, such as Halpern and Shoham's HS (in its original formulation) and Goranko, Montanari, and Sciavicco's branching CDT (BCDT$^+$) [5], only feature temporal operators that express properties of single timelines.

The main difference between interval-based and point-based temporal logics is that the former can express properties of *pairs* of time points (think of intervals as constructed out of points), rather than *single* time points. In most cases, this feature leads to undecidability since it prevents one from the possibility of reducing interval-based temporal logics to point-based ones. However, by imposing suitable *syntactic and/or semantic restrictions*, such a reduction can be defined, thus allowing one to benefit from the good computational properties of point-based logics [10]. A possible syntactic restriction is the choice of a suitable subset of interval modalities. As for semantic restrictions, one possibility is to constrain the truth value of propositional variables over intervals. We already mention the locality assumption. Another simplifying assumption often made is that of homogeneity, which states that a propositional variable is true over an interval if and only it it is true over all of its subintervals. Another possibility is to achieve decidability by constraining the classes of interval structures over which formulas are interpreted. The problem of identifying expressive enough, yet decidable, *genuinely* interval-based logics, that is, logics which are not directly translated into point-based logics and not invoking semantic restrictions, is still largely unexplored.

While various tableau methods have been developed for linear and branching time point-based temporal logics, not much work has been done on tableau methods for interval-based temporal logics. One reason for this disparity is that operators of interval temporal logics are in many respects more difficult to deal with [8]. In [5,8], Goranko et al. outline a tableau method for Venema's CDT logic interpreted over partial orders (BCDT$^+$). The method can be easily adapted to variations and subsystems of BCDT$^+$, thus providing a

general tableau method for propositional interval logics. However, while most existing tableau methods for temporal logics are terminating methods for decidable logics, and thus they yield decision procedures, that for BCDT$^+$ only provides a semi-decision procedure for unsatisfiability. In [1], we propose a tableau-based decision procedure for the future fragment of (strict) Propositional Neighborhood Logic, that we call Right Propositional Neighborhood Logic (RPNL$^-$ for short), interpreted over natural numbers. By combining syntactic restrictions (future temporal operators) and semantic ones (the domain of natural numbers), we succeeded in developing a tableau-based decision procedure for RPNL$^-$. To the best of our knowledge, RPNL$^-$ is the first non-trivial case of a *genuine* propositional interval logic for which a tableau-based decision procedure has been given.

In this paper, we generalize the method to a suitable branching-time propositional interval temporal logic, that we call Branching-Time Right (Propositional) Neighborhood Logic (BTNL[R]$^-$ for short). Such a logic combines the interval neighborhood operators of RPNL$^-$ with the path quantifiers of CTL [4]. Emerson and Halpern's tableau-based decision procedure for CTL [3] consists of an initial construction phase, followed by an elimination phase. The elimination phase encompasses both a local pruning, that removes local inconsistencies, and a global pruning, that removes nodes including eventualities which are not fulfilled by the current graph. It turns out that a CTL formula is satisfiable if and only if the final graph is not empty. By combining such a tableau method for CTL and the one we developed for RPNL$^-$ [1], we have been able to devise a doubly-exponential tableau-based decision procedure for BTNL[R]$^-$.

The rest of the paper is organized as follows. In Section 2, we introduce the syntax and semantics of BTNL[R]$^-$. In Section 3, we present our decision procedure, we prove its soundness and completeness, and we address complexity issues. In Section 4, we show our procedure at work on a simple example. Conclusions provide an assessment of the work and outline future research directions.

2 The Logic BTNL[R]$^-$

2.1 Tree-like structures

According to a commonly accepted perspective [4], the underlying temporal structure of branching-time temporal logics has a branching-like nature where each time point may have many successors points. The structure of time thus corresponds to an infinite tree. We shall further assume that the timeline defined by every (infinite) path in the tree is isomorphic to $\langle \mathbb{N}, < \rangle$. We allow a node in the tree to have infinitely many (possibly, uncountably many) successors, while we require each node to have at least one successor. It will turn out that, as far as our logic is concerned, such trees are undistinguishable

from trees with finite branching.
Given a directed graph $\mathbb{D} = \langle D, R\rangle$, a *finite R-sequence* over $\mathbb{D}$ is a sequence of nodes $d_1 d_2 \dots d_n$, with $n \geq 2$ and $d_i \in D$ for $i = 1, \dots, n$, such that $R(d_i, d_{i+1})$ for $i = 1, \dots, n-1$. *Infinite R-sequences* can be defined analogously. We define a *path* π in $\mathbb{D}$ as a finite or infinite R-sequence. In the following, we shall take advantage of a relation $R^+ \subseteq D \times D$ such that $R^+(d_i, d_j)$ if and only if d_i and d_j are respectively the first and the last element of a finite R-sequence.
Temporal structures for branching time logics are infinite tree defined as follows.

Definition 2.1 An infinite *tree* is an infinite directed graph $\mathbb{D} = \langle D, R\rangle$, with a distinguished element $d_0 \in D$, called the *root* of the tree, where D is the set of nodes, called *time points*, and the set of edges $R \subseteq D \times D$ is a relation such that:

- for every $d(\neq d_0) \in D$, $R^+(d_0, d)$, that is, every $d(\neq d_0) \in D$ is R-reachable from d_0;
- for every $d(\neq d_0) \in D$, there exists at most one $d' \in D$ such that $R(d', d)$ (together with the previous one, this condition guarantees that every $d(\neq d_0) \in D$ has exactly one R-predecessor);
- there exists no d' such that $R(d', d_0)$, that is, d_0 has no R-predecessors;
- for every $d(\neq d_0) \in D$, there exists at least one $d' \in D$ such that $R(d, d')$, that is, every $d \in D$ has at least one R-successor.

It is not difficult to show that infinite trees are *acyclic* graphs, that is, there exist no finite paths which start from and end at the same node.
Given an infinite tree $\mathbb{D} = \langle D, R\rangle$, we can define a partial order $<$ over D such that, for every $d, d' \in D$, $d < d'$ if and only if $R^+(d, d')$. It is immediate to see that, for every infinite path π in $\mathbb{D}$, $\langle \pi, <\rangle$ is isomorphic to $\langle \mathbb{N}, <\rangle$.
Intervals over infinite trees are defined as follows. A *(strict) interval* over $\mathbb{D} = \langle D, R\rangle$ is a pair $[d_i, d_j]$ such that $d_i, d_j \in D$ and $d_i < d_j$. We denote the set of all strict intervals over a tree $\mathbb{D}$ as $\mathbb{I}(\mathbb{D})^-$. For any pair of intervals $[d_i, d_j]$ and $[d_j, d_k]$, we say that $[d_j, d_k]$ is a *right neighbor* of $[d_i, d_j]$. A point $d \in D$ belongs to the interval $[d_i, d_j]$ if $d_i \leq d \leq d_j$. In the following, we shall interpret our logic over *interval structures* $\langle \mathbb{D}, \mathbb{I}(\mathbb{D})^-\rangle$, where $\mathbb{D}$ is an infinite tree and $\mathbb{I}(\mathbb{D})^-$ is the set of all strict intervals over it.

2.2 Syntax and semantics of BTNL[R]$^-$

In this section, we give syntax and semantics of BTNL[R]$^-$ interpreted over infinite trees. BTNL[R]$^-$ is a propositional interval temporal logic based on the neighborhood relation between intervals. Its formulas are built from a set AP of propositional letters $p, q, \dots$, by using the Boolean connectives $\neg$ and $\vee$ and the future temporal operators $E\langle A\rangle$ and $E[A]$. The other classical propositional connectives, as well as the logical constants $\top$ (true) and $\bot$

(false), are defined in the usual way. Furthermore, we introduce the temporal operator $A[A]$ as a shorthand for $\neg E\langle A\rangle\neg$ and the temporal operator $A\langle A\rangle$ as a shorthand for $\neg E[A]\neg$, and, from now on, we identify $\neg E\langle A\rangle\psi$ with $A[A]\neg\psi$ and $\neg E[A]\psi$ with $A\langle A\rangle\neg\psi$.
Formulas of BTNL[R]$^-$, denoted by $\varphi, \psi, \ldots$, are recursively defined by the following grammar:

$$\varphi = p \mid \neg\varphi \mid \varphi \vee \varphi \mid E\langle A\rangle\varphi \mid E[A]\varphi.$$

We denote by $|\varphi|$ the size of φ, that is, the number of symbols in φ, where the quantifier $E\langle A\rangle$ (resp., $E[A]$) is counted as one symbol. Whenever there are no ambiguities, we call a BTNL[R]$^-$ formula just a formula. A formula of the form $E\langle A\rangle\psi$, $E[A]\psi$, $A\langle A\rangle\psi$ or $A[A]\psi$, is called a *temporal formula*. A temporal formula whose main temporal operator is either $E\langle A\rangle$ or $E[A]$ is called an *existential formula*, while a temporal formula whose main temporal operator is either $A\langle A\rangle$ or $A[A]$ is called a *universal formula*.

A *model* for a formula is a tuple $\mathbf{M}^- = \langle\langle\mathbb{D}, \mathbb{I}^-(\mathbb{D})\rangle, \mathcal{V}\rangle$, where the pair $\langle\mathbb{D}, \mathbb{I}(\mathbb{D})^-\rangle$ is an interval structure, and $\mathcal{V} : \mathbb{I}(\mathbb{D})^- \longrightarrow 2^{AP}$ is a *valuation function* that assigns to every interval the set of propositional letters true on it. We place ourselves in the most general setting, and we do not impose any locality or monotonicity constraint on the valuation function. As an example, it may happen that $\mathcal{V}$ maps an interval $[d_i, d_j]$ to some set of atomic propositions and that it maps an interval contained in/that contains $[d_i, d_j]$ to a different set.

The semantics of BTNL[R]$^-$ is defined recursively by the *satisfiability relation* $\Vdash$ as follows. Let $\mathbf{M}^- = \langle\langle\mathbb{D}, \mathbb{I}^-(\mathbb{D})\rangle, \mathcal{V}\rangle$ be some given model, and let $[d_i, d_j] \in \mathbb{I}(\mathbb{D})^-$:

- for every propositional letter $p \in AP$, $\mathbf{M}^-, [d_i, d_j] \Vdash p$ iff $p \in \mathcal{V}([d_i, d_j])$;
- $\mathbf{M}^-, [d_i, d_j] \Vdash \neg\psi$ iff $\mathbf{M}^-, [d_i, d_j] \nVdash \psi$;
- $\mathbf{M}^-, [d_i, d_j] \Vdash \psi_1 \vee \psi_2$ iff $\mathbf{M}^-, [d_i, d_j] \Vdash \psi_1$, or $\mathbf{M}^-, [d_i, d_j] \Vdash \psi_2$;
- $\mathbf{M}^-, [d_i, d_j] \Vdash E\langle A\rangle\psi$ iff there exists $d_k \in D$, $d_j < d_k$, such that $\mathbf{M}^-, [d_j, d_k] \Vdash \psi$;
- $\mathbf{M}^-, [d_i, d_j] \Vdash E[A]\psi$ iff there exists an infinite path $\pi = d_j d_{j+1} \ldots$ rooted at d_j such that, for every d_k in π, with $d_j < d_k$, $\mathbf{M}^-, [d_j, d_k] \Vdash \psi$.

Since our logic has only future time operators, without loss of generality, we can restrict our attention to *standard trees*, namely, trees such that the root d_0 has only one R-successor d_1. In such a case, the corresponding interval structure has an *initial interval* $[d_0, d_1]$, and we say that a formula φ is satisfied by the interval structure if and only if φ holds at the initial interval.

3 A tableau-based decision procedure for BTNL[R]$^-$

To check the satisfiability of a formula φ, we build a graph, that we call a *tableau* for φ, whose nodes represent points of the temporal domain $\mathbb{D}$ and

whose edges represent the relation R connecting a point to its successors in the tree. We shall take advantage of such a construction to reduce the problem of finding a model for φ to the problem of testing whether the tableau satisfies some suitable properties or not.

Let $\mathbf{M}^- = \langle\langle \mathbb{D}, \mathbb{I}^-(\mathbb{D})\rangle, \mathcal{V}\rangle$ be a model for φ and let d_j be a point in $\mathbb{D}$. We have that, given an interval $[d_i, d_j]$ ending in d_j, every right neighbor of it is also a right neighbor of every other interval $[d_k, d_j]$ ending in d_j. Hence, every *temporal formula* φ holds over $[d_i, d_j]$ if and only if it holds over every other interval $[d_k, d_j]$ ending in d_j. We denote by $\mathrm{REQ}(d_j)$ the set of temporal formulas which hold over all intervals ending in d_j.

The building blocks for the tableau construction are φ-atoms. For every interval $[d_i, d_j] \in \mathbb{I}^-(\mathbb{D})$, we introduce a pair of sets of formulas $(\mathcal{A}_{[d_i,d_j]}, \mathcal{C}_{[d_i,d_j]})$, that we call an *atom* for φ (φ-atom for short). The set $\mathcal{A}_{[d_i,d_j]}$ is a subset of $\mathrm{REQ}(d_i)$, which collects the set of requests in $\mathrm{REQ}(d_i)$ relevant to the interval $[d_i, d_j]$. In general, $\mathcal{A}_{[d_i,d_j]}$ may differ from $\mathcal{A}_{[d_i,d_k]}$ for $j \neq k$. The set $\mathcal{C}_{[d_i,d_j]}$ contains all and only the formulas that (should) hold over $[d_i, d_j]$. We can associate with every point $d_j \in D$ the set of φ-atoms $\{(\mathcal{A}_{[d_i,d_j]}, \mathcal{C}_{[d_i,d_j]}) : d_i < d_j\}$, which includes all φ-atoms paired with intervals ending in d_j. These sets of atoms are the (macro)nodes of the tableau for φ.

3.1 Basic notions

Let φ be a BTNL[R]$^-$-formula to be checked for satisfiability and let AP be the set of its propositional variables.

Definition 3.1 The *closure* $\mathrm{CL}(\varphi)$ of φ is the set of all subformulas of φ and their negations (we identify $\neg\neg\psi$ with ψ).

Definition 3.2 The set of *temporal requests* of φ is the set $\mathrm{TF}(\varphi)$ of all temporal formulas in $\mathrm{CL}(\varphi)$.

By induction on the structure of φ, it can be easily proved that $|\,\mathrm{CL}(\varphi)| \leq 2{\cdot}|\varphi|$ and $|\,\mathrm{TF}(\varphi)| \leq 2 \cdot |\varphi|$.

We are now ready to introduce the key notion of φ-atom.

Definition 3.3 Let φ be a BTNL[R]$^-$-formula. A φ-*atom* is a pair $(\mathcal{A}, \mathcal{C})$, with $\mathcal{A} \subseteq \mathrm{TF}(\varphi)$ and $\mathcal{C} \subseteq \mathrm{CL}(\varphi)$, such that:

- for every $\psi \in \mathrm{CL}(\varphi)$, $\psi \in \mathcal{C}$ iff $\neg\psi \notin \mathcal{C}$;
- for every $\psi_1 \vee \psi_2 \in \mathrm{CL}(\varphi)$, $\psi_1 \vee \psi_2 \in \mathcal{C}$ iff $\psi_1 \in \mathcal{C}$ or $\psi_2 \in \mathcal{C}$;
- for every $\psi \in \mathrm{TF}(\varphi)$, if $\psi \in \mathcal{A}$ then $\neg\psi \notin \mathcal{A}$;
- for every $A[A]\psi \in \mathcal{A}$, $\psi \in \mathcal{C}$;
- for every $E[A]\psi \in \mathcal{A}$, $\psi \in \mathcal{C}$.

Formulas in $\mathcal{C}$ are called *current formulas*, while (temporal) formulas in $\mathcal{A}$ are called *active requests*.

As for the set $\mathcal{A}$, it is worth pointing out that there may exist a φ-atom $(\mathcal{A}, \mathcal{C})$

and a temporal formula ψ such that neither ψ nor $\neg\psi$ belongs to $\mathcal{A}$.
We denote the set of all φ-atoms by $\mathcal{A}_\varphi$. It is not difficult to show that $|\mathcal{A}_\varphi| \leq 2^{2|\varphi|}$.
Atoms come into play in the proposed tableau method as follows. The method associates an atom $(\mathcal{A}, \mathcal{C})$ with any interval $[d_i, d_j]$. The set $\mathcal{A}$ includes all formulas of the form $A[A]\psi$ that belong to $\mathrm{REQ}(d_i)$ as well as some formulas of the forms $E[A]\psi$, $A\langle A\rangle\psi$, and $E\langle A\rangle\psi$ in $\mathrm{REQ}(d_i)$; the set $\mathcal{C}$ includes all formulas $\psi \in \mathrm{CL}(\varphi)$ which (should) hold over $[d_i, d_j]$. Moreover, for all formulas of the forms $A[A]\psi$ and $E[A]\psi$ in $\mathcal{A}$, we put ψ into $\mathcal{C}$, while for any formula of the forms $A\langle A\rangle\psi$ and $E\langle A\rangle\psi$ in $\mathcal{A}$, it may happen that $\psi \in \mathcal{C}$, but this is not necessarily the case.

Atoms are connected by the following binary relation.

Definition 3.4 Let X_φ be a binary relation over $\mathcal{A}_\varphi$ such that, for every pair of atoms $(\mathcal{A}, \mathcal{C}), (\mathcal{A}', \mathcal{C}') \in \mathcal{A}_\varphi$, $(\mathcal{A}, \mathcal{C}) X_\varphi (\mathcal{A}', \mathcal{C}')$ if (and only if):

- $\mathcal{A}' \subseteq \mathcal{A}$;
- for every $A[A]\psi \in \mathcal{A}$, $A[A]\psi \in \mathcal{A}'$;
- for every $A\langle A\rangle\psi \in \mathcal{A}$, $A\langle A\rangle\psi \in \mathcal{A}'$ iff $\neg\psi \in \mathcal{C}$.

In the next section we shall show that for any pair of points $d_i < d_j$, the relation X_φ connects the atom associated with the interval $[d_i, d_j]$ to the atom associated with the interval $[d_i, d_{j+1}]$, where d_{j+1} is an R-successor of d_j. In particular, it will turn out that, if $(\mathcal{A}, \mathcal{C})$ is associated with the interval $[d_i, d_j]$, then for every formula $A[A]\psi \in \mathrm{REQ}(d_i)$ and every atom $(\mathcal{A}', \mathcal{C}')$ such that $(\mathcal{A}, \mathcal{C}) X_\varphi (\mathcal{A}', \mathcal{C}')$, we have that $A[A]\psi \in \mathcal{A}'$ (and thus $\psi \in \mathcal{C}'$), while for every formula $A\langle A\rangle\psi \in \mathcal{A}$, if $[d_i, d_j]$ satisfies ψ then $A\langle A\rangle\psi \notin \mathcal{A}'$. This guarantees that temporal requests of the form $A[A]\psi$ are propagated through X_φ-successors, while temporal requests of the form $A\langle A\rangle\psi$ are discarded once that ψ has been satisfied by the set of current formulas of some atom.

3.2 Tableau construction

Definition 3.5 A *node* is a set N of φ-atoms such that, for any pair $(\mathcal{A}, \mathcal{C})$, $(\mathcal{A}', \mathcal{C}') \in N$ and any $\psi \in \mathrm{TF}(\varphi)$, $\psi \in \mathcal{C} \;\Leftrightarrow\; \psi \in \mathcal{C}'$. We denote by $\mathcal{N}_\varphi$ the set of all nodes that can be built from $\mathcal{A}_\varphi$ and by $Init(\mathcal{N}_\varphi)$ the subset of all *initial nodes*, that is, the set $\{\{(\emptyset, \mathcal{C})\} \in \mathcal{N}_\varphi : \varphi \in \mathcal{C}\}$. Furthermore, for any node N, we denote by $\mathrm{REQ}(N)$ the set $\{\psi \in \mathrm{TF}(\varphi) : \exists(\mathcal{A}, \mathcal{C}) \in N(\psi \in \mathcal{C})\}$ (or, equivalently, the set $\{\psi \in \mathrm{TF}(\varphi) : \forall(\mathcal{A}, \mathcal{C}) \in N(\psi \in \mathcal{C})\}$).

From Definition 3.5, it follows that $|\mathcal{N}_\varphi| \leq 2^{2^{2\cdot|\varphi|}}$.

Nodes can be viewed as (maximal) collections of intervals ending at the same point of the temporal domain, that is, we can associate every node N with a point $d_j \in D$ and every atom $(\mathcal{A}, \mathcal{C}) \in N$ with some interval $[d_i, d_j]$ ending in d_j. Accordingly, we have that $\mathrm{REQ}(N) = \mathrm{REQ}(d_j)$. The relation between a node N, associated with point d_j, and a node M, associated with an R-

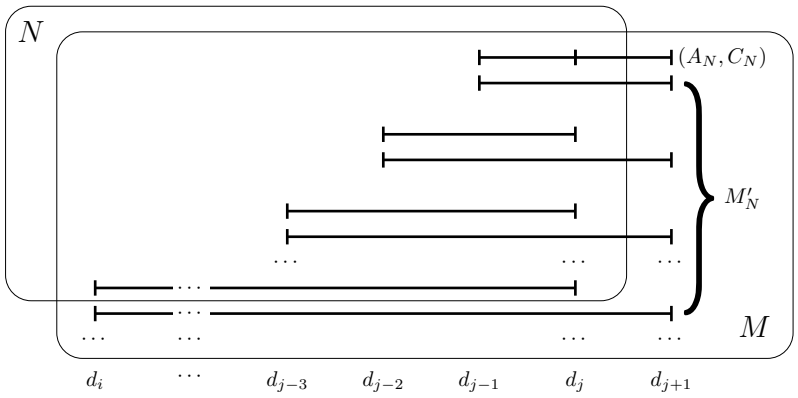


Figure 1. Connecting two nodes.

successor d_{j+1} of d_j, as well as the relations between intervals ending in d_j and intervals ending in d_{j+1} (and, thus, between atoms in N and atoms in M), are graphically depicted in Figure 1. We have that *(i)* the interval $[d_j, d_{j+1}]$ is a right neighbor of all intervals $[d_i, d_j]$ ending in d_j, and *(ii)* for every interval $[d_i, d_j]$ there exists an interval $[d_i, d_{j+1}]$. Thus, M should contain:

- an atom $(\mathcal{A}_N, \mathcal{C}_N)$ associated to the interval $[d_j, d_{j+1}]$;
- for every atom $(\mathcal{A}, \mathcal{C}) \in N$, an atom $(\mathcal{A}', \mathcal{C}')$ such that $(\mathcal{A}, \mathcal{C}) X_\varphi (\mathcal{A}', \mathcal{C}')$.

Definition 3.6 The *tableau* for a BTNL[R]$^-$-formula φ is a (finite) directed graph $\mathcal{T}_\varphi = \langle \mathcal{N}_\varphi, \mathcal{R}_\varphi \rangle$, where for any pair $N, M \in \mathcal{N}_\varphi$, $(N, M) \in \mathcal{R}_\varphi$ if and only if $M = \{(\mathcal{A}_N, \mathcal{C}_N)\} \cup M'_N$, where

1. $(\mathcal{A}_N, \mathcal{C}_N)$ is an atom such that $\mathcal{A}_N \subseteq \mathrm{REQ}(N)$ and for all universal formulas $\psi \in \mathrm{REQ}(N)$, $\psi \in \mathcal{A}_N$;
2. for every $(\mathcal{A}, \mathcal{C}) \in N$, there exists $(\mathcal{A}', \mathcal{C}') \in M'_N$ such that $(\mathcal{A}, \mathcal{C}) X_\varphi (\mathcal{A}', \mathcal{C}')$;
3. for every $(\mathcal{A}', \mathcal{C}') \in M'_N$, there exists $(\mathcal{A}, \mathcal{C}) \in N$ such that $(\mathcal{A}, \mathcal{C}) X_\varphi (\mathcal{A}', \mathcal{C}')$.

Let $N, M \in \mathcal{N}_\varphi$. If $(N, M) \in \mathcal{R}_\varphi$, we say that M is an $\mathcal{R}_\varphi$-*successor* of N. We say that M is an $\mathcal{R}_\varphi$-*descendant* of N if there exists a (finite) path from N to M in $\mathcal{T}_\varphi$.

Definition 3.7 Given a (finite or infinite) path $\pi = N_1 N_2 \ldots$ in $\mathcal{T}_\varphi$, an *atom path* in π is a sequence of atoms $(\mathcal{A}_1, \mathcal{C}_1)(\mathcal{A}_2, \mathcal{C}_2) \ldots$ such that:

- for every $i \geq 1$, $(\mathcal{A}_i, \mathcal{C}_i) \in N_i$;
- for every $1 \leq i (< n)$, $(\mathcal{A}_i, \mathcal{C}_i) X_\varphi (\mathcal{A}_{i+1}, \mathcal{C}_{i+1})$.

Given a node N and an atom $(\mathcal{A}, \mathcal{C}) \in N$, we say that the atom $(\mathcal{A}', \mathcal{C}')$ is an X_φ-*descendant* of $(\mathcal{A}, \mathcal{C})$ if and only if there exists a node M such that $(\mathcal{A}', \mathcal{C}') \in M$, and there exists a path π from N to M such that there is an atom path from $(\mathcal{A}, \mathcal{C})$ to $(\mathcal{A}', \mathcal{C}')$ in π.

Definition 3.8 An infinite path $\pi = N_1 N_2 \ldots$ in $\mathcal{T}_\varphi$ is a *fulfilling path* if and only if, for every $i \geq 1$, every atom $(\mathcal{A}, \mathcal{C}) \in N_i$, and every formula $A\langle A \rangle \psi \in \mathcal{A}$, either $\psi \in \mathcal{C}$ or there exist N_j, with $j > i$, and $(\mathcal{A}', \mathcal{C}') \in N_j$ such that $(\mathcal{A}', \mathcal{C}')$ is an X_φ-descendant of $(\mathcal{A}, \mathcal{C})$ in π and $\psi \in \mathcal{C}'$.

Definition 3.9 A *substructure* is a subgraph $\langle \mathcal{N}, \mathcal{R} \rangle \subseteq \mathcal{T}_\varphi$ such that:

- there exists a node $N_0 \in \mathcal{N} \cap Init(\mathcal{N}_\varphi)$ (*initial node*) such that all other nodes in $\mathcal{N}$ are $\mathcal{R}$-reachable from it;
- for every node $N \in \mathcal{N}$, there exists a fulfilling path (in $\langle \mathcal{N}, \mathcal{R} \rangle$) starting from N.

Substructures represent *candidate models* for φ. The truth of formulas devoid of temporal operators and of formulas of the form $A[A]\psi$, indeed, follows from Definition 3.3. Moreover, the truth of formulas of the form $A\langle A \rangle \psi$ follows from Definition 3.8. However, to obtain a model for φ we must also guarantee the truth of formulas of the forms $E[A]\psi$ and $E\langle A \rangle \psi$. To this end, we introduce the notion of *fulfilling substructure*.

Definition 3.10 A substructure $\langle \mathcal{N}, \mathcal{R} \rangle \subseteq \mathcal{T}_\varphi$ is *fulfilling* if and only if, for every node $N \in \mathcal{N}$ and every atom $(\mathcal{A}, \mathcal{C}) \in N$, the following conditions hold:

(F1) for every existential formula $\psi \in \mathcal{C}$, there exist an $\mathcal{R}$-successor M of N and an atom $(\mathcal{A}', \mathcal{C}') \in M$ such that $\psi \in \mathcal{A}'$ and $\mathcal{A}' \subseteq \mathrm{REQ}(N)$, and for all universal formulas $\xi \in \mathrm{REQ}(N)$, $\xi \in \mathcal{A}'$;

(F2) for every formula $E\langle A \rangle \psi \in \mathcal{A}$, either $\psi \in \mathcal{C}$ or there exist an $\mathcal{R}$-descendant M of N and an X_φ-descendant $(\mathcal{A}', \mathcal{C}')$ of $(\mathcal{A}, \mathcal{C})$ in M such that $\psi \in \mathcal{C}'$;

(F3) for every formula $E[A]\psi \in \mathcal{A}$, there exist a fulfilling path $\pi = N_0 N_1 N_2 \ldots$ and an atom path $(\mathcal{A}_0, \mathcal{C}_0)(\mathcal{A}_1, \mathcal{C}_1)(\mathcal{A}_2, \mathcal{C}_2) \ldots$ in π such that: *(i)* $(\mathcal{A}_0, \mathcal{C}_0) = (\mathcal{A}, \mathcal{C})$; *(ii)* $N_0 = N$; *(iii)* for every $i \geq 0$, $E[A]\psi \in \mathcal{A}_i$; *(iv)* for every formula $A\langle A \rangle \theta \in \mathcal{A}_0$, there exists $j \geq 0$ such that $\theta \in \mathcal{C}_j$.

Theorem 3.11 *If the formula φ is satisfiable (in a standard tree), then there exists a fulfilling substructure $\langle \mathcal{N}, \mathcal{R} \rangle \subseteq \mathcal{T}_\varphi$.*

Proof Let $\mathbf{M}^- = \langle \langle \mathbb{D}, \mathbb{I}(\mathbb{D})^- \rangle, \mathcal{V} \rangle$ be a model for φ such that $\mathbb{D} = \langle D, R \rangle$, with distinguished element d_0, is a standard tree. For every interval $[d_i, d_j] \in \mathbb{I}(\mathbb{D})^-$, we define an atom $(\mathcal{A}_{[d_i,d_j]}, \mathcal{C}_{[d_i,d_j]})$ as follows.

- $\mathcal{A}_{[d_i,d_j]}$ contains exactly:
  - all formulas $A[A]\psi \in \mathrm{REQ}(d_i)$;
  - all formulas $A\langle A \rangle \psi \in \mathrm{REQ}(d_i)$ such that, for every $d_i < d_l < d_j$, $\mathbf{M}^-, [d_i, d_l] \Vdash \neg\psi$;
  - all formulas $E[A]\psi \in \mathrm{REQ}(d_i)$ such that there exists an infinite path $\pi = d_i d_{i+1} \ldots d_j d_{j+1} \ldots$, starting from d_i and containing d_j, such that $\mathbf{M}^-, [d_i, d_k] \Vdash \psi$ for every $d_k \in \pi$;
  - all formulas $E\langle A \rangle \psi \in \mathrm{REQ}(d_i)$ such that there exists $d_k \geq d_j$ such that $\mathbf{M}^-, [d_i, d_k] \Vdash \psi$ and, for every $d_i < d_l < d_k$, $\mathbf{M}^-, [d_i, d_l] \Vdash \neg\psi$;
- $\mathcal{C}_{[d_i,d_j]}$ contains exactly all formulas $\psi \in \mathrm{CL}(\varphi)$ such that $\mathbf{M}^-, [d_i, d_j] \Vdash \psi$.

It is easy to check that, for every $[d_i, d_j] \in \mathbb{I}(\mathbb{D})^-$ and for every R-successor d_{j+1} of d_j, $(\mathcal{A}_{[d_i,d_j]}, \mathcal{C}_{[d_i,d_j]})$ is an atom such that $(\mathcal{A}_{[d_i,d_j]}, \mathcal{C}_{[d_i,d_j]}) X_\varphi (\mathcal{A}_{[d_i,d_{j+1}]},$

$\mathcal{C}_{[d_i,d_{j+1}]}$).

For every $d_j \in D$, with $d_j \neq d_0$, let $N_j = \{(\mathcal{A}_{[d_i,d_j]}, \mathcal{C}_{[d_i,d_j]}) : d_i < d_j\}$, $\mathcal{N} = \{N_j : d_j \in D, d_j \neq d_0\}$, and $\mathcal{R} = \mathcal{R}_\varphi \cap (\mathcal{N} \times \mathcal{N})$. It is easy to check that for all $d_j \neq d_0$, N_j is a node and that $\langle \mathcal{N}, \mathcal{R} \rangle$ is a fulfilling substructure. $\Box$

The next theorem shows that a model for φ can be obtained by unfolding a fulfilling substructure $\langle \mathcal{N}, \mathcal{R} \rangle$, starting from its initial node N_0.

Theorem 3.12 *If there exists a fulfilling substructure $\langle \mathcal{N}, \mathcal{R} \rangle \subseteq \mathcal{T}_\varphi$, then the formula φ is satisfiable.*

Proof Let $\langle \mathcal{N}, \mathcal{R} \rangle$ be a fulfilling substructure. To define a model for φ, we first build an infinite tree $\mathbb{D} = \langle D, R \rangle$ by *unfolding* the fulfilling substructure $\langle \mathcal{N}, \mathcal{R} \rangle$ from its initial node N_1 as follows:

- D is the (infinite) set of all finite $\mathcal{R}$-sequences $N_1 N_2 \dots N_k$ starting from the initial node N_1, including the empty sequence ε (the root of the tree);
- R is such that, for any pair of points $d, d' \in D$, $(d, d') \in R$ if and only if $d = N_1 \dots N_k$ and $d' = N_1 \dots N_k N_{k+1}$ and N_{k+1} is an $\mathcal{R}$-successor of N_k.

Notice that $\mathbb{D}$ is a standard tree with distinguished element $d_0 = \varepsilon$, since the only successor of d_0 is $d_1 = N_1$.

Given an arbitrary order of the nodes of $\mathcal{N}$, we define a *total* order $\lhd$ over finite $\mathcal{R}$-sequences, that is, on points of D, as follows:

- given two $\mathcal{R}$-sequences d and d' such that the length of d is less than the length of d', we have that $d \lhd d'$;
- given two $\mathcal{R}$-sequences d and d' of the same length, $d \lhd d'$ if and only if d precedes d' on the lexicographical order based on the given (arbitrary) order of nodes of $\mathcal{N}$.

In order to build a model for φ, we define a suitable (partial) labelling function $L : \mathbb{I}(\mathbb{D})^- \to A_\varphi$. For any $d_j = N_1 \dots N_k$, L associates an atom $(\mathcal{A}, \mathcal{C}) \in N_k$ with any interval $[d_i, d_j]$. We define L by (infinite) induction on the total order $\lhd$.

Base case. We start by defining the labelling of the initial interval $[d_0, d_1]$ (where $d_0 = \varepsilon$ and $d_1 = N_1$). Since N_1 is the initial node of $\langle \mathcal{N}, \mathcal{R} \rangle$, we have that $N_1 = \{(\emptyset, \mathcal{C})\}$, with $\varphi \in \mathcal{C}$. We put $L([d_0, d_1]) = (\emptyset, \mathcal{C})$. Two cases may arise.

$\mathcal{C}$ contains no existential formulas. In such a case, we define the labelling of an infinite branch starting from d_1 (remind that we admit only infinite models). Since $\langle \mathcal{N}, \mathcal{R} \rangle$ is a substructure, there exists a fulfilling path $N_1 N_2 \dots$ starting from N_1. Let $\pi = d_1 d_2 \dots$ be the corresponding infinite branch in $\mathbb{D}$ (where, for every $i \geq 1$, $d_i = N_1 \dots N_i$). We define the labelling $L([d_i, d_j])$ of every interval $[d_i, d_j]$, with $d_i, d_j \in \pi$, in such a way that:

- for all $j > 1$, $L([d_{j-1}, d_j]) = (\mathcal{A}', \mathcal{C}')$ is such that $\mathcal{A}' \subseteq \mathrm{REQ}(N_{j-1})$ and, for all universal formulas $\theta \in \mathrm{REQ}(N_{j-1})$, $\theta \in \mathcal{A}'$ (see condition 1 of

Definition 3.6);

- for all $j > 1$ and all intervals $[d_i, d_j]$, with $d_j > d_{i+1}$, $L([d_i, d_{j-1}])X_\varphi L([d_i, d_j])$ (see conditions 2 and 3 of Definition 3.6);
- for every interval $[d_i, d_j]$, with $L([d_i, d_j]) = (\mathcal{A}', \mathcal{C}')$, if there exists a formula $A\langle A\rangle\theta \in \mathcal{A}'$, then there exists a point $d_k \geq d_j$ such that $L([d_i, d_k]) = (\mathcal{A}'', \mathcal{C}'')$ and $\theta \in \mathcal{C}''$ (see Definition 3.8).

Definitions 3.6 and 3.8 guarantee that there exists a labelling with such properties.

$\mathcal{C}$ contains at least one existential formula. In such a case, for every existential formula $\psi \in \mathcal{C}$, we guarantee that ψ gets satisfied by properly labelling an infinite branch starting from d_1. Let $\psi \in \mathcal{C}$ be an existential formula. Two cases may arise (depending on the structure of ψ).

- $\psi = E\langle A\rangle\theta$. By condition F1, there exists an $\mathcal{R}$-successor N_2 of N_1 and an atom $(\mathcal{A}', \mathcal{C}') \in N_2$ such that $\psi \in \mathcal{A}'$. Let $d_2 = N_1N_2$. We put $L([d_1, d_2]) = (\mathcal{A}', \mathcal{C}')$. By condition F2, either $\theta \in \mathcal{C}'$ (and thus θ is satisfied over the interval $[d_1, d_2]$) or there exists a path $N_2N_3 \dots N_k$ and a corresponding atom path $(\mathcal{A}', \mathcal{C}') = (\mathcal{A}_2, \mathcal{C}_2)(\mathcal{A}_3, \mathcal{C}_3) \dots (\mathcal{A}_k, \mathcal{C}_k)$, such that $\theta \in \mathcal{C}_k$. In the latter case, let $d_2d_3 \dots d_k$ be the branch in $\mathbb{D}$ corresponding to $N_2N_3 \dots N_k$. We put, for every $3 \leq i \leq k$, $L([d_1, d_i]) = (\mathcal{A}_i, \mathcal{C}_i)$, in order to satisfy θ over $[d_1, d_k]$. In both cases, we extend the finite branch to an infinite one (starting from N_2 or from N_k, respectively) and we define the labelling of all other intervals on the branch as in the case in which $\mathcal{C}$ contains no existential formulas.
- $\psi = E[A]\theta$. By condition F1, there exists an $\mathcal{R}$-successor N_2 of N_1 and an atom $(\mathcal{A}', \mathcal{C}') \in N_2$ such that $\psi \in \mathcal{A}'$. Let $d_2 = N_1N_2$. We put $L([d_1, d_2]) = (\mathcal{A}', \mathcal{C}')$. By condition F3, there exist a fulfilling infinite path $N_2N_3 \dots$ and a corresponding infinite atom path $(\mathcal{A}', \mathcal{C}') = (\mathcal{A}_2, \mathcal{C}_2)(\mathcal{A}_3, \mathcal{C}_3) \dots$, such that $E[A]\theta \in \mathcal{A}_i$ for every $i \geq 2$. Let $d_2d_3 \dots$ be the infinite branch in $\mathbb{D}$ corresponding to $N_2N_3 \dots$. We put, for every $i \geq 3$, $L([d_1, d_i]) = (\mathcal{A}_i, \mathcal{C}_i)$, in order to satisfy θ over $[d_1, d_i]$. As before, we define the labelling of all other intervals on the branch as in the case in which $\mathcal{C}$ contains no existential formulas.

We repeat such a procedure for every existential formula in $\mathcal{C}$.

Inductive step. Let $d \in D$ such that *(i)* L is defined over all intervals $[d', d]$; *(ii)* for all $d' \lhd d$, either d' has been already taken into consideration or L is not defined over any interval $[d'', d']$; *(iii)* d has not been taken into consideration yet. Consider the set $\mathrm{REQ}(N)$, with N such that $d = N_1 \dots N$, and suppose there exists an existential formula $\psi \in \mathrm{REQ}(N)$. Two cases may arise. Either ψ is satisfied by the current labelling L, and we are done, or there are no branches starting from d that satisfy ψ. In the latter case, we satisfy ψ by defining a suitable labelling of an unlabeled infinite branch starting from d, as we have done in the base case of the induction. By repeating such a procedure

for every existential formula $\psi \in \mathrm{REQ}(N)$, we guarantee that all existential formulas in $\mathrm{REQ}(N)$ are satisfied.

The model satisfying φ contains all and only the labelled (infinite) branches of $\mathbb{D}$. Let $\mathbb{D}'$ be the infinite tree obtained from $\mathbb{D}$ by removing all unlabeled branches and let $\mathcal{V}$ be a valuation function $\mathcal{V}$ such that, for every $p \in AP$ and $[d_i, d_j] \in \mathbb{I}(\mathbb{D}')^-$, $p \in \mathcal{V}([d_i, d_j])$ if and only if $L([d_i, d_j]) = (\mathcal{A}_{[d_i,d_j]}, \mathcal{C}_{[d_i,d_j]})$ and $p \in \mathcal{C}_{[d_i,d_j]}$. We prove by induction on the structure of $\psi \in \mathrm{CL}(\varphi)$ that, for every $[d_i, d_j] \in \mathbb{I}(\mathbb{D}')^-$, we have that $\mathbf{M}^-, [d_i, d_j] \Vdash \psi$ if and only if $L([d_i, d_j]) = (\mathcal{A}_{[d_i,d_j]}, \mathcal{C}_{[d_i,d_j]})$ and $\psi \in \mathcal{C}_{[d_i,d_j]}$.

- The base case, as well as the case of the propositional connectives $\neg$ and $\vee$, are straightforward.
- Let ψ be the formula $E\langle A\rangle\chi$. Suppose that $E\langle A\rangle\chi \in \mathcal{C}_{[d_i,d_j]}$. By the definition of L, there exists an infinite path $d_j d_{j+1} \ldots$ and an interval $[d_j, d_k]$ such that $\chi \in \mathcal{C}_{[d_j,d_k]}$. By inductive hypothesis, we have that $\mathbf{M}^-, [d_j, d_k] \Vdash \chi$, and thus $\mathbf{M}^-, [d_i, d_j] \Vdash E\langle A\rangle\chi$.

  As for the opposite implication, assume by contradiction that $\mathbf{M}^-, [d_i, d_j] \Vdash E\langle A\rangle\chi$ and $E\langle A\rangle\chi \notin \mathcal{C}_{[d_i,d_j]}$. By atom definition, this implies that $\neg E\langle A\rangle\chi = A[A]\neg\chi \in \mathcal{C}_{[d_i,d_j]}$. By the definition of L, we have that $A[A]\neg\chi \in \mathcal{A}_{[d_j,d_k]}$ for every $d_k > d_j$, and thus $\neg\chi \in \mathcal{C}_{[d_j,d_k]}$. By inductive hypothesis, this implies that $\mathbf{M}^-, [d_j, d_k] \Vdash \neg\chi$ for every $d_k > d_j$, and thus $\mathbf{M}^-, [d_i, d_j] \Vdash A[A]\neg\chi$, which contradicts the hypothesis that $\mathbf{M}^-, [d_i, d_j] \Vdash E\langle A\rangle\chi$.
- Let ψ be the formula $E[A]\chi$. Suppose that $E[A]\chi \in \mathcal{C}_{[d_i,d_j]}$. By the definition of L, there exists an infinite path $\pi = d_j d_{j+1} \ldots$ such that, for every $d_k \in \pi$, $d_k > d_j$, $E[A]\chi \in \mathcal{A}_{[d_j,d_k]}$. By atom definition, this implies that $\chi \in \mathcal{C}_{[d_j,d_k]}$ and, by inductive hypothesis, we have that $\mathbf{M}^-, [d_j, d_k] \Vdash \chi$, for every $d_k \in \pi$, $d_k > d_j$, and thus $\mathbf{M}^-, [d_i, d_j] \Vdash E[A]\chi$.

  As for the opposite implication, assume by contradiction that $\mathbf{M}^-, [d_i, d_j] \Vdash E[A]\chi$ and $E[A]\chi \notin \mathcal{C}_{[d_i,d_j]}$. By atom definition, this implies that $\neg E[A]\chi = A\langle A\rangle\neg\chi \in \mathcal{C}_{[d_i,d_j]}$. By the definition of L, we have that, for every infinite path $d_j d_{j+1} \ldots$ starting from d_j, there exists $d_k \in \pi$, $d_k > d_j$ such that $\neg\chi \in \mathcal{C}_{[d_j,d_k]}$. By inductive hypothesis, this implies that for every infinite path $\pi = d_j d_{j+1} \ldots$ there exists a point $d_k \in \pi$, $d_k > d_j$, such that $\mathbf{M}^-, [d_j, d_k] \Vdash \neg\chi$, and thus $\mathbf{M}^-, [d_i, d_j] \Vdash A\langle A\rangle\neg\chi$, which contradicts the hypothesis that $\mathbf{M}^-, [d_i, d_j] \Vdash E[A]\chi$.

Since $\langle \mathcal{N}, \mathcal{R}\rangle$ is a substructure, $\varphi \in \mathcal{C}_{[d_0,d_1]}$, and thus $\mathbf{M}^-, [d_0, d_1] \Vdash \varphi$. □

3.3 The decision procedure

In this section, we present a decision procedure for BTNL[R]$^-$, that progressively removes from $\mathcal{T}_\varphi$ nodes that cannot contribute to fulfilling substructures.

Algorithm 1 *Let φ be the formula we want to test for satisfiability. The decision procedure works as follows.*

(i) *Build the (unique) initial tableau* $\mathcal{T}_\varphi = \langle \mathcal{N}_\varphi, \mathcal{R}_\varphi \rangle$.

(ii) *Look for a fulfilling substructure by repeatedly applying the following deletion rules, until no more nodes in the tableau can be deleted:*

- *delete any node which is not* $\mathcal{R}_\varphi$*-reachable from an initial node;*
- *delete any node such that there are no fulfilling paths starting from it;*
- *delete any node which does not satisfy the conditions of Definition 3.10.*

(iii) *Let* $\mathcal{T}^* = \langle \mathcal{N}^*, \mathcal{R}^* \rangle$ *be the final tableau. If* $\mathcal{T}^*$ *is not empty, return* true*, otherwise return* false*.*

The check for the existence of fulfilling paths can be performed as follows. Given a formula $A\langle A\rangle\psi \in \mathrm{CL}(\varphi)$, we execute the following marking procedure. First, for all nodes N, mark all atoms $(\mathcal{A}, \mathcal{C}) \in N$ such that $A\langle A\rangle\psi \in \mathcal{A}$ and $\psi \in \mathcal{C}$. Then, for all nodes N, mark all unmarked atoms $(\mathcal{A}, \mathcal{C}) \in N$ such that there exists an $\mathcal{R}_\varphi$-successor M of N that contains a marked atom $(\mathcal{A}', \mathcal{C}')$ such that $(\mathcal{A}, \mathcal{C}) X_\varphi (\mathcal{A}', \mathcal{C}')$. Repeat this last step until no more atoms can be marked. Then, delete all nodes that either contain an unmarked atom $(\mathcal{A}, \mathcal{C})$ with $A\langle A\rangle\psi \in \mathcal{A}$ or have no $\mathcal{R}_\varphi$-successors.

The other non trivial step of the algorithm is the removal of nodes that do not satisfy the conditions of Definition 3.10. Given a node N, condition F1 can be easily checked by visiting the $\mathcal{R}_\varphi$-successors of N, while condition F2 can be checked by visiting the $\mathcal{R}_\varphi$-descendants of N. Finally, given a node N_0, an atom $(\mathcal{A}_0, \mathcal{C}_0) \in N_0$, and a formula $E[A]\psi \in \mathcal{A}_0$, condition F3 is satisfied if we can find a *finite* path of nodes $N_0 N_1 \dots N_j N_{j+1} \dots N_k$ and a corresponding path of atoms $(\mathcal{A}_0, \mathcal{C}_0)(\mathcal{A}_1, \mathcal{C}_1) \dots (\mathcal{A}_j, \mathcal{C}_j)(\mathcal{A}_{j+1}, \mathcal{C}_{j+1}) \dots (\mathcal{A}_k, \mathcal{C}_k)$ such that *(i)* $N_j = N_k$, *(ii)* $(\mathcal{A}_j, \mathcal{C}_j) = (\mathcal{A}_k, \mathcal{C}_k)$, *(iii)* for all $0 \leq i \leq k$, $E[A]\psi \in \mathcal{A}_i$, *(iv)* for every formula $A\langle A\rangle\theta \in \mathcal{A}_0$, there exists $i \geq 0$ such that $\theta \in \mathcal{C}_i$. Furthermore, to guarantee that the infinite path

$$N_0 N_1 \dots N_j N_{j+1} \dots N_{k-1} N_j N_{j+1} \dots N_{k-1} \dots$$

is a fulfilling one, it suffices to check that, for every atom $(\mathcal{A}', \mathcal{C}') \in N_j$ and every formula $A\langle A\rangle\xi \in \mathcal{C}'$, either $\xi \in \mathcal{C}'$ or there exists a node N_l, with $j < l < k$, and an atom $(\mathcal{A}'', \mathcal{C}'') \in N_l$ such that $\xi \in \mathcal{C}''$ and $(\mathcal{A}'', \mathcal{C}'')$ is a X_φ-descendant of $(\mathcal{A}', \mathcal{C}')$.

As for complexity issues, we have that:

- $|\mathcal{T}_\varphi| = 2^{2^{O(|\varphi|)}}$;
- all checkings of step (ii) of the algorithm can be done in time polynomial in the size of $|\mathcal{T}_\varphi|$;
- after deleting at most $|\mathcal{N}_\varphi|$ nodes, the algorithm terminates.

Hence, checking the satisfiability for a BTNL[R]$^-$ formula has an overall time bound of $2^{2^{O(|\varphi|)}}$, that is, doubly exponential in the size of φ.

4 The decision procedure at work

In this section we apply the proposed decision procedure to the (satisfiable) formula $\varphi = E[A]p$. We show only a portion of the whole tableau, which is sufficiently large to include a fulfilling substructure for φ, and thus to prove that φ is satisfiable.

When searching for a fulfilling substructure for φ, we must take into consideration atoms which have been obtained by suitably combining one set of active requests with one set of current formulas. The sets of active requests and current formulas are the following ones (the left column reports the sets of active requests, while the middle and the right columns report the sets of current formulas):

$$\begin{array}{lll} \emptyset; & \mathcal{C}_0 = \{E[A]p, p\}; & \mathcal{C}_3 = \{A\langle A\rangle\neg p, \neg p\}. \\ \mathcal{A}_0 = \{E[A]p\}; & \mathcal{C}_1 = \{E[A]p, \neg p\}; & \\ \mathcal{A}_1 = \{A\langle A\rangle\neg p\}; & \mathcal{C}_2 = \{A\langle A\rangle\neg p, p\}; & \end{array}$$

As an example, consider the initial node $N_0 = \{(\emptyset, \mathcal{C}_0)\}$. Figure 2 depicts a portion of $\mathcal{T}_\varphi$ that is $\mathcal{R}_\varphi$-reachable from N_0.

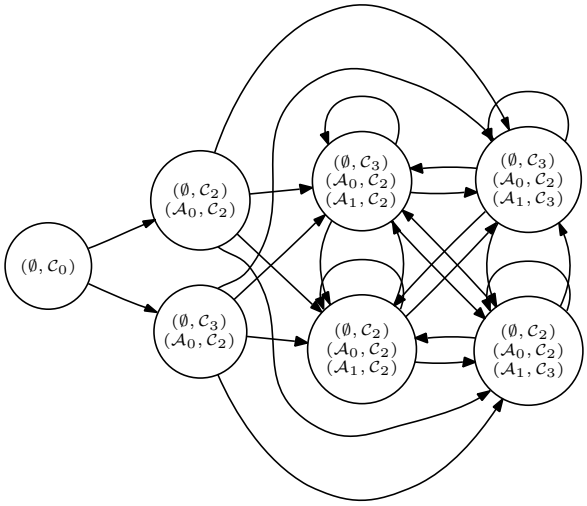


Figure 2. A portion of the tableau for $E[A]p$.

The only atoms with $A\langle A\rangle$-formulas in their set of active requests are $(\mathcal{A}_1, \mathcal{C}_2)$ and $(\mathcal{A}_1, \mathcal{C}_3)$, since $A\langle A\rangle\neg p \in \mathcal{A}_1$. The atom $(\mathcal{A}_1, \mathcal{C}_3)$ immediately fulfills $A\langle A\rangle\neg p$, since $\neg p \in \mathcal{C}_3$. The atom $(\mathcal{A}_1, \mathcal{C}_2)$ does not fulfill $A\langle A\rangle\neg p$, but we have that $(\mathcal{A}_1, \mathcal{C}_2) X_\varphi (\mathcal{A}_1, \mathcal{C}_3)$. Thus, since every node that contains $(\mathcal{A}_1, \mathcal{C}_2)$ can reach a node containing $(\mathcal{A}_1, \mathcal{C}_3)$, for every node of the substructure of Figure 2 there exists a fulfilling path.

As for condition F1, the only sets of current formulas which contains an existential formula are $\mathcal{C}_1$ and $\mathcal{C}_0$. $\mathcal{C}_1$ does not belong to any node in Figure 2, and thus we can ignore it. $\mathcal{C}_0$ only belongs to the initial node, whose two successors include the atom $(\mathcal{A}_0, \mathcal{C}_2)$ with $E[A]p \in \mathcal{A}_0$. Hence, condition F1 is satisfied. Since there are no atoms containing formulas of the form $E\langle A\rangle\psi$,

condition F2 is trivially satisfied. As for condition F3, consider the atom $(\mathcal{A}_0, \mathcal{C}_2)$. It is easy to see that, for every node N containing $(\mathcal{A}_0, \mathcal{C}_2)$, there exists a fulfilling infinite path starting from N such that every node contains $(\mathcal{A}_0, \mathcal{C}_2)$. Thus, condition F3 is satisfied. This allows us to conclude that the substructure depicted in Figure 2 is fulfilling, and thus our decision procedure correctly concludes that the formula $E[A]p$ is satisfiable.

5 Conclusions and further work

In this paper, we proposed a new propositional interval temporal logic, interpreted over infinite trees, which combines the interval neighborhood operators $\langle A \rangle$ and $[A]$ of RPNL$^-$ with the path quantifiers A and E of branching time temporal logics, and we provided it with a doubly-exponential tableau-based decision procedure. We do not know yet if the satisfiability problem for BTNL[R]$^-$ is doubly EXPTIME-complete or not (we conjecture it is not). As for possible extensions of the work, we are studying the decision problem for combinations of path quantifiers operators with other sets of interval logic operators, e.g., those of PNL [6].

Acknowledgements

We would like to thank the anonymous referees for their helpful comments and suggestions. This work has been partially funded by the Italy/South Africa joint project "Temporal logics in computer and information sciences" and by the INTAS project on "Algebraic and deduction methods in non-classical logics and their applications to computer science".

References

[1] D. Bresolin and A. Montanari, *A tableau-based decision procedure for right propositional neighborhood logic*, in: *Proc. of the International Conference TABLEAUX 2005*, vol. 3702 of *LNAI* (2005), 63–77.

[2] Z. Chaochen and M. R. Hansen, *An adequate first order interval logic*, in: W. de Roever, H. Langmark and A. Pnueli, editors, *Compositionality: the Significant Difference*, vol. 1536 of *LNCS* (1998), 584–608.

[3] E. Emerson and J. Halpern, *Decision procedures and expressiveness in the temporal logic of branching time*, Journal of Computer and System Sciences **30** (1985), 1–24.

[4] E. A. Emerson, *Temporal and modal logic*, in: J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, vol. B, MIT Press, 1990 995–1072.

[5] V. Goranko, A. Montanari, and G. Sciavicco, *A general tableau method for propositional interval temporal logic*, in: *Proceedings of the International Conference TABLEAUX 2003*, vol. 2796 of *LNAI* (2003), 102–116.

[6] V. Goranko, A. Montanari, and G. Sciavicco, *Propositional interval neighborhood temporal logics*, Journal of Universal Computer Science **9** (2003), 1137–1167.

[7] V. Goranko, A. Montanari, and G. Sciavicco, *A road map of interval temporal logics and duration calculi*, Journal of Applied Non-Classical Logics **14** (2004), 9–54.

[8] V. Goranko, A. Montanari, G. Sciavicco, and P. Sala, *A general tableau method for propositional interval temporal logics: theory and implementation*, Journal of Applied Logic (2005), to appear.

[9] J. Halpern and Y. Shoham, *A propositional modal logic of time intervals*, Journal of the ACM **38** (1991), 935–962.

[10] A. Montanari, *Propositional interval temporal logics: some promising paths*, in: *Proc. of the 12th International Symposium on Temporal Representation and Reasoning (TIME)* (2005), 201–203.

[11] B. Moszkowski, "Reasoning about digital circuits," Tech. rep. stan-cs-83-970, Dept. of Computer Science, Stanford University, Stanford, CA (1983).

[12] B. Paech, *Gentzen-systems for propositional temporal logics*, in: *Proceedings of CSL '88, 2nd Workshop on Computer Science Logic*, vol. 385 of *LNCS* (1989), 240–253.

[13] Y. Venema, *A modal logic for chopping intervals*, Journal of Logic and Computation **1** (1991), 453–476.

Verifying qualitative and quantitative properties with LTL over concrete domains

Régis Gascon

Laboratoire Spécification et Vérification
CNRS & ENS Cachan
FRANCE

gascon@lsv.ens-cachan.fr

Abstract. We introduce different extensions of LTL where propositional variables are replaced by constraints interpreted in $\mathbb{Z}$. We show different decidability and complexity results for the satisfiability and model checking problems of these logics. The extension of LTL over a wide set of *qualitative* constraints is shown to be PSPACE-complete. When introducing some *quantitative* constraints, we must consider strong restrictions to regain decidability.

1 Introduction

Model checking counter automata. The verification of infinite-state systems has benefited from the numerous decidable model-checking problems. But though decidability can be obtained via numerous proof techniques, showing undecidability of model-checking is often easier since the halting problem for Minsky machines [Min67] is already undecidable. Decidability is difficult to establish and sometimes it can be regained by naturally restricting the class of models or by considering fragments of the specification language. Symbolic representations of infinite sets of states are often the key argument to get decidability (see for instance [HMR05]).

Structures with a finite set of control states augmented with a finite set of variables interpreted as integers (counter automata) are ubiquitous in computer science. They are used as operational models of numerous infinite-state systems, including broadcast protocols [FL02], and even their restrictions to one counter have found applications to verify cryptographic protocols [LLT05] or to validate XML streams [CR04]. However, the class of counter machines has numerous undecidable model-checking problems such as the reachability problem. This does not end the story since many subclasses admit a decidable reachability problem as reversal-bounded multicounter machines [Iba78] or flat counter systems [Boi98,FL02].

Motivations. Classical problems studied on counter automata include reachability issues and verification of properties on the control states. We aim at checking richer properties by introducing constraints on counters in temporal

logic. Defining decidable model-checking problems with such logics is difficult since introducing some *quantitative* constraints (i.e. counting mechanism) like $y = x + 1$ allows to encode a two counter machine. A first alternative is to consider only *qualitative* constraints where the relation between the constrained terms is not sharp, like $y \leq x$ or $y \equiv_k x + 1$. Another option is to consider decidable fragments of undecidable logics by restricting some resources of the formulae. For instance, bounding the number of variables could be a good restriction since we need to store the value of three different counters to encode a two-counter machine (the two counters and the instruction counter).

Contributions. Extensions of the temporal logic LTL over concrete domains have already been considered in [WZ00,BC02,DD02]. We introduce new extensions of LTL over concrete domains where variables are interpreted in $\mathbb{Z}$. The propositional variables are replaced by constraints (induced by the concrete domain) between terms representing the variables at different states of the models. For instance, the constraint $x < \mathsf{XX}x$ means that the current value of x is smaller than the value of x two states further. Such constraints can also be found in description logics over concrete domains [Lut04]. We consider here linear models but the constraint language is very rich.

First, we show the PSPACE-completeness of an extension of LTL over a wide set of qualitative constraints, including periodicity constraints of the form $y \equiv_k x+1$ and comparisons of the form $y < x$. Such constraints are used for instance in calendar formalisms [LM01] when one needs to consider the periodicity of time or in abstraction of counter automata where operations are defined modulo some power of two [LS01]. Indeed, common programming languages perform arithmetical operations modulo 2^k [MOS05] (k is equal to 16 or 32).

Then we consider another extension introducing quantitative constraints. This full formalism can easily be shown to be undecidable and so we study restrictions of the logic bounding syntactic resources of the formulae. The resources we consider are the number of variables and the distance between the positions (in the model) of the terms that are compared. For example, the distance between $\mathsf{X}x$ and $\mathsf{XXX}x$ is 2. Even with these restrictions the decidability is very difficult to regain. We show that the fragment with one variable and distance two and the fragment with two variables and distance one are undecidable. However, we can prove that the model checking of the fragment with one variable and distance one over one counter automata is PSPACE-complete.

2 LTL over concrete domains

Let $V = \{x_0, x_1, \dots\}$ be a countable infinite set of variables. A *concrete domain* is a pair $\mathcal{D} = \langle D, (R_\alpha)_{\alpha \in I} \rangle$ where D is a specific domain of interpretation for the variables and $(R_\alpha)_{\alpha \in I}$ is a countable family of relations on the elements of D. An *atomic $\mathcal{D}$-constraint* is an expression of the form $\mathsf{R}(x_1, \dots, x_n)$ where R is interpreted as a relation of the domain and n is the arity of this relation. A *D-valuation* is a function $v : V \to D$ that assigns to every variable a value in

D and a constraint is satisfied by v, denoted by $v \models \mathsf{R}(x_1, \ldots, x_n)$, whenever $(v(x_1), \ldots, v(x_n)) \in R$, R being the relation in $\mathcal{D}$ associated to the symbol R.

We define the logic CLTL($\mathcal{D}$) as the extension of LTL where the propositional variables are replaced by atomic $\mathcal{D}$-constraints over *terms* representing different states of the variables, denoted by $\mathsf{R}(\mathsf{X}^{i_1}x_1, \ldots, \mathsf{X}^{i_n}x_n)$. The formulae of CLTL($\mathcal{D}$) are defined by the following grammar:

$$\phi ::= \mathsf{R}(\mathsf{X}^{i_1}x_1, \ldots, \mathsf{X}^{i_k}x_k) \mid \phi \wedge \phi \mid \neg\phi \mid \mathsf{X}\phi \mid \phi\mathsf{U}\phi.$$

The symbols X and U are the classical operators next and until of LTL. A term $\mathsf{X}^i x$ (abbreviation for $\mathsf{X}\ldots\mathsf{X}x$) represents the value of x at the ith next state. A *one-step constraint* is an atomic formula of the form $\mathsf{R}(\mathsf{X}^{i_1}x_1, \ldots, \mathsf{X}^{i_k}x_k)$ such that $i_1, \ldots, i_k \leq 1$. Given a CLTL($\mathcal{D}$) formula ϕ, we define its X-*length* $|\phi|_{\mathsf{X}}$ as the maximal number i such that a term of the form $\mathsf{X}^i x$ occurs in ϕ. Intuitively, the X-length defines a frame of consecutive states that can be compared. The models of CLTL($\mathcal{D}$) are sequences of D-valuations $\sigma : \mathbb{N} \to (V \to D)$ and the satisfaction relation is defined like the LTL satisfaction relation except at the atomic level:

- $\sigma, i \models \mathsf{R}(\mathsf{X}^{j_1}x_1, \ldots, \mathsf{X}^{j_n}x_n)$ iff $(\sigma(i+j_1)(x_1), \ldots, \sigma(i+j_n)(x_n)) \in R$,
- $\sigma, i \models \phi \wedge \phi'$ iff $\sigma, i \models \phi$ and $\sigma, i \models \phi'$,
- $\sigma, i \models \neg\phi$ iff $\sigma, i \not\models \phi$,
- $\sigma, i \models \mathsf{X}\phi$ iff $\sigma, i+1 \models \phi$,
- $\sigma, i \models \phi\mathsf{U}\phi'$ iff there is $j \geq i$ s.t. $\sigma, j \models \phi'$ and for every $i \leq k < j$, $\sigma, k \models \phi$.

As usual, a formula $\phi \in$ CLTL($\mathcal{D}$) is satisfiable whenever there exists a model σ such that $\sigma, 0 \models \phi$. The *satisfiability problem* takes as input a formula and checks whether it is satisfiable.

The *model checking problem* is defined for the class of $\mathcal{D}$-*automata* that are Büchi automata with transitions labeled by one-step constraints. Formally, a k-variable $\mathcal{D}$-automaton $\mathcal{A}$ is a structure $\langle Q, \delta, I, F\rangle$ such that Q is a finite set of states, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states and δ is a subset of $Q \times 1SC_k \times Q$ where $1SC_k$ is the set of Boolean combinations of one-step constraints built over the set of variables $\{x_1, \ldots, x_k\}$. $\mathcal{D}$-automata have no input alphabet: we do not use them as language acceptors but we are rather interested in the behaviors of such systems. A *configuration* of $\mathcal{A}$ is a tuple $\langle q, \boldsymbol{c}\rangle \in Q \times D^k$. Let $\boldsymbol{c}_{[i]}$ denote the ith value of $\boldsymbol{c}$. The one-step transition relation $\rightarrow \subseteq (Q \times D^k)^2$ is defined by: $\langle q, \boldsymbol{c}\rangle \rightarrow \langle q', \boldsymbol{c'}\rangle \stackrel{\text{def}}{\Leftrightarrow}$ there is a transition $\langle q, \varphi, q'\rangle \in \delta$ such that $[x_1 \leftarrow \boldsymbol{c}_{[1]}, \ldots, x_k \leftarrow \boldsymbol{c}_{[k]}, \mathsf{X}x_1 \leftarrow \boldsymbol{c'}_{[1]}, \ldots, \mathsf{X}x_k \leftarrow \boldsymbol{c'}_{[k]}] \models \varphi$.

An infinite path w is a map $\mathbb{N} \to (Q \times D^k)$ such that for every $i \in \mathbb{N}$, we have $w(i) \rightarrow w(i+1)$. An accepting run for $\mathcal{A}$ is an infinite path w such that $w(0)$ is of the form $\langle q, \boldsymbol{c}\rangle$ with $q \in I$ and the set of configurations in w of the form $\langle q_f, \boldsymbol{c}\rangle$ with $q_f \in F$ is infinite.

Given an infinite path w, let σ_w be the CLTL($\mathcal{D}$) model such that for all $i \in \mathbb{N}$ and $j \in \{1, \ldots, k\}$, $\sigma_w(i)(x_j) = \boldsymbol{c}_{[j]}$ where $w(i) = \langle q, \boldsymbol{c}\rangle$. The model-checking problem for CLTL($\mathcal{D}$) takes as input a CLTL($\mathcal{D}$) formula ϕ and a $\mathcal{D}$-automaton $\mathcal{A}$ and checks whether there is an accepting run w of $\mathcal{A}$ such that

$\sigma_w, 0 \models \phi$. It is easy to show that the satisfiability problem can be reduced to the model-checking problem.

3 Verifying qualitative constraints

We introduce in this section an extension of LTL over a concrete domain with qualitative constraints only. By qualitative, we mean that the relation between the constrained terms is non deterministic. We distinguish two types of constraints: comparisons and periodicity constraints that are congruence modulo relations. We consider the language of constraints IPC*, which is an extension of the language IPC of [Dem04], defined as following:

$$\begin{array}{c} \varphi ::= \varphi_{mod} \mid x < y \mid \varphi \wedge \varphi \mid \neg\varphi \\ \varphi_{mod} ::= \; x \equiv_k [c_1, c_2] \mid x \equiv_k y + [c_1, c_2] \mid x = y \mid x < d \mid x = d \mid \\ \varphi_{mod} \wedge \varphi_{mod} \mid \neg\, \varphi_{mod} \mid \exists x \; \varphi_{mod} \end{array}$$

where $x, y \in V$, $k \in \mathbb{N} \setminus \{0\}$, $c_1, c_2 \in \mathbb{N}$ and $d \in \mathbb{Z}$. The models of IPC* are $\mathbb{Z}$-valuations of the form $v : V \to \mathbb{Z}$ and the satisfaction relation is defined by:

- $v \models x \sim y \stackrel{\text{def}}{\Leftrightarrow} v(x) \sim v(y)$ and $v \models x \sim d \stackrel{\text{def}}{\Leftrightarrow} v(x) \sim d$ where $\sim \in \{<, =\}$,
- $v \models x \equiv_k [c_1, c_2] \stackrel{\text{def}}{\Leftrightarrow} v(x)$ equals c modulo k for some $c_1 \leq c \leq c_2$,
- $v \models x \equiv_k y + [c_1, c_2] \stackrel{\text{def}}{\Leftrightarrow} v(x) - v(y)$ equals c modulo k for some $c_1 \leq c \leq c_2$,
- $v \models p \wedge p' \stackrel{\text{def}}{\Leftrightarrow} v \models p$ and $v \models p'$,
- $v \models \neg p \stackrel{\text{def}}{\Leftrightarrow}$ not $v \models p$,
- $v \models \exists x \; p \stackrel{\text{def}}{\Leftrightarrow}$ there exists $z \in \mathbb{Z}$ such that $v[x \leftarrow z] \models p$ where $v[x \leftarrow z]$ is defined by $v[x \leftarrow z](x') = v(x')$ for every $x \neq x'$ and $v[x \leftarrow z](x) = z$.

In the following, a set of constraints is satisfiable if there is a valuation satisfying all its elements.

We write CLTL(IPC*) to denote the extension of LTL over the concrete domain induced by the constraint language IPC*. For instance, $\mathsf{X}x \equiv_5 \mathsf{X}y + [0, 2]$ or $\mathsf{X}x < x$ are one-step constraints of CLTL(IPC*) and $(\mathsf{X}x < x)\mathsf{U}(x = 0)$ is a CLTL(IPC*) formula. The satisfiability and model-checking problems for CLTL(IPC*) are reducible to each other in logarithmic space following techniques from [SC85]. So the results below about the satisfiability problem also extend to the model-checking problem. Note that extending IPC* by allowing constraints of the form $x < y$ in the scope of $\exists$ leads to undecidability since the successor relation for integers is then definable and the halting problem for Minsky machines can be easily encoded by a formula of the corresponding extension of LTL.

Following the approach in [VW94], we have shown in [DG05] that given a CLTL(IPC*) formula ϕ, we can build a standard Büchi automaton $\mathcal{A}_\phi$ such that ϕ is satisfiable iff $\mathrm{L}(\mathcal{A}_\phi)$ is non-empty. Moreover, checking emptiness of $\mathrm{L}(\mathcal{A}_\phi)$ can be done in polynomial space in $|\phi|$.

Symbolic models. The language recognized by the Büchi automaton $\mathcal{A}_\phi$ is not a set of models but a set of symbolic models. We briefly explain below how we define a symbolic representation of the models (see details in [DG05]). Given a formula ϕ, a symbolic valuation is a set of constraints depending of the least common multiple of the integers k_i such that a relation $\equiv_{k_i}$ occurs in ϕ, the set of constants d occurring in constraints of the form $x \sim d$ and the finite set of variables occurring in ϕ. This symbolic representation of a $\mathbb{Z}$-valuation contains all the relevant information to evaluate constraints. Symbolic valuations define an equivalence relation between $\mathbb{Z}$-valuations, like regions for timed automata.

Let ρ be an infinite symbolic valuation sequence. We say that ρ is satisfiable iff there is a model $\sigma : \mathbb{N} \to (V \to \mathbb{Z})$ such that for every $i \in \mathbb{N}$ we have $\sigma, i \models \rho(i)$ (and we note $\sigma \models \rho$). A symbolic model for the formula ϕ is a *one-step consistent* sequence of symbolic valuation wrt ϕ. A sequence is one-step consistent if the constraints are propagated along the sequence: for instance if $\mathsf{X}^a x \sim \mathsf{X}^b y + d$ is a constraint of $\rho(i)$ and both $a > 0$ and $b > 0$, it must be the case that the constraint $\mathsf{X}^{a-1} x \sim \mathsf{X}^{b-1} y + d$ occurs at the next position of the sequence (i.e. $\rho(i+1)$). We need this property to eliminate some irrelevant sequences since non one-step consistent sequences are not satisfiable. We define a symbolic satisfaction relation between symbolic models and formulae denoted $\rho \models_{\text{symb}} \phi$. In order to prove the decidability we use the fact below:

Lemma. *A* CLTL(IPC*) *formula ϕ is satisfiable iff there are a symbolic model ρ and a model σ such that $\sigma \models \rho$ and $\rho \models_{\text{symb}} \phi$.*

Approximation. The main difficulty in the construction of $\mathcal{A}_\phi$ is to characterize by means of automata the set of satisfiable symbolic models (one-step consistence is not enough), which is not ω-regular. We introduce an ω-regular condition $(\mathcal{C})$ (see definition in [DG05]) that over-approximate this set: every satisfiable symbolic model verifies $(\mathcal{C})$ but some symbolic models verifying $(\mathcal{C})$ are not satisfiable. However this condition is equivalent to satisfiability whenever the symbolic model is *ultimately periodic*. A symbolic model is ultimately periodic if it is of the form $\gamma \cdot (\delta)^\omega$ (the suffix δ is repeated infinitely often).

Construction of the automaton. The automaton $\mathcal{A}_\phi$ is defined as the intersection $\mathcal{A}_{\text{LTL}} \cap \mathcal{A}_{\mathcal{C}}$ of Büchi automata where $\text{L}(\mathcal{A}_{\text{LTL}})$ is the set of symbolic models satisfying ϕ and $\text{L}(\mathcal{A}_{\mathcal{C}})$ is the set of symbolic models verifying $(\mathcal{C})$. We can prove that a CLTL(IPC*) formula ϕ is satisfiable iff $\text{L}(\mathcal{A}_\phi)$ is non-empty. Like in [DD02], the main trick of the proof is that if $\text{L}(\mathcal{A}_\phi)$ is non-empty then there exists an ultimately periodic ω-sequence $\rho \in \text{L}(\mathcal{A}_\phi)$. Thus ρ is satisfiable since it is ultimately periodic and verifies $(\mathcal{C})$.

Given a formula ϕ we can effectively build $\mathcal{A}_\phi$ and check whether $\text{L}(\mathcal{A}_\phi)$ is empty. So the satisfiability problem for CLTL(IPC*) is decidable. We can also establish the PSPACE upper bound by using the fact that all the components of the automaton can be built in PSPACE and subtle arguments from complexity theory and [Saf89].

Theorem. [DG05] *The satisfiability problem and the model checking problem for the logic* CLTL(IPC*) *are* PSPACE*-complete.*

Since the operators of CLTL(IPC*) are definable in the monadic second order logic (MSO), we can also prove using [GK03] that any extension of CLTL(IPC*) obtained by adding MSO-definable operators remains in PSPACE. We only need to update the automaton $\mathcal{A}_{\text{LTL}}$.

4 Verifying quantitative constraints

In this section, we introduce another instance of LTL over a concrete domain where the variables are interpreted in $\mathbb{Z}$ and the underlying constraint language is a fragment of Presburger arithmetic. The main difference with IPC* is the introduction of quantitative constraints of the form $x = y + d$ where $d \in \mathbb{Z}$. We show that such constraints easily lead to undecidability, even when restricting strongly the syntactic resources of the formulae. More details will be available in [DG]. The constraint language $\mathcal{P}_r$ is defined by:

$$\varphi ::= x \sim y + d \mid x \sim d$$

where $x, y \in V$, $\sim \in \{<, =\}$ and $d \in \mathbb{Z}$. The variables of $\mathcal{P}_r$ are interpreted in $\mathbb{Z}$ and given a valuation $v : V \to \mathbb{Z}$, the satisfaction relation $v \models \varphi$ is defined in the obvious way. Like in the previous section, the extension of LTL over the concrete domain induced by $\mathcal{P}_r$ is denoted by CLTL($\mathcal{P}_r$).

Counter Automata. The class of $\mathcal{P}_r$-automata contains many known classes of counter automata and can easily simulate two-counter non-deterministic Minsky machines. A k-counter automaton is a $\mathcal{P}_r$-automaton such that the constraints on transitions are conjunctions of constraints expressing zero tests, sign tests and updates of the counters of the form $\mathsf{X}x = x + d$ where $d \in \{-1, 0, 1\}$. The existence of an accepting run for one-counter automata is a more difficult question than similar questions studied for instance in [LLT05] since we deal with a Büchi acceptance condition, the counter is interpreted in $\mathbb{Z}$ and zero test and sign test are allowed.

Decidability status. Full CLTL($\mathcal{P}_r$) can easily be shown to be undecidable. We consider restrictions of the syntactic resources of the formulae while preserving the full strength of the logical operators. We denote with $\text{CLTL}_k^l(\mathcal{P}_r)$ the fragment of CLTL($\mathcal{P}_r$) where the number of variables of the formulae is bounded by k and the X-length by l. Known results show that the fragments $\text{CLTL}_2^\omega(\mathcal{P}_r)$ and $\text{CLTL}_3^1(\mathcal{P}_r)$ have undecidable satisfiability problems (see [CC00] and [DD02]). We refine these undecidability results: the decidability status of the different fragments are shown in the following table (our results are in bold characters):

$k \setminus l$	1	2	ω
1	**pspace-complete**	**undecidable**	**undecidable**
2	**undecidable**	undecidable	undecidable [DD02]
3	undecidable [CC00]	undecidable	undecidable

We show that bounding the number of variables is not enough to regain decidability by proving the undecidability of $\mathrm{CLTL}_1^{\omega}(\mathcal{P}_r)$. We also establish sharp decidability limits of these fragments by showing the undecidability of $\mathrm{CLTL}_1^2(\mathcal{P}_r)$ and $\mathrm{CLTL}_2^1(\mathcal{P}_r)$. These undecidability results can be proved by encoding two-counter automata and reducing different fragments of the logic.

To complete the picture, we show that the fragment $\mathrm{CLTL}_1^1(\mathcal{P}_r)$ is PSPACE-complete. As in the previous section, the proof relies on the construction of a particular one-counter automaton recognizing symbolic representations of the models. This complexity result uses an auxiliary result about reachability in one-counter automata. We prove that the existence of an accepting run in such automata can be decided in NLOGSPACE.

To conclude, we mention that the extension $\mathcal{P}_r'$ of $\mathcal{P}_r$ with constraints of the form $ax + by = 0$ where $a, b \in \mathbb{Z}$ leads to undecidability even for the fragment $\mathrm{CLTL}_1^1(\mathcal{P}_r')$. Indeed, the values of two counters $\langle c_1, c_2 \rangle$ in the configuration of a Minsky machine can be encoded by the value $2^{c_1}3^{c_2}$ for the variable. Zero tests, increments and decrements are encoded by constraints of the form $x \equiv_2 0$, $x \equiv_3 0$, $\mathsf{X}x = 2x$ (incrementation of the first counter) etc. So, as far as decidability is concerned, the underlying fragment of Presburger arithmetic in $\mathrm{CLTL}(\mathcal{P}_r)$ is quite optimal with respect to the restrictions we consider.

References

[BC02] Ph. Balbiani and J.F. Condotta. Computational complexity of propositional linear temporal logics based on qualitative spatial or temporal reasoning. In *FroCoS'02*, volume 2309 of *LNAI*, pages 162–173. Springer, 2002.

[Boi98] B. Boigelot. *Symbolic methods for exploring infinite state spaces*. PhD thesis, Université de Liège, 1998.

[CC00] H. Comon and V. Cortier. Flatness is not a weakness. In *CSL'00*, volume 1862 of *LNCS*, pages 262–276. Springer, 2000.

[CR04] C. Chitic and D. Rosu. On validation of XML streams using finite state machines. Technical Report CSRG-489, University of Toronto, 2004.

[DD02] S. Demri and D. D'Souza. An automata-theoretic approach to constraint LTL. In *FST&TCS'02*, volume 2256 of *LNCS*, pages 121–132. Springer, 2002.

[Dem04] S. Demri. LTL over integer periodicity constraints. In *FOSSACS'04*, volume 2987 of *LNCS*, pages 121–135. Springer, 2004.

[DG] S. Demri and R. Gascon. The Effects of Bounding Syntactic Resources on Presburger LTL. Submitted.

[DG05] S. Demri and R. Gascon. Verification of qualitative $\mathbb{Z}$-constraints. In *CONCUR'05*, volume 3653 of *LNCS*, pages 518–532. Springer, 2005.

[FL02] A. Finkel and J. Leroux. How to compose Presburger accelerations: Applications to broadcast protocols. In *FST&TCS'02*, volume 2256 of *LNCS*, pages 145–156. Springer, 2002.

[GK03] P. Gastin and D. Kuske. Satisfiability and model checking for MSO-definable temporal logics are in PSPACE. In *CONCUR'03*, volume 2761 of *LNCS*, pages 222–236. Springer, 2003.

[HMR05] Th. Henzinger, R. Majumdar, and J.F. Raskin. A classification of symbolic transitions systems. *ACM Transactions on Computational Logic*, 6(1):1–32, 2005.

[Iba78] O. Ibarra. Reversal-bounded multicounter machines and their decision problems. *JACM*, 25(1):116–133, 1978.

[LLT05] P. Lafourcade, D. Lugiez, and R. Treinen. Intruder deduction for AC-like equational theories with homomorphisms. In *RTA'05*, volume 3467 of *LNCS*, pages 308–322. Springer, 2005.

[LM01] U. Dal Lago and A. Montanari. Calendars, time granularities, and automata. In *Int. Symposium on Spatial and Temporal Databases*, volume 2121 of *LNCS*, pages 279–298. Springer, 2001.

[LS01] G. Logothetis and K. Schneider. Abstraction from counters: an application on real-time systems. In *TIME'01*, pages 214–223. IEEE, 2001.

[Lut04] C. Lutz. NEXPTIME-complete description logics with concrete domains. *ACM Transactions on Computational Logic*, 5(4):669–705, 2004.

[Min67] M. Minsky. *Computation, Finite and Infinite Machines*. Prentice Hall, 1967.

[MOS05] M. Müller-Olm and H. Seidl. Analysis of modular arithmetic. In *ESOP'05*, volume 3444 of *LNCS*, pages 46–60. Springer, 2005.

[Saf89] S. Safra. *Complexity of Automata on Infinite Objects*. PhD thesis, The Weizmann Institute of Science, 1989.

[SC85] A. Sistla and E. Clarke. The complexity of propositional linear temporal logic. *JACM*, 32(3):733–749, 1985.

[VW94] M. Vardi and P. Wolper. Reasoning about infinite computations. *I & C*, 115:1–37, 1994.

[WZ00] F. Wolter and M. Zakharyaschev. Spatio-temporal representation and reasoning based on RCC-8. In *KR'00*, pages 3–14, 2000.

Complexity of Hybrid Logics over Transitive Frames

Martin Mundhenk[1], Thomas Schneider[1], Thomas Schwentick[2], and Volker Weber[2]

[1] Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany
{mundhenk, schneider}@cs.uni-jena.de

[2] Fachbereich Informatik, Universität Dortmund, Germany
{thomas.schwentick, volker.weber}@udo.edu

Abstract. This paper examines the complexity of hybrid logics over transitive frames and transitive trees. We show that satisfiability over transitive frames for the hybrid language extended with $\downarrow$ is NEXP-complete. This is in contrast to undecidability of satisfiability over arbitrary frames for this language [2]. We also show that adding @ or the past modality leads to undecidability over transitive frames, but not over transitive trees, where we show the richest language to be nonelementarily decidable. Moreover, we establish 2EXP and EXP upper bounds for satisfiability over transitive frames and transitive trees, respectively, for the hybrid Until/Since language. An EXP lower bound is shown to hold for the modal Until language over both frame classes.

1 Introduction

We examine the computational complexity of satisfiability for hybrid logics over transitive frames. This is an important frame class, because transitivity is the minimal requirement in many temporal applications, for example temporal verification. Modal, hybrid, and first-order logic over transitive models have been studied recently in [3, 14, 30, 18, 19, 17, 11]. Although the complexity of hybrid (tense) logics has been extensively examined [7, 15, 2, 3, 13], there are highly expressive hybrid languages for whose satisfiability problems only results over arbitrary, but not over restricted, temporally relevant, frame classes have been known.

Hybrid Languages are extensions of the language of modal logic that allow for naming and accessing states of a model explicitly. This renders hybrid logic an adequate representation formalism for many applications where the basic modal and/or temporal language does not suffice. Moreover, reasoning systems are easier to devise for hybrid than for modal logic.

Hybrid logic, as well as the foundations of temporal logic, goes back to Arthur Prior [25]. Since then, many — more or less powerful — languages have been studied. Here we briefly introduce the extensions that shall concern us in this paper.

Nominals are special atomic formulae that name states of models. They allow, for instance, for an axiom expressing irreflexivity, which cannot be captured by modal formulae: $i \to \neg\Diamond i$.

The at operator @ can be used to directly jump to states named by nominals, independently of the accessibility relation. Hence, the above formula could also be written as $@_i\neg\Diamond i$.

With the help of *the downarrow operator* $\downarrow$, it is possible to bind variables to states. Whenever $\downarrow x$ is encountered during the evaluation of a formula, the variable x is bound to the actual state s. All occurrences of x in the scope of this $\downarrow$ are treated like nominals naming s. As an example, the formula $\downarrow x.\neg\Diamond\Diamond x$ reads as: Name the actual state x and make sure that it is not possible to reach x in two steps. This is an axiom for asymmetry, another property not expressible in modal logic.

Combined with the @ operator, $\downarrow$ leads to a very powerful language that can formulate many desirable properties and goes far beyond the scope of the simple nominal language. To give a more impressive example, we consider the Until operator. The formula $\mathsf{U}(\varphi, \psi)$ reads as "there is a point in the future at which φ holds, and at all points between now and this point, ψ holds". What the basic modal language is not able to express, can be achieved by the hybrid $\downarrow$-@ language.

$$\mathsf{U}(\varphi, \psi) \equiv \downarrow x.\Diamond\downarrow y.\varphi \wedge @_x\Box(\Diamond y \to \psi).$$

Besides more advanced temporal concepts such as "until" and "since", hybrid temporal languages can express other desirable temporal notions such as "now", "yesterday", "today", or "tomorrow". Moreover, with hybrid logic one can capture many temporally relevant frame properties (besides the above mentioned, antisymmetry, trichotomy, directedness, ...). For this reason, hybrid temporal languages are of great interest where basic temporal logic reaches its limits [9, 5, 4, 13].

Why Transitivity? Hybrid logic is interpreted over Kripke frames and models, as is modal logic. A frame consists of a set of states (points in time) and an accessibility relation (where xRy says that y is in the future of x). In most temporal applications one requires this relation to be transitive. We concentrate on transitive frames because transitivity is a property that the future relations of many different temporal applications have in common, even if they differ in other properties such as tree-likeness, trichotomy, irreflexivity, or asymmetry.

But there are other reasons why this frame class is of interest, particularly in connection with computational complexity. In the special case of linear frames, nominals and @ can be simulated using the conventional modal operator and its converse. They do not add expressive power to the language in this case. The $\downarrow$ operator is useless even on transitive trees, a representation of branching time. Though the class of transitive frames (and transitive trees for $\downarrow$-free hybrid languages, respectively) is a restricted frame class, it is general enough to separate hybrid from modal languages in terms of expressive power.

Yet another reason for considering precisely transitive frames will become clear in the next paragraph.

Complexity of Hybrid Logics. We use the complexity classes NP, PSPACE, EXP, NEXP, nEXP, $n \geq 2$, and coRE as known from [24]. A problem is nonelementarily decidable if it is decidable and not contained in any nEXP.

It goes without saying that reasoning tasks for richer logics require more resources than those for simpler languages such as the basic modal language. We focus on one reasoning task, namely satisfiability. The modal and temporal satisfiability problems over arbitrary as well as transitive frames are PSPACE-complete [22, 29]. If the "somewhere" modality E is added, satisfiability becomes EXP-complete over arbitrary frames [28]. For many, more restricted, frame classes, modal and temporal satisfiability is NP-complete [22, 23, 27]. In contrast, the known part of the complexity spectrum of hybrid satisfiability reaches up to undecidability.

Many complexity results for hybrid languages have been established in [2, 3]. It was shown in [2] that the hybrid language with nominals and @ has a PSPACE-complete satisfiability problem and that satisfiability for the hybrid tense language is EXP-complete, even if @ or E are added. It was proven in [3] that these problems have the same complexity (or drop to PSPACE-complete or NP-complete, respectively) if the class of frames is restricted to transitive frames (or transitive trees, or linear frames, respectively).

Moreover, the authors of [3] established EXP-completeness of satisfiability for the hybrid $\mathsf{Until}/\mathsf{Since}$ language. The complexity of this language over transitive frames and transitive trees, respectively, has been open. PSPACE-completeness over linear frames is known from [13]. We want to find out at which exact requirements to the frame classes the decrease from EXP to PSPACE takes place.

Undecidability results for languages containing $\downarrow$ originate from [7, 15]. The strongest such result, namely for a restricted fragment of the $\downarrow$ language, is given in [2]. In recent work [31], it was shown that decidability of the $\downarrow$ language can be regained under certain restrictions on the frame classes. It is possible that transitivity is another property under which the $\downarrow$ language can be "tamed".

Moreover, we have already observed that over transitive trees and linear orders, the $\downarrow$ operator on its own is useless. Hence satisfiability over these frame classes is the same as for modal logic, namely complete for PSPACE and NP, respectively. This makes it more likely that we can "tame" $\downarrow$ over transitive frames. But if so, to what extent? What happens if we allow for interactions of $\downarrow$ with @ or the backward modality?

New Road-Map Pages. This paper establishes two groups of complexity results for hybrid languages over transitive frames and transitive trees.

First we examine satisfiability of the hybrid $\downarrow$ language. Our most surprising result is the "taming" of this language over transitive frames: the satisfiability problem is NEXP-complete. This high level of complexity is retained even over complete frames (clusters). We also show that enriching the language by the backward-looking modality P or the @ operator leads to undecidability in the case of transitive frames. Over transitive trees, the situation is different. Decidability for even the richest $\downarrow$ language is easy to see, but it is nonelementary, as we will show.

As a second step, we consider satisfiability over transitive frames and transitive trees for the hybrid Until/Since-E language. We establish EXP-hardness for not more than the modal language with Until only. This is matched by an EXP upper bound for the full language in the case of transitive trees. As for transitive frames, we give a 2EXP upper bound.

Table 1 gives an overview of the satisfiability problems considered in this paper (marked bold) and visualizes how our complexity results arrange into a collection of previously known results. It makes use of the denotation of hybrid languages introduced in Section 2. Complexity classes without addition stand for completeness results; "nonel." stands for "nonelementarily decidable". The work from which the results originate, is cited. Conclusions from surrounding results are abbreviated by "c.". The question mark stands for an open question, but decidability follows from the last result in that column. As for $\mathcal{HL}^{\mathsf{E}}_{\mathsf{U,S}}$ over transitive frames and transitive trees, EXP-hardness even holds for $\mathcal{ML}_{\mathsf{U}}$.

hybrid lang.	complexity over arbitrary frames	complexity over transitive frames	complexity over transitive trees	complexity over linear orders
$\mathcal{HL}^{@}$	PSPACE [2]	PSPACE [3]	PSPACE [3, c.]	NP [3, c.]
$\mathcal{HL}_{\mathsf{F,P}}$	EXP [2]	EXP [3]	PSPACE [3]	NP [3]
$\mathcal{HL}^{\mathsf{E}}_{\mathsf{F,P}}$	EXP [3]	EXP [3]	PSPACE [3]	NP [3]
$\mathcal{HL}^{\mathsf{E}}_{\mathsf{U,S}}$	EXP [3]	**in 2EXP** (Th. 14), **EXP-hard** (Th. 12)	**EXP** (Th. 12,15)	PSPACE-hard [26]
$\mathcal{HL}^{\downarrow}$	coRE [2]	**NEXP** (Th. 1)	PSPACE [3]	NP [13]
$\mathcal{HL}^{\downarrow,@}$	coRE [2]	**coRE** (Th. 10)	**nonel.** (Th. 11)	?
$\mathcal{HL}^{\downarrow}_{\mathsf{F,P}}$	coRE [2, c.]	**coRE** (Th. 10)	**nonel.** (Th. 11)	nonel. [13, c.]
$\mathcal{HL}^{\downarrow,@}_{\mathsf{F,P}}$	coRE [2, c.]	**coRE** (c.)	**nonel.** (Th. 11)	nonel. [13]

Table 1. An overview of complexity results for hybrid logics.

Legend. This paper is organized as follows. In Section 2, we give all necessary definitions and notations of modal and hybrid logic as well as an overview of our results in the context of previous work. We present the decidability and undecidability results for the hybrid $\downarrow$ languages in Sections 3 and 4. The hybrid Until/Since language is examined in section 5, and Section 6 ends the paper with some concluding remarks.

2 Modal and Hybrid Logic

We define the basic concepts and notations of modal and hybrid logic that are relevant for this paper. The fundamentals of modal logic can be found in [6]; those of hybrid logic in [2, 5].

Modal Logic. Let PROP be a countable set of *propositional atoms*. The language $\mathcal{ML}$ of modal logic is the set of all formulae of the form $\varphi ::= p \mid \neg\varphi \mid$

$\varphi \wedge \varphi' \mid \Diamond\varphi$, where $p \in \mathrm{PROP}$. We use the well-known abbreviations $\vee$, $\rightarrow$, $\leftrightarrow$, $\top$ ("true"), and $\bot$ ("false"), as well as $\Box\varphi := \neg\Diamond\neg\varphi$.

A *(Kripke) model* is a triple $\mathcal{M} = (M, R, V)$, where M is a nonempty set of *states*, $R \subseteq M \times M$ is a binary relation — the *accessibility relation* —, and $V : \mathrm{PROP} \rightarrow \mathfrak{P}(M)$ is a function — the *valuation function*. The structure $\mathcal{F} = (M, R)$ is called a *frame*. Given a model $\mathcal{M} = (M, R, V)$ and a state $m \in M$, the *satisfaction relation* is defined by

$$\begin{aligned}
&\mathcal{M}, m \models p && \text{iff } \ m \in V(p),\ p \in \mathrm{PROP},\\
&\mathcal{M}, m \models \neg\varphi && \text{iff } \ \mathcal{M}, m \not\models \varphi,\\
&\mathcal{M}, m \models \varphi \wedge \psi && \text{iff } \ \mathcal{M}, m \models \varphi \ \&\ \mathcal{M}, m \models \psi,\\
&\mathcal{M}, m \models \Diamond\psi && \text{iff } \ \exists n \in M (mRn \ \&\ \mathcal{M}, n \models \psi).
\end{aligned} \tag{2.1}$$

A formula φ is *satisfiable* if there exist a model $\mathcal{M} = (M, R, V)$ and a state $m \in M$, such that $\mathcal{M}, m \models \varphi$. If all states from $\mathcal{M}$ satisfy φ, we write $\mathcal{M} \models \varphi$ and say that φ is *globally satisfied* by $\mathcal{M}$.

Temporal Logic. The language of temporal logic (tense logic) is the set of all formulae of the form $\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \mathsf{F}\varphi \mid \mathsf{P}\varphi$, where $p \in \mathrm{PROP}$. It is common practice to use the abbreviations $\mathsf{G}\varphi := \neg\mathsf{F}\neg\varphi$ and $\mathsf{H}\varphi := \neg\mathsf{P}\neg\varphi$. The operator F replaces $\Diamond$, hence satisfaction for F-formulae is defined as in (2.1). In the case of P-formulae, the term mRn must be replaced by nRm.

Whenever one wants to speak not only of states accessible from the actual state, but also of states "between" the actual and some accessible state, one can make use of the binary operators U ("until") and S ("since"), for which satisfaction is defined by

$$\begin{aligned}
&\mathcal{M}, m \models \mathsf{U}(\varphi, \psi) \ \text{ iff } \ \exists n \in M \big(mRn \ \&\ \mathcal{M}, n \models \varphi \ \&\ \forall s \in M (mRsRn \Rightarrow \mathcal{M}, s \models \psi)\big),\\
&\mathcal{M}, m \models \mathsf{S}(\varphi, \psi) \ \text{ iff } \ \exists n \in M \big(nRm \ \&\ \mathcal{M}, n \models \varphi \ \&\ \forall s \in M (nRsRm \Rightarrow \mathcal{M}, s \models \psi)\big).
\end{aligned}$$

The U/S language is strictly stronger than the basic temporal language in the sense that F and P can be expressed by U and S $\big($e. g. $\mathsf{F}\varphi = \mathsf{U}(\varphi, \top)\big)$, but not vice versa.

In [3], a variant of the U/S operators, U^+ and S^+, is introduced. Satisfaction for U^+ (analogously for S^+) is defined by

$$\begin{aligned}
&\mathcal{M}, m \models \mathsf{U}^+(\varphi, \psi)\\
&\quad \text{iff } \ \exists n \in M \big(mRn \ \&\ \mathcal{M}, n \models \varphi \ \&\ \forall s \in M (mR^+sR^+n \Rightarrow \mathcal{M}, s \models \psi)\big),
\end{aligned} \tag{2.2}$$

where R^+ is the transitive closure of R. With the help of U^+ and S^+, the authors of [3] "simulated" transitive frames syntactically. We go a step further and define another modification, U^{++} and S^{++}, with the satisfaction relation from (2.2), where the last remaining term mRn is replaced by mR^+n. This temporal language is an even closer simulation of transitivity, as we will see in Section 5.

Hybrid Logic. As indicated in the previous section, *the* hybrid language does not exist. Rather there are several extensions of the modal language allowing

for explicit references to states and therefore being called hybrid. We introduce the richest of those hybrid languages that will interest us in this paper. The definitions and notations are taken from [2, 3].

Let NOM be a countable set of *nominals*, SVAR be a countable set of *state variables*, and ATOM = PROP $\cup$ NOM $\cup$ SVAR. It is common practice to write propositional atoms as $p, q, \ldots$, nominals as $i, j, \ldots$, and state variables as $x, y, \ldots$ The *full hybrid language* $\mathcal{HL}^{\downarrow,@}$ is the set of all formulae of the form $\varphi ::= a \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \Diamond\varphi \mid @_t\varphi \mid \downarrow x.\varphi$, where $a \in$ ATOM, $t \in$ NOM $\cup$ SVAR, and $x \in$ SVAR.

A formula is called *pure* iff it contains no propositional atoms; *nominal-free* iff it contains no nominals; and a *sentence* iff it contains no free state variables. (*Free* and *bound* are defined as usual; the only binding operator here is $\downarrow$.)

A *hybrid model* is a Kripke model with the valuation function V extended to PROP $\cup$ NOM, where for all $i \in$ NOM, $|V(i)| = 1$. Whenever it is clear from the context, we will omit the word "hybrid" when referring to models. In order to evaluate $\downarrow$-formulae, an *assignment* $g :$ SVAR $\to M$ *for* $\mathcal{M}$ is necessary. Given an assignment g, a state variable x and a state m, *an* x*-variant* g^x_m *of* g is defined by $g^x_m(x) = m$ and $g^x_m(x') = g(x')$ for all $x' \neq x$. For any atom a, let $[V, g](a) = \{g(a)\}$ if $a \in$ SVAR, and $V(a)$, otherwise. The satisfaction relation for hybrid formulae is defined by

$$
\begin{array}{lll}
\mathcal{M}, g, m \models a & \text{iff} & m \in [V, g](a),\ a \in \text{ATOM}, \\
\mathcal{M}, g, m \models \neg\varphi & \text{iff} & \mathcal{M}, g, m \not\models \varphi, \\
\mathcal{M}, g, m \models \varphi \wedge \psi & \text{iff} & \mathcal{M}, g, m \models \varphi \ \&\ \mathcal{M}, g, m \models \psi, \\
\mathcal{M}, g, m \models \Diamond\varphi & \text{iff} & \exists n \in M (mRn \ \&\ \mathcal{M}, g, n \models \varphi), \\
\mathcal{M}, g, m \models @_t\varphi & \text{iff} & \mathcal{M}, g, n \models \varphi \ \&\ [V, g](t) = \{n\}, \\
\mathcal{M}, g, m \models \downarrow x.\varphi & \text{iff} & \mathcal{M}, g^x_m, m \models \varphi.
\end{array}
$$

A formula is *satisfiable* if there exist a model $\mathcal{M} = (M, R, V)$, an assignment g for $\mathcal{M}$, and a state $m \in M$, such that $\mathcal{M}, g, m \models \varphi$.

We sometimes use the "somewhere" modality E having the interpretation $\mathcal{M}, g, m \models \mathsf{E}\varphi$ iff $\exists n \in M (\mathcal{M}, g, n \models \varphi)$. In this case, @ is needless, because $@_t\,\varphi$ can be expressed by $\mathsf{E}(t \wedge \varphi)$.

First-order Logic. Modal and hybrid logic can be embedded into fragments of first-order logic (FOL). We will always use the standard notation of FOL.

We will make use of certain fragments of FOL and denote them in the style of [10]: $[\text{all}, (u, 1)]$, where $u \in \omega$. This notation stands for the fragment without equality, without function symbols, and with no other relation symbols than one binary and u unary ones. We denote the satisfiability problem for such a fragment by $[\text{all}, (u, 1)]$-SAT and $[\text{all}, (u, 1)]$-trans-SAT, where the latter requires that the binary relation symbol is interpreted by a transitive relation.

The *Standard Translation* ST [31] embeds hybrid logic into FOL and consists of two functions ST_x and ST_y defined recursively. Since ST_y is obtained from ST_x by exchanging x and y, we only give ST_x here.

$$\begin{aligned}
\mathrm{ST}_x(p) &= P(x), & \mathrm{ST}_x(\Diamond\varphi) &= \exists y\big(xRy \wedge \mathrm{ST}_y(\varphi)\big),\\
\mathrm{ST}_x(t) &= t{=}x, & \mathrm{ST}_x(@_t\varphi) &= \exists y\big(y{=}t \wedge \mathrm{ST}_y(\varphi)\big),\\
\mathrm{ST}_x(\neg\varphi) &= \neg\,\mathrm{ST}_x(\varphi), & \mathrm{ST}_x(\downarrow v.\varphi) &= \exists v\big(x{=}v \wedge \mathrm{ST}_x(\varphi)\big),\\
\mathrm{ST}_x(\varphi\wedge\psi) &= \mathrm{ST}_x(\varphi)\wedge\mathrm{ST}_x(\psi), & \mathrm{ST}_x(\mathsf{E}\varphi) &= \exists y\big(\,\mathrm{ST}_y(\varphi)\big),
\end{aligned}$$

where $p \in \mathrm{PROP}$, $t \in \mathrm{NOM} \cup \mathrm{SVAR}$, and $v \in \mathrm{SVAR}$.

Properties of Models and Frames. Let $\mathcal{M} = (M, R, V)$ be a (Kripke or hybrid) model with the underlying frame $\mathcal{F} = (M, R)$. By R^+ we denote the transitive closure of R. For any subset $M' \subseteq M$, we write $R{\restriction_{M'}}$ and $V{\restriction_{M'}}$ for the restrictions of R and V to M'. We will refer to *transitive frames* or *linear frames* whenever we mean frames whose accessibility relation is transitive or a linear order, respectively. A *linear order* is an irreflexive, transitive, and trichotomous $\big(\forall xy(xRy$ or $x{=}y$ or $yRx)\big)$ relation. The frame $\mathcal{F}$ is a *tree* iff it is acyclic and connected, and every point has at most one R-predecessor. A *transitive tree* is any (M, R^+), where (M, R) is a tree.

Satisfiability Problems. Whenever we leave one or more operators out of the hybrid language, we omit the according superscript of $\mathcal{HL}$. If we proceed to a hybrid tense language, we add the suitable temporal operator(s) as subscript(s) to $\mathcal{HL}$. Analogously, when equipping the modal language with additional operators, we add them as sub- or superscripts to $\mathcal{ML}$.

For any hybrid language $\mathcal{HL}^x_y$, the *satisfiability problem* $\mathcal{HL}^x_y$-SAT is defined as follows: Given a formula $\varphi \in \mathcal{HL}^x_y$, do there exist a hybrid model $\mathcal{M}$, an assignment g for $\mathcal{M}$, and a state $m \in M$ such that $\mathcal{M}, g, m \models \varphi$? If $\downarrow$ is not in the considered language, then the assignment g may be left out of this formulation. If we only ask for *transitive models* (or *transitive trees* or *linear models*, respectively) satisfying φ, then we speak of $\mathcal{HL}^x_y$-trans-SAT (or $\mathcal{HL}^x_y$-tt-SAT, or $\mathcal{HL}^x_y$-lin-SAT, respectively). As an example for our notation, the satisfiability problem over transitive frames for the hybrid temporal $\downarrow$ language is denoted by $\mathcal{HL}^{\downarrow}_{\mathsf{F},\mathsf{P}}$-trans-SAT.

3 Deciding $\mathcal{HL}^{\downarrow}$ over Transitive Frames

In [2] Areces, Blackburn, and Marx proved that the downarrow operator $\downarrow$ turns the satisfiability problem for hybrid logics undecidable in general, even if no interaction with @ or P is allowed. We prove that undecidability vanishes if frames are required to be transitive.

Theorem 1 *The satisfiability problem for $\mathcal{HL}^{\downarrow}$ over transitive frames is complete for* NEXP.

Before we start with the proof we have a first look at $\mathcal{HL}^{\downarrow}$ over transitive frames. Obviously, it has no finite model property. E.g., the following sentence requires a model containing an infinite chain of states labeled p.

$$p \wedge \Diamond p \wedge \Box \Diamond p \wedge \Box \downarrow x. \neg \Diamond x$$

Neither is it always possible to find a model that is a transitive tree. But, in some way, we can get close to this. Although our models may contain cycles, transitivity ensures that all states in a cycle are pairwise connected. I.e., the subframe consisting of these states is complete. Therefore, we can view a model as consisting of maximal complete subframes and single states, that are connected in a transitive but acyclic fashion.

For every transitive model $\mathcal{M} = (M, R, V)$ we define $B(\mathcal{M}) = (M', R', V')$ to be the model defined as follows. First, we replace each maximal complete subframe with a single vertex. Second, we unravel the resulting structure into a (potentially infinite) transitive tree T. Then we replace each vertex of T by (a copy of) the clique of $\mathcal{M}$ from which it is derived. We will refer to the resulting model as the *block tree* of $\mathcal{M}$. Note that a block tree is not a tree but its clique frame is a transitive tree.

In the following we are often interested in the underlying tree structure of a block tree and refer to the cliques of a block tree as *nodes*. For a state s, we denote its node by u_s. We say that a node v *is below* a node u if the states of v are reachable from the states in u (but not vice versa) and a node v is a *child* of a node u, if v is below u but there is no node w below u and above v.

Likewise we use the terms tree, subtree, and leaf for block tree, sub block tree and leaf clique, respectively.

We have to be careful about how to treat nominals when unraveling a model. If $V(i) = \{s\}$ for a nominal i and a state $s \in M$, we define $V'(i)$ to be the set of states from M', that are copies via the unraveling of s. Therefore, $B(\mathcal{M})$ is not a model as defined in Section 2 because nominals may hold at more than one state, but by viewing i as a propositional atom it can be treated as a model. The satisfaction relation is not affected and it is therefore easy to see that this transformation preserves satisfaction of $\mathcal{HL}^{\downarrow}$-sentences: The relation associating each state of $\mathcal{M}$ with every copy in $B(\mathcal{M})$ is a quasi-injective bisimulation[3] [8].

Lemma 2 *For every transitive model $\mathcal{M}$ and every $\mathcal{HL}^{\downarrow}$-sentence φ:*

$$\mathcal{M} \models \varphi \iff B(\mathcal{M}) \models \varphi.$$

Note that we can always get a model for φ from $B(\mathcal{M})$ by joining the states labeled with the same nominals, but this model might be different from $\mathcal{M}$.

Before we show how to use this tree-like structure to decide $\mathcal{HL}^{\downarrow}$ over transitive frames, we focus on complete subframes and show that their size can be bounded.

[3] The notion of bisimulation has to be extended by requiring states carrying the same nominal to be related.

3.1 $\mathcal{HL}^{\downarrow}$ over Complete Frames

As complete subframes are a significant part of transitive models for $\mathcal{HL}^{\downarrow}$-sentences, we are now going to study the satisfiability problem of $\mathcal{HL}^{\downarrow}$ over complete frames. The most important result for our purpose is an exponential-size model property of $\mathcal{HL}^{\downarrow}$ over complete frames.

We want to start by giving some insight why this property holds. In complete frames, the accessibility relation does not distinguish different states. Of course, states can be told apart if they are labeled differently by propositions. But the number of different labelings is exponentially bounded in the size of the formula. To use more states, we have to distinguish states labeled equally. This can only be done by assigning names to these states. But the number of states we can distinguish in this way is bounded by the number of different state variables.

While intuition is clear, we can prove this bound by observing that $\mathcal{HL}^{\downarrow}$ over complete frames is equivalent to the *Monadic Class with equality* ($\mathsf{MC}_{=}$), the fragment of first-order logic with only unary predicates, equality, and no function symbols [10].

This result allows us to transfer complexity results and model properties for $\mathsf{MC}_{=}$ [10] to $\mathcal{HL}^{\downarrow}$ over complete frames.

Theorem 3 *$\mathcal{HL}^{\downarrow}$ over complete frames has the exponential-size model property and its satisfiability problem is complete for* NEXP.

The lower bound can be transfered directly to the case of transitive frames.

Corollary 4 *The satisfiability problem for $\mathcal{HL}^{\downarrow}$ over transitive frames is hard for* NEXP.

3.2 On Transitive Frames for $\mathcal{HL}^{\downarrow}$

Let us summarize what we have seen so far. For every $\mathcal{HL}^{\downarrow}$-formula satisfiable over transitive frames, instead of a transitive model we can consider its block tree. The size of the cliques in the block tree can be exponentially bounded in the size of the formula by Corollary 3.

The algorithm for testing $\mathcal{HL}^{\downarrow}$-satisfiability will essentially guess a model and verify that it is correct. As there is no finite model property, all models might be infinite. Nevertheless, we will show that, if the formula is satisfiable, there is always a model with a regular structure in which certain finite patterns are repeated infinitely often. This will allow us to find a finite representation of such a model.

To this end, Definition 5 captures the information about a state of a block tree that will be needed for the following. Intuitively, the φ-type of a state captures the information needed about its subtree in order to evaluate a subformula of a given formula φ at this state. Here, $\psi[free/\bot]$ is the sentence obtained from ψ by replacing every free variable by $\bot$ and $sub(\varphi)$ is the set of all subformulae of φ.

Definition 5 *Let φ be a $\mathcal{HL}^{\downarrow}$-sentence and $\mathcal{B} = (M, R, V)$ a block tree model which is a model of φ. The* φ-type *of a state $s \in M$ is the set of all sentences from $\{\psi[free/\bot] \mid \Diamond\psi \in sub(\varphi)\}$ that hold at some state in the subtree rooted at s.*

Note that states in the same clique have the same φ-type. Therefore, we can speak of the φ-type of a node. The type of a node is always a superset of the types of its children. More precisely, it is always the union of the types of the children together with the set of relevant formulae which hold in the node itself.

When evaluating a subformula of a $\mathcal{HL}^{\downarrow}$-sentence φ at some state s of a block tree, all we need to know about states strictly below u_s are the φ-types of the children of u_s. I.e., we can replace subtrees below u_s by subtrees of the same φ-type. In the following lemma, for a block tree $\mathcal{B}$ and two states s_1, s_2, $\mathcal{B}[s_1/s_2]$ denotes the block tree resulting from $\mathcal{B}$ by replacing the subtree rooted at s_1 by the subtree rooted at s_2. The result of this substitution is again a block tree.

Lemma 6 *Let φ be a $\mathcal{HL}^{\downarrow}$-sentence, $\mathcal{B} = (M, R, V)$ a block tree model of φ and s_1 and s_2 states of $\mathcal{M}$ such that there is a path from s_1 to s_2 but not vice versa. For every formula $\psi \in sub(\varphi)$, every state s_3 of $\mathcal{M}$ of the same φ-type as s_2, and every assignment g that maps all free variables in ψ to states in M preceding s_1:*

$$\mathcal{B}, g, s_1 \models \psi \iff \mathcal{B}[u_{s_2}/u_{s_3}], g, s_1 \models \psi.$$

Note that we restricted the choice of g only to those assignments that are really relevant when evaluating the sentence φ.

We can use the previous lemma to get some nice restrictions on the block trees under consideration. E.g., we can assume that for every sentence in the type of a node, there is a witness in the node itself or in one of its children.

Lemma 7 *Let φ be a $\mathcal{HL}^{\downarrow}$-sentence satisfiable over transitive frames. Then there is a block tree model $\mathcal{B}$ for φ, in which*

- *every node has at most $|\varphi|$ children,*
- *for every node u with φ-type t and every $\mathcal{HL}^{\downarrow}$-sentence $\psi \in t$, ψ holds at a state in u or at a state in a child of u, and*
- *on every path from the root, infinite or ending at a leaf, every φ-type occurs only once or infinitely often.*

3.3 Deciding $\mathcal{HL}^{\downarrow}$-SAT over Transitive Frames

We will now finish the proof of Theorem 1 by presenting a nondeterministic algorithm that decides $\mathcal{HL}^{\downarrow}$-SAT over transitive frames in exponential time, basically by guessing and verifying the finite representation of a block tree model for a given $\mathcal{HL}^{\downarrow}$-sentence φ.

Given a block tree $\mathcal{B}$ with the properties of Lemma 7, we get a finite representation as follows. For each path of $\mathcal{B}$ we consider the first node v that has the same type as its parent node u. We replace the subtree below v by a single state labeled with a reference to u. We need to keep v because it might be the only

witness for a formula in the φ-type of u (cf. Lemma 7). Clearly, the resulting structure is finite.

By Lemma 6 and Lemma 7, we can get a block tree model from this representation by replacing each reference with the subtree rooted at the referenced node, i.e., essentially by an unraveling.

Due to Lemma 6, the size of the representation can be reduced even further. If there are two nodes u and v of the same φ-type which are not on the same path and both are the first node of their type on their path from the root, we can replace the subtree rooted at v with the subtree of u. I.e., whenever two nodes have the same φ-type, we can assume that their generated subtrees are equal. We have to check them only once.

Such a representation can be described by a structure $(M \dot{\cup} C, R, V)$ such that the states in C have no outgoing edges, and a function f from C to M. Note that a state in C is a node of its own, in fact a leaf, and cannot be in the same complete subframe as a state in M. A state $s \in C$ stands for a repetition respectively duplication of the subtree rooted at $f(s)$, including states from C. This causes infinite repetition if s is below $f(s)$.

This observation can be reflected in our representation by replacing every duplicate with a reference. This causes every type to appear at most twice in the representation, thus the number of nodes is at most exponential.

Summing up, if φ is satisfiable, there is a representation for of a block tree model for φ of size at most exponential in the length of φ. The first step of the algorithm is to guess such a representation (step 1).

In order to obtain an algorithm which tests whether the representations indeed represents a model of φ, we describe how to modify the model checking algorithm MCFULL by Franceschet and de Rijke presented in [12] to do so. First, we deal with the states in C (step 2). Next, the model checking algorithm MCFULL is used on the states in M (step 3). We have to modify this algorithm in two respects. First, it has to use the information guessed for the states in C. Second, it should compute the φ-types of the states in M. To this end, it first evaluates the sentences resulting from subformulae of φ by replacing free variables with $\bot$. We call this modified algorithm MCFULL'. The changes are straightforward.

After running MCFULL' the algorithm has computed for each state the set of formulae that hold at this state. These sets depend on the guesses in Step 2. Therefore, the algorithm has to verify the consistency of these guesses. This can be done by comparing the φ-types of the states in C with the types of the referenced states (step 4).

The last two steps are easy. The algorithm checks that the φ-types are consistent and reject if this is not the case (step 5). Finally, it checks if φ holds at some state in M (step 6).

Our algorithm for $\mathcal{HL}^{\downarrow}$-satisfiability over transitive frames
1: Guess the representation of a block tree.
2: Guess a φ-type for every state in C.
3: Run MCFULL' for the states in M.
4: Compute the φ-type for every state in M referenced by a state in C.
5: Check for every state $s \in C$: $f(s)$ has the same φ-type as s. If not, reject.
6: Accept iff φ holds at some state in M.

Theorem 8 *The algorithm presented above decides $\mathcal{HL}^{\downarrow}$-satisfiability over transitive frames nondeterministically in exponential time.*

Proof. In Section 3.2, we have seen that for every satisfiable sentence φ there is a block tree model as described in Lemma 7. We have also seen how to represent this block tree in a finite manner. The algorithm can guess this representation and the φ-types of the states in C. The computation of the φ-types of the states in M works correctly, because we have a witness for every sentence in the type in our representation. This is by Lemma 7, which ensures that witnesses are in the node of the state or in one of its children. Therefore, we cut below these witnesses when building the finite representation. Consequently, the algorithm will accept.

On the other hand, if the algorithm accepts, it is straightforward to construct a block tree model from the guessed representation. The only critical point for soundness is the verification of the φ-types guessed in Step 2, more precisely, the computation of the φ-types of the states in M. First, the φ-type of some state $s \in M$ contains only sentences that hold at some state of the represented model below s. This can be assured by looking only at states in M and not at states in C. That the φ-type of s contains all sentences that hold below s can be assured by following the links represented by states in C.

The first two steps of the algorithm can be performed in exponential time because the representation is of at most exponential size. That Step 3 runs in exponential time follows from Theorem 4.5 of [12], the truth of which is not affected by our modifications. The time bounds for the other steps follow again from the exponential size bound of the representation. ❑

From Theorem 8 and Corollary 4 we can directly conclude Theorem 1.

4 Richer hybrid ↓ Logics over Transitive Frames and Transitive Trees

This section is concerned with satisfiability over transitive frames and transitive trees for extensions of $\mathcal{HL}^{\downarrow}$.

Our first point is that we cannot sustain decidability over transitive frames if we enrich $\mathcal{HL}^{\downarrow}$ with @ or the backward looking modality P. We prove undecidability in both cases, proceeding in two steps. First, we show that [all, (4, 1)]-trans-SAT is undecidable. This is done by a reduction from [all, (0, 1)]-SAT. The undecidability of the latter is a consequence of the undecidability of contained

traditional standard classes [10]. The second step consists of reduction from [all, (4, 1)]-trans-SAT to $\mathcal{HL}^{\downarrow,@}$-trans-SAT and $\mathcal{HL}^{\downarrow}_{\mathsf{F,P}}$-trans-SAT, respectively. To be more precise, the ranges of these reductions will be the fragments of the respective hybrid languages consisting of all nominal-free sentences.

Lemma 9 [all, (4, 1)]-trans-SAT *is undecidable.*

Proof. In order to obtain the required reduction from [all, (0, 1)]-SAT, we will transform a (not necessarily transitive) model satisfying α into a transitive one. Simply taking the transitive closure in most cases adds new pairs to the interpretation of the relation and is not sufficient for keeping the information which pairs were in the "old" relation and which pairs were not. This problem does not arise if we instead use a variation of the *zig-zag technique* successfully applied in [3] for a reduction between a modal and a hybrid language. The core idea of this technique is to simulate an R-step $t_1 R t_2$ in the original model $\mathcal{M} = (D, I)$ by a zig-zag transition in a model $\mathcal{M}' = (D', I')$, where $I'(R)$ is transitive, as shown in Figure 1.

We define a translation function $(\cdot)^t$ using four extra predicate symbols $0, 1, 2, 3$ as follows.

$$
\begin{aligned}
(xRy)^t &= \exists abc \big(xRa \wedge bRa \wedge bRc \wedge yRc \\
&\qquad \wedge 0(x) \wedge 1(a) \wedge 2(b) \wedge 3(c) \wedge 0(y)\big), \\
(\neg\alpha)^t &= \neg(\alpha^t), \\
(\alpha \wedge \beta)^t &= \alpha^t \wedge \beta^t, \\
(\exists x\, \alpha)^t &= \exists x \big(0(x) \wedge \alpha^t\big).
\end{aligned}
$$

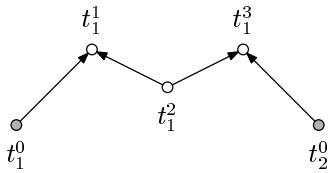


Fig. 1. A zig-zag transition.

The translation of the xRy-atoms exactly reflects the shown zig-zag transition. It is straightforward to prove the following claim: *For each formula α, α is satisfiable iff $f(\alpha)$ is satisfiable in some model that interprets R by a transitive relation.* Since $(\cdot)^t$ is an appropriate (even polynomial-time) reduction function, we have established undecidability for [all, (4, 1)]-trans-SAT. ❑

Theorem 10 $\mathcal{HL}^{\downarrow,@}$-trans-SAT *and* $\mathcal{HL}^{\downarrow}_{\mathsf{F,P}}$-trans-SAT *are undecidable.*

Proof. We reduce [all, (4, 1)]-trans-SAT to $\mathcal{HL}^{\downarrow,@}$-trans-SAT and $\mathcal{HL}^{\downarrow}_{\mathsf{F,P}}$-trans-SAT, respectively, invoking a spypoint argument (see [7, 2]). A spypoint is a state s of a hybrid model that sees all other states and is named by a fresh nominal i. Since our reduction will not make use of any nominals, we can establish this undecidability result for the nominal-free fragments of the hybrid languages in question. We simply treat i as a state variable and bind it to s.

We first treat the case of $\mathcal{HL}^{\downarrow,@}$ and define a translation function $(\cdot)^t$ from the first-order fragment to $\mathcal{HL}^{\downarrow,@}$ by $(xRy)^t = @_x \Diamond y$, $\big(P(x)\big)^t = @_x\, p$, $(\neg\alpha)^t = \neg(\alpha^t)$, $(\alpha \wedge \beta)^t = \alpha^t \wedge \beta^t$, and $(\exists x\, \alpha)^t = @_i \Diamond \downarrow x.\alpha^t$. The (polynomial) reduction function f is defined by $f(\alpha) = \downarrow i.(\neg\Diamond i \wedge \Diamond\alpha^t)$.

It is straightforward to show that each formula α is satisfiable iff $f(\alpha)$ is satisfiable, and to adopt the translation function $(\cdot)^t$ to the case of $\mathcal{HL}^{\downarrow}_{\mathsf{F},\mathsf{P}}$. ❑

The second point of this section is that over transitive trees, where decidability of $\mathcal{HL}^{\downarrow}$ is trivial, even the extension $\mathcal{HL}^{\downarrow,@}_{\mathsf{F},\mathsf{P}}$ is decidable. This is an immediate consequence of the decidability of the monadic second-order theory of the countably branching tree, SωS, [10]. However, we have to face a nonelementary lower bound in both cases $\mathcal{HL}^{\downarrow,@}$ and $\mathcal{HL}^{\downarrow}_{\mathsf{F},\mathsf{P}}$, which is obtained by a reduction from the nonelementarily decidable $\mathcal{HL}^{\downarrow}_{\mathsf{F},\mathsf{P}}$-lin-SAT [13].

Theorem 11 *$\mathcal{HL}^{\downarrow}_{\mathsf{F},\mathsf{P}}$-tt-SAT, $\mathcal{HL}^{\downarrow,@}$-tt-SAT, and $\mathcal{HL}^{\downarrow,@}_{\mathsf{F},\mathsf{P}}$-tt-SAT are nonelementary decidable.*

5 Hybrid Until/Since Logic over Transitive Frames and Transitive Trees

In this section, we will consider $\mathcal{HL}^{\mathsf{E}}_{\mathsf{U},\mathsf{S}}$-trans-SAT and $\mathcal{HL}^{\mathsf{E}}_{\mathsf{U},\mathsf{S}}$-tt-SAT. In [3] it was shown that $\mathcal{HL}^{@}_{\mathsf{U},\mathsf{S}}$-SAT is EXP-complete.

As for the lower bound, we establish a result as general as possible, namely EXP-hardness of $\mathcal{ML}_{\mathsf{U}}$-trans-SAT and $\mathcal{ML}_{\mathsf{U}}$-tt-SAT.

Theorem 12 *$\mathcal{ML}_{\mathsf{U}}$-trans-SAT and $\mathcal{ML}_{\mathsf{U}}$-tt-SAT are* EXP*-hard.*

Proof. We will reduce the *global* satisfiability problem for $\mathcal{ML}$ to both our (local) problems $\mathcal{ML}_{\mathsf{U}}$-trans-SAT and $\mathcal{ML}_{\mathsf{U}}$-tt-SAT using the same reduction function. The global satisfiability problem is defined by $\mathcal{ML}$-GLOBSAT $= \{\varphi \in \mathcal{ML} \mid \varphi$ is true in *all* states of some Kripke model $\mathcal{M}\}$. Its EXP-completeness is a direct consequence of the EXP-completeness of $\mathcal{ML}^{\mathsf{E}}$-SAT [28].

It may seem difficult to try reducing this problem over *arbitrary* frames to our satisfiability problem over *transitive* frames. The critical point lies in making a non-transitive model transitive: taking the transitive closure of its relation forces us to add new accessibilities that would disturb satisfaction of $\neg\Diamond$-formulae. Fortunately though, the U operator can make us distinguish the accessibilities from the original model from those that have been added to make the relation transitive. Hence, a translation of $\Diamond\varphi$ should demand: "Make sure that the actual state sees a state in which the translation of φ holds, and that there is no state in between." This translates as $\mathsf{U}(\varphi^t, \bot)$ into the modal language.

To construct the required reduction, we define a translation function $(\cdot)^t : \mathcal{ML} \to \mathcal{ML}_{\mathsf{U}}$ by $p^t = p$, $p \in$ PROP; $(\varphi \wedge \psi)^t = \varphi^t \wedge \psi^t$; $(\neg\varphi)^t = \neg(\varphi^t)$; and $(\Diamond\varphi)^t = \mathsf{U}(\varphi^t, \bot)$. Using $(\cdot)^t$, we construct a reduction function $f : \mathcal{ML} \to \mathcal{ML}_{\mathsf{U}}$ —that is clearly computable in polynomial time— via $f(\varphi) = \varphi^t \wedge \Box\varphi^t$. It is straightforward to prove the following two claims for each $\varphi \in \mathcal{ML}$.

(1) If $\varphi \in \mathcal{ML}$-GLOBSAT, then $f(\varphi) \in \mathcal{ML}_{\mathsf{U}}$-tt-SAT.

(2) If $f(\varphi) \in \mathcal{ML}_{\mathsf{U}}$-trans-SAT, then $\varphi \in \mathcal{ML}$-GLOBSAT. ❑

The upper bounds for $\mathcal{HL}^{\mathsf{E}}_{\mathsf{U},\mathsf{S}}$-trans-SAT and $\mathcal{HL}^{\mathsf{E}}_{\mathsf{U},\mathsf{S}}$-tt-SAT require separate treatment. As for $\mathcal{HL}^{\mathsf{E}}_{\mathsf{U},\mathsf{S}}$-trans-SAT, we use an embedding into an appropriate fragment of first-order logic. In order to eliminate transitivity, we "simulate" this semantic property by syntactic means, namely using the operators U^{++} and S^{++} defined in Section 2.

Lemma 13 *For any* $X \subseteq \{@, \mathsf{E}\}$, $\mathcal{HL}^{X}_{\mathsf{U},\mathsf{S}}$-trans-SAT *and* $\mathcal{HL}^{X}_{\mathsf{U}^{++},\mathsf{S}^{++}}$-SAT *are polynomially reducible to each other.*

Now it is not difficult anymore to obtain a 2EXP upper bound for $\mathcal{HL}^{@}_{\mathsf{U},\mathsf{S}}$-trans-SAT by an embedding into the loosely μ-guarded fragment μLGF of first-order logic whose satisfiability problem is 2EXP-complete [16]. Only the E operator requires a more careful analysis.

Theorem 14 $\mathcal{HL}^{\mathsf{E}}_{\mathsf{U},\mathsf{S}}$-trans-SAT *is in* 2EXP.

To show that $\mathcal{HL}^{\mathsf{E}}_{\mathsf{U},\mathsf{S}}$-tt-SAT is in EXP, too, we use an embedding into $\mathcal{PDL}_{\text{tree}}$, the propositional dynamic logic for sibling-ordered trees introduced in [20, 21]. Finite, node-labelled, sibling-ordered trees are the logical abstraction of XML (eXtensible Markup Language) documents. In [1], it was shown that satisfiability of $\mathcal{PDL}_{\text{tree}}$ formulae at the root of finite trees ($\mathcal{PDL}_{\text{tree}}$-SAT) is decidable in EXP.

Theorem 15 $\mathcal{HL}^{\mathsf{E}}_{\mathsf{U},\mathsf{S}}$-tt-SAT *is in* EXP.

6 Conclusion

We have established two groups of complexity results for hybrid logics over two temporally relevant frame classes, i. e. that of transitive frames and that of transitive trees.

First we have "tamed" $\mathcal{HL}^{\downarrow}$ over transitive frames showing that $\mathcal{HL}^{\downarrow}$-trans-SAT is NEXP-complete. We have shown that, in contrast, $\mathcal{HL}^{\downarrow,@}$-trans-SAT and $\mathcal{HL}^{\downarrow}_{\mathsf{F},\mathsf{P}}$-trans-SAT are undecidable. Over transitive trees, three enrichments of $\mathcal{HL}^{\downarrow}$ are decidable, albeit nonelementarily, namely $\mathcal{HL}^{\downarrow,@}$-tt-SAT, $\mathcal{HL}^{\downarrow}_{\mathsf{F},\mathsf{P}}$-tt-SAT, and $\mathcal{HL}^{\downarrow,@}_{\mathsf{F},\mathsf{P}}$-tt-SAT.

In the second part of our work, we established an EXP lower bound for $\mathcal{ML}_{\mathsf{U}}$-trans-SAT and $\mathcal{ML}_{\mathsf{U}}$-tt-SAT and matched the latter with an EXP upper bound for $\mathcal{HL}^{\mathsf{E}}_{\mathsf{U},\mathsf{S}}$-tt-SAT. This is the same complexity as for satisfiability over *arbitrary* frames for the same language. As for $\mathcal{HL}^{\mathsf{E}}_{\mathsf{U},\mathsf{S}}$-trans-SAT, we have given a 2EXP upper bound. We conjecture EXP-completeness.

Over linear frames, the complexity of hybrid U/S logic is still open. As a special case, satisfiability of $\mathcal{HL}^{@}_{\mathsf{U},\mathsf{S}}$ over $(\mathbb{N}, >)$ is known to be PSPACE-complete [13]. Furthermore, in [13] it was shown that $\mathcal{ML}_{\mathsf{U}}$ is PSPACE-complete over general linear time.

Another open question is the complexity of $\mathcal{HL}^{\downarrow,@}$-lin-SAT and $\mathcal{HL}^{\downarrow}_{\mathsf{F},\mathsf{P}}$-lin-SAT (see the question marks in Table 1). Merely decidability follows from decidability of $\mathcal{HL}^{\downarrow,@}_{\mathsf{F},\mathsf{P}}$-lin-SAT [13].

Acknowledgments

We thank Balder ten Cate, Massimo Franceschet, Maarten Marx, and the anonymous referees for helpful suggestions, discussions, and comments.

References

1. Afanasiev, L., P. Blackburn, I. Dimitriou, B. Gaiffe, E.Goris, M. Marx and M. de Rijke, *PDL for ordered trees*, Journal of Applied Non-Classical Logics **15** (2005), pp. 115–135.
2. Areces, C., P. Blackburn and M. Marx, *A road-map on complexity for hybrid logics*, in: J. Flum and M. Rodríguez-Artalejo, editors, *CSL*, Lecture Notes in Computer Science **1683** (1999), pp. 307–321.
3. Areces, C., P. Blackburn and M. Marx, *The computational complexity of hybrid temporal logics*, Logic Journal of the IGPL **8** (2000), pp. 653–679.
4. Areces, C., P. Blackburn and M. Marx, *Hybrid logics: Characterization, interpolation and complexity*, J. Symb. Log. **66** (2001), pp. 977–1010.
5. Blackburn, P., *Representation, reasoning, and relational structures: a hybrid logic manifesto*, Logic Journal of the IGPL **8** (2000), pp. 339–365.
6. Blackburn, P., M. de Rijke and Y. Venema, "Modal Logic," Cambridge Tracts in Theoretical Computer Science **53**, Cambridge University Press, 2001.
7. Blackburn, P. and J. Seligman, *Hybrid languages*, Journal of Logic, Language and Information **4** (1995), pp. 251–272.
8. Blackburn, P. and J. Seligman, *What are hybrid languages?*, in: M. Kracht, M. de Rijke, H. Wansing and M. Zakharyaschev, editors, *Advances in Modal Logic*, CSLI Publications **1**, Stanford University, 1998 pp. 41–62.
9. Blackburn, P. and M. Tzakova, *Hybrid languages and temporal logic*, Logic Journal of the IGPL **7** (1999), pp. 27–54.
10. Börger, E., E. Grädel and Y. Gurevich, "The Classical Decision Problem," Perspectives of Mathematical Logic, Springer-Verlag, 1997.
11. Dawar, A. and M. Otto, *Modal characterisation theorems over special classes of frames*, in: *LICS* (2005), pp. 21–30.
12. Franceschet, M. and M. de Rijke, *Model checking hybrid logics (with an application to semistructured data)*, Journal of Applied Logic (2005).
13. Franceschet, M., M. de Rijke and B.-H. Schlingloff, *Hybrid logics on linear structures: Expressivity and complexity*, in: *TIME* (2003), pp. 166–173.
14. Ganzinger, H., C. Meyer and M. Veanes, *The two-variable guarded fragment with transitive relations*, in: *Logic in Computer Science*, 1999, pp. 24–34.
15. Goranko, V., *Hierarchies of modal and temporal logics with reference pointers*, Journal of Logic, Language and Information **5** (1996), pp. 1–24.
16. Grädel, E. and I. Walukiewicz, *Guarded fixed point logic*, in: *LICS*, 1999, pp. 45–54.
17. Immerman, N., A. M. Rabinovich, T. W. Reps, S. Sagiv and G. Yorsh, *The boundary between decidability and undecidability for transitive-closure logics*, in: J. Marcinkowski and A. Tarlecki, editors, *CSL*, Lecture Notes in Computer Science **3210** (2004), pp. 160–174.

18. Kieronski, E., *EXPSPACE-complete variant of guarded fragment with transitivity*, in: H. Alt and A. Ferreira, editors, *STACS*, Lecture Notes in Computer Science **2285** (2002), pp. 608–619.
19. Kieronski, E., *The two-variable guarded fragment with transitive guards is 2EXP-TIME-hard*, in: A. D. Gordon, editor, *FoSSaCS*, Lecture Notes in Computer Science **2620** (2003), pp. 299–312.
20. Kracht, M., *Syntactic codes and grammar refinement*, Journal of Logic, Language and Information **4** (1995), pp. 41–60.
21. Kracht, M., *Inessential features*, in: C. Retoré, editor, *LACL*, Lecture Notes in Computer Science **1328** (1996), pp. 43–62.
22. Ladner, R. E., *The computational complexity of provability in systems of modal propositional logic*, SIAM J. Comput. **6** (1977), pp. 467–480.
23. Ono, H. and A. Nakamura, *On the size of refutation Kripke models for some linear modal and tense logics*, Studia Logica **34** (1980), pp. 325–333.
24. Papadimitriou, C. H., "Computational Complexity," Addison-Wesley, 1994.
25. Prior, A., "Past, Present and Future," Oxford University Press, 1967.
26. Reynolds, M., *The complexity of the temporal logic with "until" over general linear time*, J. Comput. Syst. Sci. **66** (2003), pp. 393–426.
27. Sistla, A. P. and E. M. Clarke, *The complexity of propositional linear temporal logics*, J. ACM **32** (1985), pp. 733–749.
28. Spaan, E., "Complexity of modal logics," Ph.D. thesis, ILLC, University of Amsterdam (1993).
29. Spaan, E., *The complexity of propositional tense logics*, in: M. d. Rijke, editor, *Diamonds and Defaults*, Kluwer, 1993 pp. 287–307.
30. Szwast, W. and L. Tendera, *On the decision problem for the guarded fragment with transitivity*, in: *LICS*, 2001, pp. 147–156.
31. ten Cate, B. and M. Franceschet, *On the complexity of hybrid logics with binders*, in: C.-H. L. Ong, editor, *CSL*, Lecture Notes in Computer Science **3634** (2005), pp. 339–354.

Two Tableau-Based Decision Procedures for Hybrid Logic

Thomas Bolander

Dep. Informatics and Mathematical Modelling
Building 322
Technical University of Denmark
DK-2800 Lyngby, Denmark
tb@imm.dtu.dk

Torben Braüner

Department of Computer Science
Roskilde University
P.O. Box 260
DK-4000 Roskilde, Denmark
torben@ruc.dk

Abstract

It is well-known that various hybrid logics without binders are decidable, but decision procedures are usually not based on tableau systems. In this paper we give two tableau-based decision procedures for a very expressive hybrid logic including the universal modality. The decision procedures make use of so-called loop-checks which is a technique standardly used in connection with tableau systems for other logics, namely prefixed tableau systems for transitive modal logics, as well as prefixed tableau systems for certain description logics.

Keywords: Hybrid logic, modal logic, universal modality, tableau systems, decision procedures.

1 Introduction

The hybrid logic we consider in the present paper is obtained by adding to ordinary modal logic further expressive power in the form of a second sort of propositional symbols called nominals, and moreover, by adding so-called satisfaction operators as well as the universal modality. A nominal is assumed to be true at exactly one world, so in this sense a nominal refers to a world. If a is a nominal and ϕ is an arbitrary formula, then a new formula $a : \phi$ called a satisfaction statement can be formed. The part a: of $a : \phi$ is called a

satisfaction operator (some authors often use the notation $@_a$ instead of a:). The satisfaction statement $a{:}\phi$ is true (at any world) if and only if the formula ϕ is true at one particular world, namely the world at which the nominal a is true. The truth-condition of the universal modality E is that $E\phi$ is true (at any world) if and only if there exists a world at which the formula ϕ is true.

It is well-known that the hybrid logic described above is decidable, see [1], but decision procedures are usually not tableau-based. In fact, we are only aware of one published tableau-based decision procedure for hybrid logic, namely the one given in Miroslava Tzakova's paper [11]. However, a number of crucial details are missing in Tzakova's termination proof, and we did not find any way to fill out these details. In the present paper we give a tableau system along the lines of Tzakova's system extended with the universal modality, and give a terminating systematic tableau construction algorithm for the system. Our tableau construction algorithm is very different from Tzakova's algorithm. An essential feature of our algorithm is that it makes use of loop-checks. Besides Tzakova's tableau system, we also consider a tableau system given by Patrick Blackburn in the paper [2]. Decision procedures are not considered in Blackburn's paper. We give a terminating systematic tableau construction algorithm for Blackburn's system extended with the universal modality, again with the essential feature that it makes use of loop-checks. Analogous results follows for the weaker hybrid logic obtained by ignoring the universal modality.

The paper is structured as follows. In the second section we recapitulate the basics of hybrid logic, in the third section we give the decision procedure for our version of Tzakova's tableau system, and in the fourth section we give the decision procedure for Blackburn's tableau system. In the final section we discuss some related work.

2 The basics of hybrid logic

We shall in many cases adopt the terminology of [3] and [1]. The hybrid logic we consider is obtained by adding a second sort of propositional symbols called *nominals* to ordinary modal logic. It is assumed that a set of ordinary propositional symbols and a countably infinite set of nominals are given. The sets are assumed to be disjoint. The metavariables $p, q, r, \ldots$ range over ordinary propositional symbols and $a, b, c, \ldots$ range over nominals. Besides nominals, an operator a: called a *satisfaction operator* is added for each nominal a, and furthermore, the universal modality E is added. The formulas of hybrid modal logic are defined by the grammar

$$S ::= p \mid a \mid \neg S \mid S \wedge S \mid \Diamond S \mid a : S \mid ES$$

where p is an ordinary propositional symbol and a is a nominal. In what follows, the metavariables ϕ, ψ, χ, ... range over formulas. Formulas of the form $a : \phi$ are called *satisfaction statements*, cf. a similar notion in [2]. The operator $\Box$ and the propositional connectives not taken as primitive are defined as usual.

We now define models.

Definition 2.1 A *model* for hybrid logic is a tuple (W, R, V) where

(i) W is a non-empty set;

(ii) R is a binary relation on W; and

(iii) V is a function that to each pair consisting of an element of W and an ordinary propositional symbol assigns an element of $\{0, 1\}$.

The elements of W are called *worlds* and the relation R is called an *accessibility relation*. An *assignment* for a model $\mathcal{M} = (W, R, V)$ is a function g that to each nominal assigns an element of W. Given assignments g' and g, $g' \stackrel{a}{\sim} g$ means that g' agrees with g on all nominals save possibly a. The relation $\mathcal{M}, g, w \models \phi$ is defined inductively, where g is an assignment, w is an element of W, and ϕ is a formula.

$$\mathcal{M}, g, w \models p \text{ iff } V(w, p) = 1$$

$$\mathcal{M}, g, w \models a \text{ iff } w = g(a)$$

$$\mathcal{M}, g, w \models \neg\phi \text{ iff not } \mathcal{M}, g, w \models \phi$$

$$\mathcal{M}, g, w \models \phi \wedge \psi \text{ iff } \mathcal{M}, g, w \models \phi \text{ and } \mathcal{M}, g, w \models \psi$$

$$\mathcal{M}, g, w \models a : \phi \text{ iff } \mathcal{M}, g, g(a) \models \phi$$

$$\mathcal{M}, g, w \models \Diamond\phi \text{ iff for some } v \in W,\ wRv \text{ and } \mathcal{M}, g, v \models \phi$$

$$\mathcal{M}, g, w \models E\phi \text{ iff for some } v \in W,\ \mathcal{M}, g, v \models \phi$$

By convention $\mathcal{M}, g \models \phi$ means $\mathcal{M}, g, w \models \phi$ for every element w of W and $\mathcal{M} \models \phi$ means $\mathcal{M}, g \models \phi$ for every assignment g. A formula ϕ is *valid* if and only if $\mathcal{M} \models \phi$ for any model $\mathcal{M}$.

3 Tzakova's system extended with the universal modality

Tzakova's system [11] is a prefixed tableau calculus (see the book [4] for the basics of tableau systems). This means that the formulas occurring in the tableau rules are *prefixed formulas* on the form $\sigma\phi$, where ϕ is a formula of hybrid modal logic and σ belongs to some fixed countably infinite set of symbols called *prefixes*. In addition, the tableau rules contain *accessibility formulas* on the form $\sigma < \sigma'$ where σ and σ' are prefixes. The rules of the tableau system are given in Figure 1. Actually, the given tableau system is a modified version of Tzakova's calculus. The calculus is simplified by replacing Tzakova's rules (S-Identifying) and (L-Identifying) by (Id). Furthermore, the rule (Labeling) has been deleted. Our calculus also differs from Tzakova's by including the rules for the universal modality. However, we will still refer to this calculus as *Tzakova's system*. A *tableau* in Tzakova's system is a well-

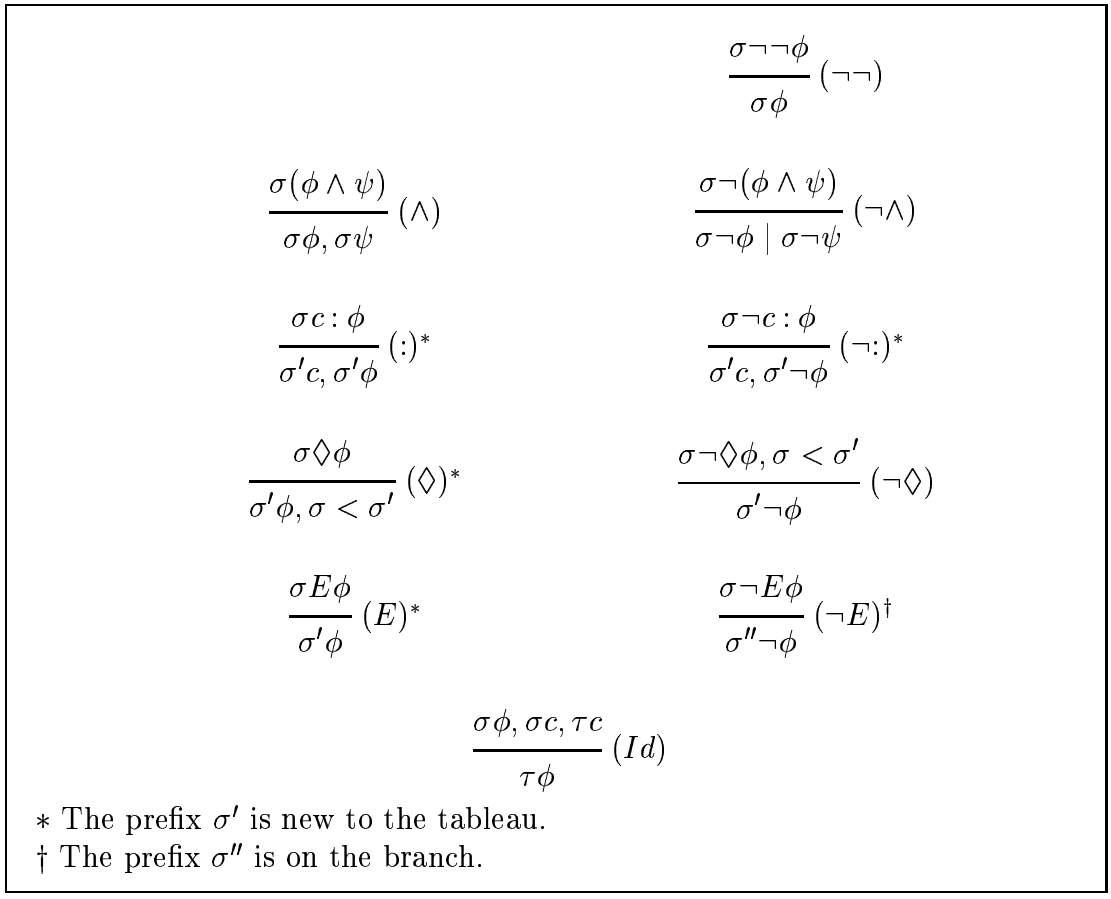


Fig. 1. Modified version of Tzakova's tableau rules

founded tree in which each node is either a prefixed formula or an accessibility formula, and the edges represent applications of tableau rules in the usual way. We impose the following conventions on the application of the rules in tableau constructions.

- In constructing a tableau, the rules (:), ($\neg$:), ($\Diamond$) and (E) are never applied to the same premise twice on the same branch.
- A formula is never added to a tableau branch where it already occurs.

3.1 Some properties of the system

Tzakova's system satisfies the following basic properties.

Lemma 3.1 (Quasi-subformula property) *If a formula $\sigma\phi$ occurs in a tableau with root $\sigma_0\phi_0$ then either ϕ or $\neg\phi$ is a subformula of ϕ_0.*

Proof. Follows immediately from the rules in Figure 1. □

Note the following consequence of Lemma 3.1: For any given tableau $\mathcal{T}$, the set $\{\phi \mid \sigma\phi \text{ occurs in } \mathcal{T}\}$ is finite. We will use this fact a number of times in the proofs below.

The only way new prefixes can be introduced to a tableau is by using one of the rules (:), ($\neg$:), ($\Diamond$) or (E). These introduce a new prefix σ' from a given prefix σ. Let Θ be a branch of a tableau. If a new prefix σ' is introduced by applying one of the rules (:), ($\neg$:), ($\Diamond$) or (E) to a prefixed formula $\sigma\phi$ then we say that σ' is *generated* by σ with respect to Θ, and we write $\sigma <_\Theta \sigma'$. This gives us a binary relation $<_\Theta$ on the prefixes occurring on Θ.

Proposition 3.2 *Let Θ be a branch of a tableau. Let N^Θ be the set of prefixes occurring on Θ. The graph $(N^\Theta, <_\Theta)$ is a finitely branching tree.*

Proof. That the graph is a tree follows from the fact that each prefix in N^Θ can be generated by at most one other prefix, and that all prefixes in N^Θ must have the prefix of the root formula as an ancestor. That the graph is finitely branching follows from the fact that for any given prefix σ the set $\{\phi \mid \sigma\phi \text{ occurs on } \Theta\}$ is finite (cf. Lemma 3.1), and each of these finitely many formulas $\sigma\phi$ can generate at most one new successor prefix σ' (by applying one of the rules (:), ($\neg$:), ($\Diamond$) or (E) to $\sigma\phi$). □

3.2 Systematic tableau construction

Before giving the systematic tableau construction algorithm we need a definition.

Definition 3.3 Let σ and τ be prefixes occurring at a branch Θ of a tableau. The prefix σ is *included* in the prefix τ with respect to Θ if for any hybrid formula ϕ, if $\sigma\phi$ occurs on Θ then $\tau\phi$ also occurs on Θ. If σ is included in τ and τ has its first occurrence on Θ no later than σ, then we write $\sigma \subseteq_\Theta \tau$.

We are now ready to define the systematic tableau construction algorithm. The algorithm we present is non-deterministic, but can easily be made deterministic by introducing suitable well-orderings.

Definition 3.4 (Tableau construction algorithm) Let ϕ be the formula whose validity we have to decide. By induction we define a sequence $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2, \ldots$ of finite tableaus, where each tableau is obtained from the previous by applying one of the tableau rules. Define $\mathcal{T}_0$ to be the tableau constituted by the single prefixed formula $\sigma\neg\phi$, where σ is any prefix. Given a tableau $\mathcal{T}_i$, we then define $\mathcal{T}_{i+1}$ to be the tableau obtained by applying some rule to $\mathcal{T}_i$ subject to the following restriction:

($\mathcal{R}$) The rules (:), ($\neg$:), ($\Diamond$) and (E) are not applied to a formula occurrence $\sigma\phi$ at a branch Θ of $\mathcal{T}_i$ if there exists a prefix τ such that $\sigma \subseteq_\Theta \tau$.

If no rule applies satisfying restriction $\mathcal{R}$, the algorithm is terminated.

Theorem 3.5 *The systematic tableau construction algorithm terminates.*

Proof. Assume to obtain a contradiction that this is not the case. Then the tableau $\cup_{i\in\omega}\mathcal{T}_i$ must be infinite. Thus it contains an infinite path Θ. By the

tableau conventions, all prefixed formulas along this path are distinct. Using Lemma 3.1, it follows that Θ must contain infinitely many different prefixes. Therefore the graph $(N^\Theta, <_\Theta)$ must be infinite. Since by Proposition 3.2 the graph is a finitely branching tree, it must contain an infinite path $\sigma_1 <_\Theta \sigma_2 <_\Theta \sigma_3 <_\Theta \cdots$. For each $i > 0$, let Θ_i be the initial segment of Θ up to, but not including, the formula containing the first occurrence of σ_{i+1}. Let Γ_i be the set $\Gamma_i = \{\phi \mid \sigma_i\phi \text{ occurs at } \Theta_i\}$. All Γ_i contain only formulas that are either subformulas of the root formula or negations of such formulas (Lemma 3.1). Since there are only finitely many such formulas, not all Γ_i can be distinct. In other words, there exists i, j with $i < j$ such that $\Gamma_i = \Gamma_j$. We will now prove that $\sigma_j \subseteq_{\Theta_j} \sigma_i$. Since i is less than j, the first occurrence of σ_i on Θ_j must come before the first occurrence of σ_j. Let ϕ be an arbitrary formula for which $\sigma_j\phi$ occurs on Θ_j, that is, $\phi \in \Gamma_j$. Since $\Gamma_i = \Gamma_j$, we have that $\sigma_i\phi$ occurs on Θ_i, and since Θ_i is an initial segment of Θ_j, we get that $\sigma_i\phi$ occurs on Θ_j. This proves that $\sigma_j \subseteq_{\Theta_j} \sigma_i$. Now consider the first formula containing an occurrence of σ_{j+1}. By definition, this is the first formula not on Θ_j, so it must be introduced by applying some rule to a formula occurrence at Θ_j. The prefix σ_{j+1} is generated by σ_j, so σ_{j+1} is introduced by applying one of the rules $(:)$, $(\neg:)$, $(\Diamond)$ or (E) to a formula $\sigma_j\psi$ at Θ_j. However, this is in contradiction with restriction $\mathcal{R}$ by which none of the rules $(:)$, $(\neg:)$, $(\Diamond)$ or (E) can be applied to the formula $\sigma_j\psi$ at Θ_j since $\sigma_j \subseteq_{\Theta_j} \sigma_i$. □

3.3 Soundness and completeness

Soundness of the tableau calculus in Figure 1 can be proved by showing that each rule preserves satisfiability [11]. The only rules in our calculus which are not already covered by Tzakova's system are (Id), (E) and $(\neg E)$. It is simple to prove that these rules preserve satisfiability in hybrid models. We now turn to the completeness proof. To prove completeness of the systematic tableau construction algorithm it is sufficient to prove that if a tableau with root $\sigma_0\phi_0$ has an open branch Θ then there exists a model $\mathcal{M}_\Theta$, an assignment g and a world w such that $\mathcal{M}_\Theta, g, w \models \phi_0$ holds. We will now describe how $\mathcal{M}_\Theta$ is constructed from an open tableau branch Θ. First a little extra machinery.

Definition 3.6 (Urfathers) Let Θ be a branch of a tableau and let σ be a prefix occurring on Θ. We define the *urfather* of σ with respect to Θ to be the earliest occurring prefix on Θ which σ is included in.[1] The urfather of σ with respect to Θ is denoted $u_\Theta(\sigma)$. Prefixes σ on Θ for which $u_\Theta(\sigma) = \sigma$ are called *urfathers* on Θ.

Lemma 3.7 *Let $\mathcal{T}$ be a tableau obtained from the tableau construction algorithm. $\mathcal{T}$ is closed under each of the rules $(\neg\neg)$, $(\wedge)$, $(\neg\wedge)$, $(\neg\Diamond)$, $(\neg E)$ and (Id) of Figure 1. Furthermore, $\mathcal{T}$ is closed under the rules $(:)$, $(\neg:)$, $(\Diamond)$ and*

[1] That is, the *urfather* of σ wrt. Θ is the unique prefix σ' satisfying: (i) $\sigma \subseteq_\Theta \sigma'$; (ii) there is no σ'' s.t. $\sigma' \subseteq_\Theta \sigma''$.

(E) whenever the premise is a formula occurrence of the form $\sigma\phi$ where σ is an urfather on the branch containing the occurrence.

Proof. Consider the sequence of tableaus constructed by the tableau algorithm leading to $\mathcal{T}$. Since the algorithm terminates, this must be a finite sequence $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_n$ where $\mathcal{T} = \mathcal{T}_n$. By definition, no rule applies to $\mathcal{T}_n$ that satisfies restriction $\mathcal{R}$. Since $\mathcal{R}$ only concerns the rules $(:)$, $(\neg:)$, $(\Diamond)$ and (E) we immediately get that the tableau is closed under all rules except possibly these. Now consider the rule $(\Diamond)$. Assume a branch Θ of $\mathcal{T}_n$ contains $\sigma\Diamond\phi$ where σ is an urfather on Θ. By definition of $\mathcal{T}_n$, no rule applies to $\sigma\Diamond\phi$ that satisfies $\mathcal{R}$. However, since σ is an urfather on Θ there is no prefix τ such that $\sigma \subseteq_\Theta \tau$. Thus the rule $(\Diamond)$ is not blocked by restriction $\mathcal{R}$. The only possible reason that the rule $(\Diamond)$ can not be applied to $\sigma\Diamond\phi$ on $\mathcal{T}_n$ is therefore that it has already been applied earlier in the tableau construction (cf. the tableau convention introduced in the beginning of Section 3). This proves closure under the rule $(\Diamond)$. Closure under the rules $(:)$, $(\neg:)$ and (E) are proved similarly. □

Lemma 3.8 *Let Θ be a branch of a tableau and let σ and τ be prefixes occurring on Θ. If there exists a nominal c such that both σc and τc occurs on Θ then σ is included in τ with respect to Θ.*

Proof. Let $\sigma\phi$ be a formula occurring on Θ. We have to prove that $\tau\phi$ occurs on Θ as well. This follows immediately from Lemma 3.7, since Θ contains all of $\sigma\phi$, σc and τc and is closed under the rule (Id). □

Given a tableau branch Θ, we define the model $\mathcal{M}_\Theta$ by

$$\begin{aligned}
\mathcal{M}_\Theta &= (W_\Theta, R_\Theta, V_\Theta), \text{ where} \\
W_\Theta &= \{u_\Theta(\sigma) \mid \sigma \text{ occurs on } \Theta\} \\
R_\Theta &= \{(\sigma, u_\Theta(\tau)) \in W_\Theta^2 \mid \sigma < \tau \text{ occurs on } \Theta\} \\
V_\Theta(\sigma, p) &= 1 \text{ iff } \sigma p \text{ occurs on } \Theta.
\end{aligned}$$

Furthermore, we define an assignment g_Θ for $\mathcal{M}_\Theta$ in the following way. We let $g_\Theta(c)$ be the prefix of the root formula of Θ if there is no prefix σ on Θ such that σc occurs on Θ (note that the prefix of the root formula is always an urfather). Otherwise we let $g_\Theta(c)$ be the urfather of the prefixes σ for which σc occurs on Θ. This defines $g_\Theta(c)$ uniquely, since if σ and σ' are prefixes such that both σc and $\sigma' c$ occurs on Θ, then it follows from Lemma 3.8 that σ will be included in σ' and they must thus have the same urfather. We are now ready to prove the completeness theorem. As mentioned above, it suffices to prove that if a tableau with root $\sigma_0\phi_0$ has an open branch Θ then there is a world w such that $\mathcal{M}_\Theta, g_\Theta, w \models \phi_0$. What we will prove is slightly stronger.

Theorem 3.9 (Completeness) *Let Θ be an open branch of a tableau constructed using the tableau algorithm of Section 3.2. For any prefixed formula $\sigma\phi$ on Θ where σ is an urfather on Θ we have $\mathcal{M}_\Theta, g_\Theta, \sigma \models \phi$.*

Proof. The proof is by induction on the structure of ϕ. First assume σp occurs on Θ where p is a propositional symbol and σ is an urfather. Then $V_\Theta(\sigma, p) = 1$ and thus $\mathcal{M}_\Theta, g_\Theta, \sigma \models p$ as needed. Now assume σc occurs on Θ where c is a nominal and σ is an urfather. Then $g_\Theta(c) = \sigma$, by definition of g_Θ, and thus $\mathcal{M}_\Theta, g_\Theta, \sigma \models c$, as needed. This covers the base case. We now turn to the induction step.

Consider the case where $\sigma\neg\neg\psi$ occurs on Θ and σ is an urfather. By closure under the rule $(\neg\neg)$ (Lemma 3.7) it follows that $\sigma\psi$ occurs on Θ as well. From the induction hypothesis we get $\mathcal{M}_\Theta, g_\Theta, \sigma \models \psi$, and thus $\mathcal{M}_\Theta, g_\Theta, \sigma \models \neg\neg\psi$ immediately follows. The other propositional cases $\sigma\psi \wedge \chi$ and $\sigma\neg(\psi \wedge \chi)$ are treated similarly.

Consider the case where $\sigma c : \psi$ occurs on Θ and σ is an urfather. By closure under the rule $(:)$ (Lemma 3.7), there exists a prefix σ' such that $\sigma' c$ and $\sigma'\psi$ also occurs on Θ. Let $\sigma'' = g_\Theta(c)$. Then σ'' is the urfather of σ' on Θ. From this it follows that $\sigma''\psi$ occurs on Θ as well. By induction hypothesis it follows that $\mathcal{M}_\Theta, g_\Theta, \sigma'' \models \psi$. Since $\sigma'' = g_\Theta(c)$ this proves $\mathcal{M}_\Theta, g_\Theta, \sigma \models c : \psi$, as needed. The case $\sigma\neg c : \psi$ is proved similarly.

Consider the case where $\sigma\Diamond\psi$ occurs on Θ and σ is an urfather. By closure under the rule $(\Diamond)$ (Lemma 3.7), there exists a prefix σ' such that both $\sigma'\psi$ and $\sigma < \sigma'$ occurs on Θ. Let $\sigma'' = u_\Theta(\sigma')$. The induction hypothesis gives $\mathcal{M}_\Theta, g_\Theta, \sigma'' \models \psi$. Since $\sigma < \sigma'$ occurs on Θ we have that R_Θ contains the pair $(\sigma, u_\Theta(\sigma')) = (\sigma, \sigma'')$. Thus we get $\mathcal{M}_\Theta, g_\Theta, \sigma \models \Diamond\psi$.

Consider the case where $\sigma\neg\Diamond\psi$ occurs on Θ and σ is an urfather. We have to prove $\mathcal{M}_\Theta, g_\Theta, \sigma \not\models \Diamond\psi$. If there is no prefix τ such that $\sigma R_\Theta \tau$ then this trivially holds. Otherwise, let τ be any prefix with $\sigma R_\Theta \tau$. We have to prove $\mathcal{M}_\Theta, g_\Theta, \tau \models \neg\psi$. By definition of R_Θ, τ is the urfather of a prefix τ' such that $\sigma < \tau'$ occurs on Θ. Since both $\sigma\neg\Diamond\psi$ and $\sigma < \tau'$ occurs on Θ, we get by closure under the rule $(\neg\Diamond)$ (Lemma 3.7) that $\tau'\neg\psi$ occurs on Θ as well. Since τ is the urfather of τ', the formula $\tau\neg\psi$ must also occur on Θ. By induction hypothesis we then have $\mathcal{M}_\Theta, g_\Theta, \tau \models \neg\psi$, as needed.

Consider the case where $\sigma E\psi$ occurs on Θ and σ is an urfather. By closure under the rule (E) (Lemma 3.7) there exists a prefix σ' such that $\sigma'\psi$ occurs on Θ. Let σ'' be the urfather of σ' on Θ. Then $\sigma''\psi$ also occurs on Θ and by induction hypothesis we get $\mathcal{M}_\Theta, g_\Theta, \sigma'' \models \psi$. This proves $\mathcal{M}_\Theta, g_\Theta, \sigma \models E\psi$.

Finally consider the case where $\sigma\neg E\psi$ occurs on Θ and σ is an urfather. We have to prove $\mathcal{M}_\Theta, g_\Theta, \sigma \models \neg E\psi$, that is, for all $\sigma' \in W_\Theta$, $\mathcal{M}_\Theta, g_\Theta, \sigma' \models \neg\psi$. To prove this, let an arbitrary element σ' in W_Θ be chosen. The element σ' is an urfather occurring on the branch Θ. By closure under the rule $(\neg E)$ (Lemma 3.7), $\sigma'\neg\psi$ occurs on Θ. Thus the induction hypothesis gives us $\mathcal{M}_\Theta, g_\Theta, \sigma' \models \neg\psi$ as needed.

□

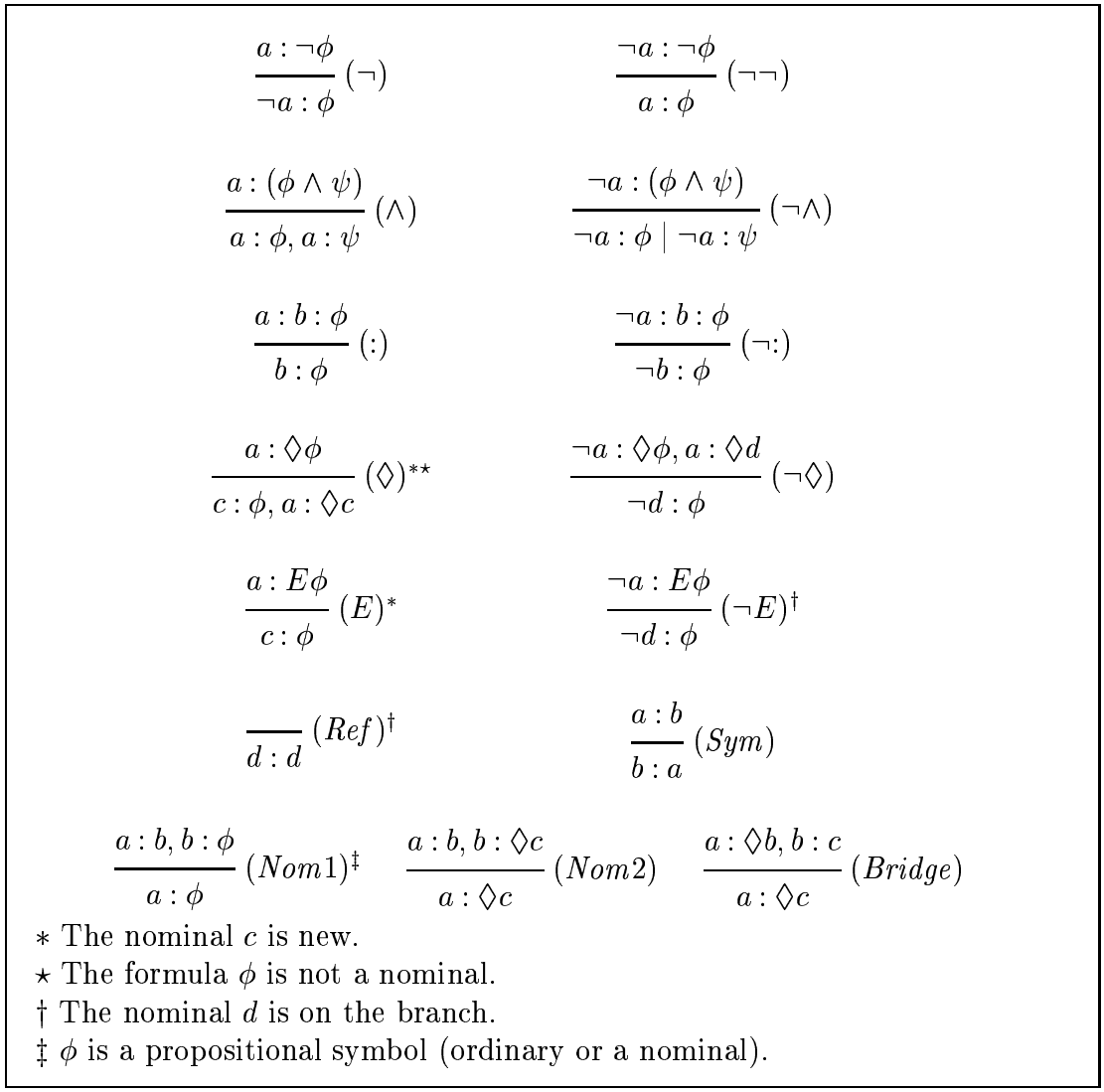


Fig. 2. Blackburn's tableau rules and rules for the universal modality

4 Blackburn's system extended with the universal modality

The tableau system considered in the present section is a slightly modified, and also extended, version of a system originally given in the paper [2] by Patrick Blackburn. The rules are given in Figure 2. The rules are identical to the rules given in [2] except that in his system the rules for the universal modality are not included, and moreover, in his system the rule $(Nom1)$ is not restricted to propositional symbols, and consequently, the rule $(Nom2)$ is omitted. It turns out that we do not need the more general version of $(Nom1)$ given in [2] and restricting it as we have done here simplifies later technical considerations. We have taken the connectives $\vee$ and $\Box$ to be defined, not primitive, so they do not need separate rules. All formulas in the rules are satisfaction statements. A *tableau* in the system is a well-founded tree in which

each node is a satisfaction statement and the edges represent applications of tableau rules in the usual way. When it is appropriate, we shall often blur the distinction between a formula and an occurrence of the formula in a tableau.

We shall make use of some important conventions about the rules of Figure 2. The rules $(\neg)$, $(\neg\neg)$, $(\wedge)$, $(\neg\wedge)$, $(:)$, $(\neg:)$, $(\Diamond)$, and (E) will be called *destructive* rules and the remaining rules will be called *non-destructive*. Note that a destructive rule has exactly one formula in the premise. The destructive rules $(\Diamond)$ and (E) will also be called *existential*. Thus, we have three categories of rules: Destructive rules which are not existential, existential destructive rules, and non-destructive rules. The rules of these three categories are applied in different ways.

- A destructive rule which is not existential is applied to a formula occurrence ϕ in a tableau by extending all branches through ϕ in accordance with the rule. After the application, it is recorded that the rule was applied to ϕ and the rule will not again be applied to ϕ.
- An existential destructive rule is applied to a formula occurrence ϕ on a branch Θ by extending Θ in accordance with the rule. After the application, it is recorded that the rule was applied to ϕ with respect to Θ and the rule will not again be applied to ϕ with respect to Θ or any extension of Θ.
- A non-destructive rule is applied to a set of formula occurrences (note that a non-destructive rule has zero, one, or two formulas in the premise) on a branch Θ by extending Θ in accordance with the rule. No information is recorded about applications of non-destructive rules.

Both destructive and non-destructive rules are subject to the exception that if a formula to be added to a branch is already present on the branch, then the addition of the formula is simply omitted. It follows that a formula cannot occur more than once at a branch.

Note that non-destructive rules are only applicable to formulas of the forms $a : p$, $a : c$, $a : \Diamond c$, $\neg a : \Diamond\phi$, and $\neg a : E\phi$ and conversely, destructive rules are only applicable to formulas not of these forms (in fact, exactly one destructive rule is applicable to any formula which is not of one of these forms). So the classification of rules as destructive and non-destructive corresponds to a classification of formulas. Also note that when applying a destructive rule which is not existential, it is applied to a formula occurrence with respect to all branches through it, but when applying an existential rule, the rule is applied with respect to one particular branch. This more fine-grained applicability of existential rules is a prerequisite for being able to incorporate so-called loop-checks, cf. Definition 4.11.

4.1 Some properties of the system

The tableau system satisfies the following important property, which is similar to the well-known subformula property of the standard propositional tableau

system.

Lemma 4.1 *(Quasi-subformula property) If a formula $a{:}\phi$ occurs in a tableau where ϕ is not a nominal and ϕ is not of the form $\Diamond b$, then ϕ is a positively occurring subformula of the root formula. If a formula $\neg a : \phi$ occurs in a tableau, then ϕ is a negatively occurring subformula of the root formula.*

Proof. A simultaneous induction where each rule is checked. □

Below we shall give some further results which shows some interesting features of the tableau system. First two definitions.

Definition 4.2 Let Θ be a branch of a tableau and let N^Θ be the set of nominals occurring in the formulas of Θ. Define a binary relation $\sim_\Theta$ on N^Θ by $a \sim_\Theta b$ if and only if the formula $a : b$ occurs at Θ. Let $\sim_\Theta^*$ be the reflexive, symmetric, and transitive closure of $\sim_\Theta$.

Definition 4.3 An occurrence of a nominal in a formula is *equational* if the occurrence is a formula (that is, if it is not part of a satisfaction operator).

For example, the occurrence of the nominal c in the formula $\phi \wedge c$ is equational but the occurrence of c in $\psi \wedge c : \chi$ is not. The justification for this terminology is that a nominal in the first-order correspondence language (and thereby also in the semantics) gives rise to an equality statement if and only if the nominal occurrence in question occurs equationally. The theorem below will be used later in the completeness theorem, Theorem 4.17.

Theorem 4.4 *Let $a : b$ be a formula occurrence on a branch Θ of a tableau. If the nominals a and b are different, then each of them has the property that it is identical to, or related by $\sim_\Theta$ to, a nominal with a positive and equational occurrence in the root formula.*

Proof. Check each rule. Lemma 4.1 is needed in a number of the cases. In the case with the rule ($\Diamond$), we make use of the restriction that the rule cannot be applied to formulas of the form $a : \Diamond\phi$ where ϕ is a nominal. □

Corollary 4.5 *Let Θ be a branch of a tableau. Any non-singleton equivalence class wrt. the equivalence relation $\sim_\Theta^*$ contains a nominal which occurs positive and equational in the root formula.*

Proof. Follows directly from Theorem 4.4. □

We think the corollary above is of independent interest. It says that non-trivial equational reasoning, that is, reasoning involving non-singleton equivalence classes, only takes place in connection with certain nominals in the root formula, namely those that occur positive and equational. Note that this implies that pure modal input to the tableau only gives rise to reasoning involving singleton equivalence classes.

Definition 4.6 A formula occurrence on a branch of a tableau is an *accessibility* formula occurrence if it is an occurrence of the formula $a : \Diamond c$ generated by the rule $(\Diamond)$.

Note that if the rule $(\Diamond)$ is applied to a formula occurrence $a:\Diamond\Diamond b$, resulting in the branch being extended with $a : \Diamond c$ and $c : \Diamond b$, then the occurrence of $a:\Diamond c$ is an accessibility formula occurrence, but the occurrence of $c:\Diamond b$ is not. The theorem below will be used later in the completeness theorem, Theorem 4.17.

Theorem 4.7 *Let $a:\Diamond b$ be a formula occurrence on a branch Θ of a tableau. Either there is a positively occurring subformula $\Diamond b'$ of the root formula such that $b \sim^*_\Theta b'$ or there is an accessibility formula occurrence $a' : \Diamond b'$ at Θ such that $a \sim^*_\Theta a'$ and $b \sim^*_\Theta b'$.*

Proof. Check each rule. Lemma 4.1 is needed in some of the cases. □

The only way new nominals can be introduced to a tableau is by using one of the rules $(\Diamond)$ or (E) which we called existential rules. This motivates the following definition.

Definition 4.8 Let Θ be a branch of a tableau. If a new nominal c is generated by applying an existential rule to a satisfaction statement $a : \phi$, then we write $a <_\Theta c$.

The definition above gives us a binary relation $<_\Theta$ on the set N^Θ.

Proposition 4.9 *Let Θ be a branch of a tableau. The graph $(N^\Theta, <_\Theta)$ is the disjoint union of a finite set of finitely branching trees.*

Proof. That the relation is the disjoint union of a set of trees follows from the observation that if $a <_\Theta c$, then the nominal c is new. That the set of trees is finite follows the observation that for any new nominal c there is a nominal a such that $a <_\Theta c$, thus, the nominal c cannot be the root of a tree. The following argument shows that the trees are finitely branching. Assume conversely that there exists an infinite sequence $a <_\Theta c_1$, $a <_\Theta c_2, \ldots$ of edges. For each i, the edge $a <_\Theta c_i$ is generated by applying an existential rule to some formula occurrence χ_i. Consider the sequence χ_1, $\chi_2, \ldots$ of formula occurrences. The existential rules are destructive, so the formula occurrences in this sequence are distinct, and moreover, a formula cannot occur more than once at a branch. It follows that the formula occurrences in the sequence χ_1, $\chi_2, \ldots$ are occurrences of infinitely many different formulas. Now, if the edge $a <_\Theta c_i$ is generated by applying the existential rule $(\Diamond)$ to χ_i, then χ_i is of the form $a : \Diamond\phi_i$ where ϕ_i is not a nominal, and hence, $\Diamond\phi_i$ is a subformula of the root formula by Lemma 4.1, and if $a <_\Theta c_i$ is generated by applying the other existential rule (E) to χ_i, then χ_i is of the form $a : E\phi_i$, and hence, $E\phi_i$ is a subformula of the root formula, again by Lemma 4.1. But there are only finitely many subformulas of the root formula, which contradicts that infinitely many different formulas occur in the the sequence χ_1, χ_2, ... □

Note that in the above results we have not made any assumptions on which rules are applied at the branch Θ, but if we assume that Θ is closed under the rules (*Ref*), (*Sym*), and (*Nom*1), then $\sim^*_\Theta$ coincides with $\sim_\Theta$.

4.2 Systematic tableau construction

Before giving the systematic tableau construction algorithm, we need a definition.

Definition 4.10 Let b and a be nominals occurring at a branch Θ of a tableau. The nominal a is *included* in the nominal b with respect to Θ if for any subformula ϕ of the root formula, if the formula $a : \phi$ occurs on Θ, then $b : \phi$ also occurs on Θ, and similarly, if $\neg a : \phi$ occurs on Θ, then $\neg b : \phi$ also occurs on Θ. If a is included in b with respect to Θ, and the first occurrence of b on Θ is before the first occurrence of a, then we write $a \subseteq_\Theta b$.

We are now ready to give the systematic tableau construction algorithm.

Definition 4.11 Let $a{:}\phi$ be the formula whose validity we have to decide. We define by induction a sequence $\mathcal{T}_0$, $\mathcal{T}_1$, $\mathcal{T}_2$, ... of finite tableaus, each of which is embedded in all its successors. Let $\mathcal{T}_0$ be the finite tableau constituted by the single unmarked formula $\neg a : \phi$. Assume that the finite tableau $\mathcal{T}_n$ is defined. If possible, apply an arbitrary rule with the following restrictions:

(i) The rule ($\Diamond$) is not applied to a formula occurrence $a : \Diamond\phi$ at a branch Θ if there exists a nominal b such that $a \subseteq_\Theta b$.

(ii) The rule (E) is not applied to a formula occurrence $a : E\phi$ at a branch Θ if there exists a nominal b such that $a \subseteq_\Theta b$.

Let $\mathcal{T}_{n+1}$ be the resulting tableau.

The conditions on applications of rules are so-called loop-check conditions. The intuition behind loop-checks is that an existential rule is not applied in a world if the information in that world can be found already in an ancestor world. Hence, the generation of a new world by the existential rule is blocked.

We shall now prove that the algorithm always terminates in the sense that there always exists an n such that $\mathcal{T}_n = \mathcal{T}_{n+1}$.

Theorem 4.12 *The systematic tableau construction algorithm terminates.*

Proof. Assume conversely that the algorithm does not terminate. Then the resulting tableau is infinite, and hence, has an infinite branch Θ. The graph $(N^\Theta, <_\Theta)$ is the disjoint union of a finite set of finitely branching trees cf. Proposition 4.9, so it has an infinite branch $a_1 <_\Theta a_2 <_\Theta a_3, \ldots$ (otherwise $\mathcal{N}^\Theta$ would be finite, and hence, by Lemma 4.1 there would only be finitely many formulas occurring at the branch Θ, contradicting that it is infinite). Now, for each i, let Θ_i be the initial segment of Θ up to, but not including, the first formula containing an occurrence of the nominal a_{i+1}. Thus, an existential rule was applied to a formula occurrence at the branch Θ_i resulting

in the generation of a_{i+1}. Let Γ_i be the set of formulas which contains any subformula ϕ of the root formula such that $a_i : \phi$ occurs at the branch Θ_i, and similarly, let Δ_i be the set of formulas which contains any subformula ϕ of the root formula such that $\neg a_i : \phi$ occurs at the branch Θ_i. Since there are only finitely many sets of subformulas of the root formula, there exists j and k such that $j < k$ and $\Gamma_j = \Gamma_k$ as well as $\Delta_j = \Delta_k$. Clearly, the first occurrence of a_j on Θ is before the first occurrence of a_k. Moreover, for any subformula ϕ of the root formula, if $a_k : \phi$ occurs on Θ_k, then $\phi \in \Gamma_k$, and hence, $\phi \in \Gamma_j$, but then $a_j : \phi$ occurs on Θ_j which is an initial segment of Θ_k. A similar argument shows that if $\neg a_k : \phi$ occurs on Θ_k, then $\neg a_j : \phi$ also occurs on Θ_k. Hence, a_k is included in a_j with respect to Θ_k. We conclude that $a_k \subseteq_{\Theta_k} a_j$. But this contradicts that an existential rule was applied to a formula occurrence at the branch Θ_k resulting in the addition of the first formula containing an occurrence of the nominal a_{k+1}. Thus, the algorithm terminates. □

We have thus given a systematic tableau construction algorithm which gradually builds up a tableau and which terminates with a tableau having the property that no rules are applicable to it except for applications of existential rules blocked by the loop-check conditions.

4.3 Soundness and completeness

Soundness is straightforwardly obtained by extending the soundness proof of [2] with the universal modality. To prove completeness, we prove a model existence theorem. Throughout this subsection, we shall assume that Θ is a given branch of a tableau generated by the systematic tableau construction algorithm. Where no confusion can occur, we shall often omit reference to the branch Θ. First some machinery.

Definition 4.13 Let W be the subset of N^Θ containing any nominal a having the property that there is no nominal b such that $a \subseteq_\Theta b$. Let $\approx$ be the restriction of $\sim_\Theta$ to W.

Note that W contains all nominals of the root formula since the root formula is the first formula of the branch Θ. Observe that Θ is closed under the rules (*Ref*), (*Sym*), and (*Nom*1), so the relation $\sim_\Theta$ and hence also the relation $\approx$ are equivalence relations. Given a nominal a in W, we let $[a]_\approx$ denote the equivalence class of a with respect to $\approx$ and we let $W/\approx$ denote the set of equivalence classes.

Definition 4.14 Let R be the binary relation on W defined by aRc if and only if there exists nominals $a' \approx a$ and $c' \approx c$, satisfying one of the following three conditions.

(i) The formula $a' : \Diamond c'$ occurs at Θ is an accessibility formula occurrence.

(ii) There exists a nominal d in N^Θ such that the formula $a' : \Diamond d$ occurs at Θ as an accessibility formula occurrence and $d \subseteq_\Theta c'$.

(iii) The formula $a' : \Diamond c'$ occurs at Θ and a' or c' occurs in the root formula.

Note that the nominal d referred to in the second item in the definition is not an element of W. It is trivial that the relation R is compatible with $\approx$. We let $\overline{R}$ be the binary relation on $W/\approx$ defined by $[a]_\approx \overline{R} [c]_\approx$ if and only if aRc.

Definition 4.15 Let V be the function that to each pair consisting of an element of W and an ordinary propositional symbol assigns an element of $\{0, 1\}$ such that $V(a, p) = 1$ if $a : p$ occurs at Θ and $V(a, p) = 0$ otherwise.

It follows from Θ being closed under the rule $(Nom1)$ that V is compatible with $\approx$, so we let $\overline{V}$ be defined by $\overline{V}([a]_\approx, p) = V(a, p)$. We are now ready to define a model.

Definition 4.16 Let $\mathcal{M}$ be the model $(W/\approx, \overline{R}, \overline{V})$ and let the assignment g for $\mathcal{M}$ be defined by $g(a) = [a]_\approx$.

The model above is in some respects similar to the model defined in [2]. One crucial difference, however, is that the model above necessarily is finite.

Theorem 4.17 *Assume that the branch Θ is open, that is, if some formula $b : \chi$ occurs at Θ, then the formula $\neg b : \chi$ does not. For any formula $a : \phi$ which only contains nominals from W, the following two statements hold.*

- *If $a : \phi$ occurs at Θ, then it is the case that $\mathcal{M}, g, [a]_\approx \models \phi$.*
- *If $\neg a : \phi$ occurs at Θ, then it is not the case that $\mathcal{M}, g, [a]_\approx \models \phi$.*

Proof. Induction in the structure of ϕ. We only cover the most interesting case, namely where ϕ is of the form $\Diamond\psi$.

Assume that $a : \Diamond\psi$ occurs at Θ. We then have to prove that $\mathcal{M}, g, [a]_\approx \models \Diamond\psi$, that is, for some equivalence class $[c]_\approx$ such that $[a]_\approx \overline{R} [c]_\approx$, it is the case that $\mathcal{M}, g, [c]_\approx \models \psi$. We have two cases, according to whether the formula ψ is a nominal or not. We first consider the case where ψ is a nominal, say b. So we just have to prove that $[a]_\approx \overline{R} [b]_\approx$. By Theorem 4.7, either there is a nominal b' of the root formula such that $b' \sim_\Theta b$ or there is an accessibility formula occurrence $a' : \Diamond b'$ at Θ such that $a' \sim_\Theta a$ and $b' \sim_\Theta b$. If the first is the case, then also $a : \Diamond b'$ occurs at Θ, and $b' \in W$, so $[a]_\approx \overline{R} [b']_\approx$, and trivially, $[b']_\approx = [b]_\approx$. If the second is the case, and moreover, $a' = a$ and $b' = b$, then clearly $[a]_\approx \overline{R} [b]_\approx$. If $a' = a$ and $b' \neq b$, then by Theorem 4.4, there is a nominal c of the root formula such that $b' \sim_\Theta c$. But then also $a : \Diamond c$ occurs at Θ, and $c \in W$, so $[c]_\approx \overline{R} [c]_\approx$, and trivially, $[c]_\approx = [b]_\approx$. If $a' \neq a$ and $b' = b$, then by Theorem 4.4, there is a nominal c of the root formula such that $a' \sim_\Theta c$. But then also $c : \Diamond b$ occurs at Θ, and $c \in W$, so $[a]_\approx \overline{R} [b]_\approx$, and trivially, $[c]_\approx = [a]_\approx$. If $a' \neq a$ and $b' \neq b$, then by Theorem 4.4, there are nominals c and d of the root formula such that $a' \sim_\Theta c$ and $b' \sim_\Theta d$. But then also $c : \Diamond d$ occurs at Θ, and $c, d \in W$, so $[c]_\approx \overline{R} [d]_\approx$, and trivially, $[c]_\approx = [a]_\approx$ and $[d]_\approx = [b]_\approx$. We now consider the case where ψ is a not nominal. By the rule $(\Diamond)$ also some

formulas $a : \Diamond c$ and $c : \psi$ occur at Θ where the nominal c is new (note that $a \in W$, so the application of the rule is not blocked by a loop-check condition). If $c \in W$, then clearly $[a]_{\approx}\overline{R}[c]_{\approx}$, and by induction $\mathcal{M}, g, [c]_{\approx} \models \psi$. If $c \notin W$, then by definition of W there exists a nominal d such that $c \subseteq_{\Theta} d$. Without loss of generality we assume that there does not exist a nominal e such that $d \subseteq_{\Theta} e$. But this implies that $d \in W$. Moreover, by Lemma 4.1, the formula ψ is a subformula of the root formula, so $d : \psi$ occurs at Θ. By induction, $\mathcal{M}, g, [d]_{\approx} \models \psi$, and clearly, $[a]_{\approx}\overline{R}[d]_{\approx}$.

Assume that $\neg a : \Diamond\psi$ occurs at Θ. We then have to prove that $\mathcal{M}, g, [a]_{\approx} \models \Diamond\psi$ does not hold, that is, for any equivalence class $[c]_{\approx}$ such that $[a]_{\approx}\overline{R}[c]_{\approx}$, it is not the case that $\mathcal{M}, g, [c]_{\approx} \models \psi$. From $[a]_{\approx}\overline{R}[c]_{\approx}$ it follows that there exists nominals $a' \approx a$ and $c' \approx c$, satisfying one of the three conditions in the definition of the relation R. In the first and third condition in this definition, the formula $a' : \Diamond c'$ occurs at Θ. Then $a : \Diamond c$ also occurs at Θ, and by the rule $(\neg\Diamond)$ also $\neg c : \psi$. By induction we conclude that $\mathcal{M}, g, [c]_{\approx} \models \psi$ does not hold. In the second condition in the definition, there exists a nominal d in N^{Θ} such that the formula $a' : \Diamond d$ occurs at Θ and $d \subseteq_{\Theta} c'$. Then $a : \Diamond d$ also occurs at Θ, and by the rule $(\neg\Diamond)$ also $\neg d : \psi$. But by Lemma 4.1, the formula ψ is a subformula of the root formula, and $d \subseteq_{\Theta} c'$, so $\neg c' : \psi$ occurs at Θ. By induction we conclude that $\mathcal{M}, g, [c']_{\approx} \models \psi$ does not hold and trivially, $[c']_{\approx} = [c]_{\approx}$. □

5 Related work

In ordinary modal logic, loop-checks are used in connection with standard Fitting-style prefixed tableau systems for transitive logics such as **K4**, see [6] and [10]. Early applications of loop-checks can be found in [9] and [5]. Now, a simple prefixed tableau system can be formulated for the modal logic **K** such that a systematic tableau construction always terminates. The systematic tableau construction algorithm for **K** does not involve loop-checks. However, a systematic tableau construction may not terminate if the tableau system for **K** is extended with the standard prefixed tableau rule

$$\frac{s : \Box\phi, s < t}{t : \Box\phi}$$

for transitivity (the notation should be self-explanatory), whereby a tableau system for **K4** is obtained. Intuitively, the problem is that the rule allows information to be moved forward from a world to any accessible world. The standard way to fix this problem is to incorporate loop-check conditions on the applications of existential rules. The intuitive reason why this technique works in the context of hybrid logic too, is that the problem here also is that information can be moved between worlds, namely in connection with applications of the rule (Id) in the Tzakova-style system and similarly, in connection with the rules $(Nom1)$ and $(Nom2)$ in the Blackburn-style system. Intuitively, these rules allow information to be moved between worlds that are

identical.

Nominals are often used in description logics, and certain tableau-based decision procedures for such logics also make use of loop-checks. This is for example the case with the decision procedure given in [7] which is based on a prefixed tableau system in line with our Tzakova-style system. [2] The logic given in that paper, and other similar logics, do not involve satisfaction operators or the universal modality, but it is well-known that if a description logic has transitive roles together with role hierarchies, which is the case with the logic in [7], then general concept inclusion axioms can be internalised into concepts, as described in [8], pages 164 and 165. This technique can also be used to define an "approximation" of the universal modality: Given roles $R_1, \ldots, R_n$ occurring in a formula ϕ and a new role U, a set of role axioms

$$\{\mathsf{Trans}(U), R_1 \sqsubseteq U, ..., R_n \sqsubseteq U\}$$

is defined ensuring that the relation for the role U is transitive and contains the relations for all the other roles. It follows that ϕ is satisfiable wrt. arbitrary models if and only if the formula ϕ' obtained by replacing any universal modality $E\psi$ in ϕ by $\psi \wedge \forall U.\psi$ is satisfiable wrt. models satisfying the axioms. In the terminology of modal logic, this is the case since the axioms ensures that $\psi \wedge \forall U.\psi$ is true at a world w if and only if ψ is true at a set of worlds containing the submodel generated by w. In case nominals are involved, further axioms have to be added such that $\psi \wedge \forall U.\psi$ is true at a world w if and only if ψ is true at a set of worlds containing the submodel generated by the set of worlds consisting of w together with the denotations of all nominals in ψ. In this sense, the universal modality can be approximated if further machinery is present, namely axioms involving transitive roles and role hierarchies.

However, we think that the universal modality and satisfaction operators are so important and widely used that it justifies independent and direct tableau-based decision procedures, as given in the present paper. Also, it seems unnecessarily complicated to obtain a decision procedure encompassing the universal modality (which is first-order definable) by a reduction to a decision procedure involving axioms for a new role (which amounts to imposing a second-order condition on models, namely the condition that there exists a relation satisfying the axioms).

Acknowledgements: Thanks to Patrick Blackburn for comments on the work presented here, in particular for suggesting to incorporate the universal modality. Also thanks to Jørgen Villadsen for comments. The authors are partially supported by the Danish Natural Science Research Council in connection with the HyLoMOL project.

[2] We thank one of the anonymous referees for pointing this out.

References

[1] Areces, C., P. Blackburn and M. Marx, *Hybrid logics: Characterization, interpolation and complexity*, Journal of Symbolic Logic **66** (2001), pp. 977–1010.

[2] Blackburn, P., *Internalizing labelled deduction*, Journal of Logic and Computation **10** (2000), pp. 137–168.

[3] Blackburn, P., M. de Rijke and Y. Venema, "Modal Logic," Cambridge Tracts in Theoretical Computer Science **53**, Cambridge University Press, 2001.

[4] D'Agostino, M., D. Gabbay, R. Hähnle and J. Posegga, editors, "Handbook of Tableau Methods," Springer, 1999.

[5] Fitting, M., "Proof Methods for Modal and Intuitionistic Logic," Reidel, 1983.

[6] Goré, R., *Chapter 6: Tableau methods for modal and temporal logics*, in: *Handbook of Tableau Methods*, Kluwer Academic Publishers, 1999 pp. 297–396.

[7] Horrocks, I. and U. Sattler, *Ontology reasoning in the $\mathcal{SHOQ}(D)$ description logic*, in: *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, 2001, pp. 199–204.

[8] Horrocks, I., U. Sattler and S. Tobies, *Practical reasoning for expressive description logics*, in: *Proceedings of LPAR'99*, Lecture Notes in Artificial Intelligence **1705** (1999), pp. 161–180.

[9] Hughes, G. and M. Cresswell, "An Introduction to Modal Logic," Methuen, 1968.

[10] Massacci, F., *Single step tableaux for modal logics*, Journal of Automated Reasoning **24** (2000), pp. 319–364.

[11] Tzakova, M., *Tableaux calculi for hybrid logics*, in: N. V. Murray, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX 1999*, Lecture Notes in Artificial Intelligence **1617** (1999), pp. 278–292.

Hybrid Logic with operations on nominals

Evangelos Tzanis

University College of London
Department of Computer Science
e.tzanis@cs.ucl.ac.uk

Abstract. The *basic hybrid logic* $\mathcal{H}(@)$ ([5]) is expanded with function symbols on nominals. We call this hybrid language $\mathcal{HLON}$ (**H**ybrid **L**ogic with **O**perators on **N**ominals), henceforth. The new language appears to be strong enough to express for example, any equationally definable class of algebras or the property of 2-colorability of a graph (not definable in the first order logic). The paper contains a number of worked examples, discussing the expressive power of $\mathcal{HLON}$.
Keywords: Hybrid Logics, skolem functions, Henkin Quantifiers

1 Introduction

Hybrid logic is the result of extending the basic modal language with a second sort of atomic propositions called nominals, and with satisfaction operators. Precisely, the nominals (denoted by $i, j, ...$) behave similarly to ordinary proposition letters, except that nominals are true at a unique world. In other words, a nominal names a state by being true there and nowhere else. The language obtained by adding nominals to the basic modal language, is called the *minimal hybrid logic* $\mathcal{H}$. Satisfaction operators allow one to express that a formula holds at the world named by a nominal. A formula of the form $@_i\phi$ expresses the fact that ϕ holds at the world named by the nominal i. The extension of the basic modal language with nominals and satisfaction operators is called the *basic hybrid language* $\mathcal{H}(@)$.

1.1 Motivation

In this paper we introduce and study extensions of the basic hybrid logic adding function symbols on nominals. There are many reasons for being interested in these hybrid languages:

Algebraic-Structure: The motivation of this point comes from [5]: *you can name states but you can not give them structure*. In this paper we consider an application of hybrid logics to relational structures on algebras, thus a set with a relation and an algebraic structure. In general, modal logics which can explicitly handle both relations and functions have received little attention. We should mention that in [17] there is a complete list of papers which study Kripke frames in which the universe of possible worlds has a specific algebraic structure, besides the traditional relational component.

Frame Definability: Following the research lines of hybrid languages we accommodate the notion of pure formulas to $\mathcal{HLON}$. Pure formula is a formula containing only nominals (rather than propositional variables), defining frame classes that are always first-order definable (see, e.g., [[16], Proposition 3.1]). It is known from [11] that axiomatizations with pure formulae give rise to *natural* completeness proofs, in [8] a completeness proof for languages containing nominals and @-operator, was given. Although many first-order conditions that are modally definable are definable by pure formulae, for instance reflexivity to name one, this is not true in general. The right directedness property: $\forall vu \exists t(vRt \wedge uRt)$ is not definable by any orthodox modal formula and also is not equivalent to any pure formulae [8]. In this paper we show how frame properties, not definable by pure formulas of $\mathcal{H}(@)$, are definable by $\mathcal{HLON}-$ pure formulas. To be precise, we show that pure formulas of $\mathcal{HLON}$ can express any frame property definable by a first order formula.

Expressivity: We show that $\mathcal{HLON}$ is strong enough to express, under a modified definition of frame validity, the property of 2-colorability of graph, not definable in first order logic [14].

The paper is as follows. In section 2 we name a version of the basic hybrid logic $\mathcal{H}(@)$ and we call it $\mathcal{HLON}$ henceforth. In section 3, we name a version of the strong hybrid language $\mathcal{H}(@, \forall, \exists)$ and we call it $\mathcal{HLOV}(@, \forall, \exists)$ (**H**ybrid **L**ogic with **O**perators only for **V**ariables). In the same chapter we show how to move from the universal fragment of $\mathcal{HLOV}(@, \forall, \exists)$ to $\mathcal{HLON}$. In section 4 we present a number of examples discussing the expressive power of $\mathcal{HLON}$. In section 5 we show that $\mathcal{HLON}$ is strong enough to express any frame property definable by a first order formula.

2 The hybrid logic: $\mathcal{HLON}$

We are ready now to start our formal work on Hybrid Logics. First some basic definitions:

Definition 1. *An algebraic similarity type is an ordered pair $\sigma = (A, \rho)$ where A is a non empty set and ρ is a function $A \to N$. Elements of A are called functions symbols; the function ρ assigns to each operator $f \in A$ a finite arity (rank), indicating the number of arguments that f can be applied to. Function symbols of rank zero are called constants.*

Let **ANOM** be a countable set of atomic nominals: **ANOM**$= \{i, i_1, \cdots\}$. Let σ be an algebraic similarity type.

Definition 2. *An algebraic language of nominals (σ,* **ANOM***) is built up using an algebraic similarity type $\sigma = (A, \rho)$ and a set of atomic nominals* **ANOM**. *The* **NOM**$=$*TERM(σ,* **ANOM***) of nominals formulas over σ and* **ANOM** *is given by the rule:*

$$\alpha := i \mid f(\alpha_1, \cdots, \alpha_{\rho(f)})$$

where $\alpha \in$ **NOM** *and $f \in \sigma$.*

Note that **CNOM** = NOM \ ANOM, is the set of non atomic nominals; we call them complex nominals henceforth. Let **PROP**= $\{p_1, p_2, \cdots\}$ be a countable set of propositional variables. Let Φ = **PROP** $\cup$ **NOM**.

Definition 3. $\mathcal{HLON}$ **FRAMES and MODELS**.

A model $\mathcal{M}$ for $\mathcal{HLON}$ is a $\mathcal{M} = (W, R_\Diamond, (F^f)_{f\in\sigma}, V)$ such that, $W \neq 0$, a set of possible worlds, $R_\Diamond$ is a binary relation over W, for each $n \geq 0$ and for all $f \in \sigma$, if f is an $n-$ary function symbol then F^f is an $n-$ary operator on W. Note that $F^f : W^n \to W$ and

- *V: $\Phi \to$ Pow(W) and V is such that for all nominals $\alpha \in$* **NOM***, V(α) is singleton of W and:*
  - *If $\alpha \in$* **NOM** *then: V(α)= $\{w\}$;*
  - *If $\alpha \in$* **CNOM** *then: V($f(\alpha_1, ..., \alpha_n)$) = $\{F^f(V(\alpha_1), ..., V(\alpha_n))\}$.*

We call the elements of $\mathcal{M}$ states or worlds, $R_\Diamond$ accessibility relations, F^f operators on W and V the valuation. We call the triple $(W, R_\Diamond, (F^f)_{f\in\sigma})$ the *functional-relational frame* underlying the model.

Definition 4. *The Hybrid Language $\mathcal{HLON}$ is a language that is used for describing models and frames. Its formulas are given by the following recursive definition.*

$$\varphi := \alpha \mid p \mid \neg\varphi \mid \varphi \to \psi \mid \Diamond\varphi \mid @_\alpha\varphi$$

where $\alpha \in$ **NOM** *and $p \in$* **PROP**.

Note that all types of atomic symbols (propositional variables and nominals) are formulas. Furthermore a formula is **pure** if it is contains no propositional variables and nominal free if it contains no nominals.

Definition 5. Semantics for $\mathcal{HLON}$

Let a model $\mathcal{M} = (W, R_\Diamond, (F^f)_{f\in\sigma}, V)$ for $\mathcal{HLON}$ (we give only the new case)

$$\mathcal{M}, w \models \alpha \quad \textit{iff} \quad V(\alpha) = \{w\}$$

$(\mathcal{M}, w)$ is called a pointed model. This is a model with a designated world, called its point, which is taken to be the actual world.

For simplicity we limit the language to one binary relation $R_\Diamond$ and to one function F (unary or binary depends on the example that we will use). In other words our hybrid model is: $\mathcal{M} = (W, R_\Diamond, F^f, V)$ or for brevity $\mathcal{M} = (W, R, F, V)$.

2.1 $\mathcal{HLON}$ Frame definability

In this section, we revisit the notion of *functional relational frame that is central to this paper*. Formally, on page 3 we defined the triple $\mathcal{F} = (W, R_\Diamond, (F^f)_{f\in\sigma})$ to be a *functional relational frame*. Using this notion of a relational functional frame we can define classes of ordinary relational Kripke frames as follows: Let Φ be a set of $\mathcal{HLON}$ formulas. Define $\mathcal{F}r(\Phi)$ the class of Kripke frames validating Φ as the class of consisting of all $(W, R_\Diamond)$ such that there is a functional relational frame $(W, R_\Diamond, (F^f)_{f\in\sigma})$ extending it, and $(W, R_\Diamond, (F^f)_{f\in\sigma}) \models \phi$ for all $\phi \in \Phi$. That is, we let Φ define the class of all Kripke frames obtained as **reducts** of functional relational frames validating Φ.

Frames like $\mathcal{F} = (W, R_\Diamond, (F^f)_{f\in\sigma})$ are useful for at least two reasons. First, there may be application driven motivations to exclude certain valuations [6]. For example, in this paper we consider some linguistic and graph theoretical examples, showing that the use of function symbols with fixed valuation increases the expressive power of the basic hybrid logic. A second reason and perhaps the most important, is that frames like $\mathcal{F}$ support a notion of validity that definitely is weaker than the frame based one, but without loosing too many information that make models so easy to work with.

However we can meet such frames in other logical formalisms. We list some of them here:

- General frames ([6], page 28) are frames with a collection of valuations A satisfying modally natural closure conditions. That is, a general frame is a structure of the form $(\mathcal{F}, A)$ where $\mathcal{F}$ is a relational frame and A stands for admissible valuations or algebra ([6], page 28). The motivation in [6] was the following: there is a universal completeness result for general frame validity, something that Kripke frame semantics lacks.
- Distance functions play an important role in the context of modal distance logics, where distance functions map worlds of a domain W to numbers. A consequence of using functions to our logical formalism is that we need to modify many Kripkean notions, like the notion of a frame. Precise definitions regarding distance frames and modal distance logics could be found in [20] (pages 18-22)

In this paper we relate the use of function symbols with the node creating rule of [7]. Actually this node creating rule amounts to using nominals as skolem functions. Roughly speaking, in [7] nominals were used to eliminate existential quantifiers of formulas not definable by any pure nominal formula. This approach might be better understood in terms of an example.

Church-Rosser is a frame condition, defined by the following first order formula $\forall x, y, z \exists t(xRy \wedge xRz \rightarrow (yRt \wedge zRt))$, but not definable by a pure formula (page 53 in [13], proposition 4.2). Therefore, one might ask wether we can obtain complete axiomatizations for frame classes definable by a set of of pure formulas together with the Church-Rosser formula. In [7] the following inference rule was used:

Whenever $@_i \diamond j$ *and* $@_i \diamond k$ *occur on a branch we are allowed to create new nominal* l *and add* $@_j \diamond l$ *and* $@_k \diamond l$ *to the branch*

That is, l is a new nominal, eliminating the existential quantifier in $\forall x, y, z \exists t(xRy \wedge xRz \rightarrow (yRt \wedge zRt))$. As was proved in [7], the above rule preserves validity and is complete for the class of confluent frames. In section 4.3, using the notion of functional relational frames we show how the nominal creating rule can be accommodated to the level of frames. The notion of skolemization is important there.

Definition 6. *A formula* ϕ *is valid at a state* w *in a functional relational frame* $\mathcal{F} = (W, R_\Diamond, (F^f)_{f \in \sigma})$*, if* ϕ *is true at* w *in every model* $(\mathcal{F}, V)$ *on* $\mathcal{F}$ *and* ϕ *is valid on a functional relational frame* $\mathcal{F}$ *if* ϕ *is true at every state in every model* $(\mathcal{F}, V)$ *on* $\mathcal{F}$*.*

We use the notation $\mathcal{HLON}_F$ for the language based on the frame $\mathcal{F} = (W, R_\Diamond, F)$, where $F = (F^f)_{f \in \sigma}$ indicates the set of operators in $\mathcal{F}$. It should be clear that the truth or falsity of a formula at a point depends only on propositional variables (denoted as $p_0^{\mathcal{M}}, ...$), nominals (denoted as $i_0^{\mathcal{M}}, ...$) and operators in it. Thus, given a sublanguage $\mathcal{HLON}_{F'} \subset \mathcal{HLON}_F$ with $F^{'} \subset F$ and a frame $\mathcal{F} = (W, R, F)$ for $\mathcal{HLON}_F$, we may define the **frame reduct** of $\mathcal{F}$ as $\mathcal{F} \restriction_{F'} = (W, R, F^{'})$.

We then have for every formula ϕ of $\mathcal{HLON}_{L'}$ and every model $\mathcal{M}$ based on $\mathcal{F}$:

$$(\mathcal{F}, p_0^{\mathcal{M}}, ..., i_0^{\mathcal{M}}, ...), w \models \phi \text{ if and only if } (\mathcal{F} \restriction_{F'}, p_0^{\mathcal{M}}, ..., i_0^{\mathcal{M}}, ...), w \models \phi$$

The observation does not depend on the presence of nominals.

3 $\mathcal{HLOV}(@, \forall, \exists)$

In this section, we name a version of the strong hybrid language $\mathcal{H}(@, \forall, \exists)$ and we call it $\mathcal{HLOV}(@, \forall, \exists)$ (**H**ybrid **L**ogic with **O**perators only for **V**ariables). We set up its language and semantics. Please treat this section as a basis for the next sections.

3.1 $\mathcal{HLOV}(@, \forall, \exists)$

Let **SVAR** be a countable set o state variables **SVAR**$= \{x_1, x_2...\}$ and let **PROP**$= \{p_1, p_2...\}$ be a countable set of propositional letters. We assume that these sets are pairwise disjoint. For simplicity, let **f** be a binary function (over state variables, and with fixed interpretation) and $R_\Diamond$ be an accessibility relation symbol.

Definition 7. *The terms* s *and formulas* ϕ *of the hybrid Language* $\mathcal{HLOV}(@, \forall, \exists)$ *are inductively defined as follows:*

$$s := x \mid \mathbf{f}(s_1, s_2)$$

$$\varphi := \top \mid s \mid p \mid \neg\varphi \mid \varphi \rightarrow \psi \mid \Diamond\varphi \mid @_s\varphi \mid \forall s\ \phi \mid \exists s\ \phi$$

where $x \in \mathbf{VAR}$ *,* $s, s_1, s_2 \in \mathbf{VAR}$ *and* $\mathbf{VAR}$=$TERM(\mathbf{f}, \mathbf{SVAR})$*.*

Remark 1. Note that the difference between nominals and state variables is that nominals can not be bound by $\forall, \exists$, whereas state variables can.

Precise definitions regarding the semantics of strong hybrid languages can be found in [10]. Strong hybrid languages and $\mathcal{HLOV}(@, \forall, \exists)$, give us full first order expressive power over frames. This is known since the work of Arthur Prior [22] and [23].

Definition 8. *(Hybrid Translation)* $\mathcal{HLOV}(@, \forall, \exists)$ *is strong enough to express any first order concept. We extend the hybrid translation* $\mathcal{HT}(\cdot)$ *(as was defined in [5]), with the following case:*

$$\mathcal{HT}(\mathbf{f}(x, y)) = \mathbf{f}(x, y) \tag{1}$$

where $\mathcal{HT}(\cdot)$ *maps first order sentences to hybrid formulas of* $\mathcal{HLOV}(@, \forall, \exists)$*.*

Proposition 1. *Let* ϕ *be a first order formula. Then for every model* $\mathcal{M}$ *and any assignment* g*,* $\mathcal{M} \models \phi[g]$ *iff* $M, g \models \mathcal{HT}(\phi)$*.*

Proof. Straightforward. Based on proposition 3.7, on page 45 of [1].

Is worth mentioning that adding to $\mathcal{HLOV}(@, \forall, \exists)$'s syntax, nominals and function over nominals then we can extend the $\mathcal{HT}(\cdot)$ translation as follows:

$$\mathcal{HT}(x = c) = @_x c \tag{2}$$

$$\mathcal{HT}(f(c_1, c_2)) = f(c_1, c_2) \tag{3}$$

Therefore, in terms of frame correspondence langauge (definition 3.4 of [6]), $\mathcal{HLOV}(\forall, \exists, @)$ is equally expressive with the first order logic with equality over the vocabulary containing constants, functions over constants, first order variables, functions over first order variables, binary relation symbols and unary predicate symbols.

3.2 $\mathcal{HLON}$ and $\forall-\mathcal{HLOV}$

In this section we study the concept of frame definability in the context of $\mathcal{HLON}$. Note that by $\forall-\mathcal{HLOV}$ we mean the universal fragment of $\mathcal{HLOV}(@, \forall, \exists)$. In what follows, we show that any formula of $\forall - \mathcal{HLOV}$ can be expressed by a pure formula of $\mathcal{HLON}$.

Definition 9. *Call a sentence of $\mathcal{HLOV}(@, \forall, \exists)$ of the form:*

$\forall x_1 ... x_n \phi(x_1, ..., x_n, \mathbf{f}_1, ..., \mathbf{f}_k)$*, where ϕ does not contain any quantifiers, propositional letters or nominals a PUNF-sentence (***P****ure,* ***U****niversal,* ***N****ominal free sentence). Note that $x_1, ..., x_n$ and $\mathbf{f}_1, ..., \mathbf{f}_k$ are the state variables and the function symbols over variables occurring in ϕ, respectively. (PUNF-sentences are actually $\forall - \mathcal{HLOV}$ sentences)*

Example 1. An $\forall - \mathcal{HLOV}$ sentence is:

$$\forall x \ \ \Diamond\Diamond \mathbf{f}(x) \rightarrow \Diamond \mathbf{f}(x)$$

Proposition 2. *1. For any pure formula ϕ of $\mathcal{HLON}$ there is a PUNF-sentence $\phi^{\forall}$ of $\mathcal{HLOV}(@, \forall, \exists)$ such that for any frame $\mathcal{F}$, ϕ is valid on $\mathcal{F}$ iff $\phi^{\forall}$ is true on $\mathcal{F}$.*

2. For any PUNF-sentence ϕ of $\mathcal{HLOV}(@, \forall, \exists)$ there is a pure formula ϕ^B of $\mathcal{HLON}$ such that for any frame $\mathcal{F}$, ϕ is true in $\mathcal{F}$ iff ϕ^B is valid on $\mathcal{F}$. Where $\mathcal{F}$ is a functional relational frame.

Proof. 1. Suppose we are given a pure formula $\phi(i_1, ..., i_n, f_1, ..., f_k)$, where $i_1, ..., i_n$ are all the nominals and the function symbols over nominals in ϕ, then the required $\phi^{\forall}$ is $\forall x_1 ... x_n \phi(x_1, ..., x_n, \mathbf{f}_1, ..., \mathbf{f}_k)$

2. Any PUNF-sentence ϕ of $\mathcal{HLOV}(@, \forall, \exists)$ must have the form: $\forall x_1 ... x_n \phi(x_1, ..., x_n, \mathbf{f}_1, ..., \mathbf{f}_k)$ where ϕ contains no quantifiers, proposition letters or nominals. Therefore, the required ϕ^B is $\phi([i_1 \leftarrow x_1, ..., i_n \leftarrow x_n, f_1 \leftarrow \mathbf{f}_1, ..., f_k \leftarrow \mathbf{f}_k])$.

Note that $f_1, ... f_n$ are function over nominals and $\mathbf{f}_1, ..., \mathbf{f}_k$ over state variables.

Based on this proposition we will show that $\mathcal{HLON}$ is strong enough to express any equationally definable class of algebras.

4 Examples

A number of examples is presented discussing the expressive power of $\mathcal{HLON}$. We start studying the connections between $\mathcal{HLON}$ and equational systems. In section 4.2 we discuss whether we can express, using pure formulas of $\mathcal{HLON}$, graph properties not definable in the first order logic. In fact, we show that pure formulas of $\mathcal{HLON}$ are strong enough to express frame properties definable in the second order logic; this a new result as far as we know.

4.1 $\mathcal{HLON}$ and Equational Systems

An example of the expressive power of $\mathcal{HLON}$ is:

Example 2. (**Semigroups**) Let G be a nonempty set and $*$ be an operation on G. By the latter we mean a rule $*$ which assigns to each pair (x, y) of elements in G a unique element denoted $x * y$ in G. Now consider the following condition:

– (G1) The operation is associative: $\forall\ x, y, z\ \ x * (y * z) = (x * y) * z$.

If (G1) holds, then G is said to be a semigroup under $*$. The condition $(G1)$ in $\mathcal{HLON}$ can be expressed by the formula: $@_{i*(j*k)}(i * j) * k$, substituting nominals for variables.

Indeed the process is completely general. $\mathcal{HLON}$ is strong enough to express any equationally definable class of algebras. Let's be more precise summarizing some elementary definitions regarding equational systems ([6], Appendix B).

Definition 10. *(Syntax of Algebras) Let σ an algebraic similarity type and a set $\mathcal{X}$ of variables. The language $\mathcal{E}$ of algebras (equational systems) is built up as follows: $t := x \mid f(t_1, ..., t_{\rho(f)})$, we call t, $\mathcal{E}-$ terms.*

Definition 11. *(Equation System) An $\mathcal{E}(X, \sigma, =)$ equation system is an expression of the form: $s = t$, where s, t are $\mathcal{E}-$ terms.*

Definition 12. *(Semantics of Equational systems) A structure $\mathcal{M}$ for the equational system $\mathcal{E}$ is $\mathcal{M} = (D, (F^f)_{f\in\sigma}, V)$ such that: $D \neq 0$, a non empty domain D; for each $n \geq 0$ and for all $f \in \sigma$, if f is an $n-$ary function symbol then F^f is an $n-$ary operator on D. Note that $F : D^n \to D$; A variable x is assigned a meaning $V(x)$ which is an element D; The $V(t)$ where t is a term, is defined as follows:*

– *If t is a variable x then: $V(x)$ is an element D;*
– *If t is $f(t_1, \cdots, t_n)$ then: $V(f(t_1, \cdots, t_n)) = F(V(t_1), \cdots, V(t_n))$*

That is, an equational system is a set of equations. In other words an equational system is a set of universally closed equations of the form: $\forall x_1, ..., x_n\ \ s = t$, where $x_1, ..., x_n$ is the set of variables occurring in the terms s, t. Therefore we have:

Proposition 3. *For any equation of the form $\forall x_1, ..., x_n\ \ s = t$ there is a pure formula ϕ of $\mathcal{HLON}$ such that for any frame $\mathcal{F}$, ϕ is true on $\mathcal{F}$ iff ϕ is valid on $\mathcal{F}$.*

Proof. As we have already mentioned, $\mathcal{HLOV}(@, \forall, \exists)$ is strong enough to express any first order concept. Therefore any formula of the form $\phi = \forall x_1, ..., x_n\ \ s = t$ can be expressed by the following $\mathcal{HLOV}(@, \forall, \exists)$ formula: $\phi^{'} = \forall x_1, ..., x_n\ \ @_s t$, preserving truth satisfiability. But $\phi^{'}$ preserves frame validity as well, since the $\forall$ quantifier captures the effect of trying out all possible assignments to the variables.

Then by proposition 2 we have that there is a pure formula ϕ^{nom} of $\mathcal{HLON}$, being valid on a frame $\mathcal{F}$ if and only if ϕ is valid on $\mathcal{F}$.

An immediate consequence of theorem 1 is that adding the $@-$ operator to the algebraic langauge (definition 2) we obtain a full equational system.

Definition 13. *A propositional equational system $\mathcal{PES}$ of nominals (***ANOM***, σ, @) is built up using an algebraic type $\sigma = (A, \rho)$, a set of atomic nominals* **ANOM** *and the @− operator. Its formulas are given the following recursive definition:*

$$\alpha := i \mid f(\alpha_1, \cdots, \alpha_{\rho_{(f)}}) \mid @_{\alpha_1}\alpha_2$$

where $\alpha_1, \alpha_2 \in$ **TERMS**=(**ANOM**, σ, @).

To sum up, since the @− operator can be regarded as the equality symbol of the first order logic, is abundantly clear that pure formulas of $\mathcal{HLON}$ can express equations, therefore algebras. To be precise, $\mathcal{HLON}$ is strong enough to express those algebras satisfying the following theorem of Birkhoff [3]:

Theorem 1. *A class of algebras is equationally definable if and only if is a variety.*

Another example regarding the expressive power of $\mathcal{HLON}$ is the following:

Example 3. Many important terms of universal algebra ([12]) like a discriminator term or a switching term are naturally expressed by $\mathcal{HLON}$ pure formulas. For instance:

$$(@_a b \rightarrow @_c t(a, b, c)) \vee (\neg @_a b \rightarrow @_a t(a, b, c)) \text{ discriminator term}$$

$$(@_a b \rightarrow @_c s(a, b, c, d)) \vee (\neg @_a b \rightarrow @_d s(a, b, c, d)) \text{ switching term}$$

4.2 Linguistics and Graph Theory

It is useful to mention some linguistic aspects of functions symbols in the context of hybrid logics. Actually using functions is possible to express sentences that speak about themselves. We will be clear on this point. First we should recall that a formula $@_i f(i)$ expresses the fact that $f(i)$ holds at the world named i and something stronger, that $f(i)$ depends on i.

Example 4. For instance the formula (can be found in [25]): **The (only) woman that every man loves is his mother**, illustrates a functional reading. The pronoun *his* is bound by the noun phrase *every man*, although it is not within its syntactic scope. We can get what we want by the formula, where f_{only} maps every man to his mother:

$$@_i < loves > f_{only}(i)$$

Roughly speaking the function $mother(i)$ is very similar to the notion of *Henkin Quantifiers*, expressing the notion of independence [9] and [21]. We continue using some terminology from graph theory:

Definition 14. *A k-coloring of an undericted graph $G = (V, E)$ is a function $col : V \to \{1, ..., k\}$ such that $col(u) \neq col(v)$ for all $(u, v) \in E$. G is called $k-$ colorable if there is a $k-$ coloring of G.*

Example 5. (k-self referred functions) Assume that our hybrid frame is $\mathcal{F} = (W, R, F_1^{f_1}, ..., F_k^{f_k})$ where $F_l^{f_l}$ are unary operators on W. $\mathcal{HLON}$'s syntax is:

$$\varphi := i \mid f_1(i) \mid ... \mid f_k(i) \mid p \mid \neg\varphi \mid \varphi \to \psi \mid \Diamond\varphi \mid @_i\varphi$$

where the intended meaning of $f_i(k)$ is: the color of i is k. For simplicity, we will be soppy in our notation, assuming that the definition above is recursive, which clearly is not. Note that $F_l^{f_l}$ $(l = 1, ..., n)$ is the interpretation of f_l and:

$$F_l^{f_l}(w) = \begin{cases} w, \text{ if } w \text{ is } l \text{ colored} \\ \\ !w, \text{ otherwise} \end{cases}$$

where $!w$, means some world other than w.

Therefore the condition $col(u) \neq col(v)$ if $(u, v) \in E$ is expressed by the formula:

$$\phi = \bigwedge_{l=1,...,k} (@_i\Diamond j \wedge @_i f_k(i)) \to \neg @_j f_k(j)$$

Also, the condition that every node is colored can be expressed as follows:

$$@_i f_1(i) \vee ... \vee @_i f_k(i)$$

It is well known that the property of $2-$ colorability is not definable in first order logic; [14] and definable in monadic second order logic. We show that this property is definable by $\mathcal{HLON}$ pure formulas as well.

Let ϕ be a conjunction of:

$$@_i f_1(i) \vee @_i f_2(i)$$

$$\bigwedge_{l=1,2} (@_i\Diamond j \wedge @_i f_l(i)) \to \neg @_j f_l(j)$$

Then ϕ is valid on the frame $\mathcal{F}$ if and only if the graph defined by the relation R is $2-$ colorable. Note that a Kripke frame is the reduct of a functional relational frame validating ϕ if and only if the graph defined by the relation R is 2 colorable.

That is, we have shown that using pure nominal formulas we can express properties not definable in first order logic and definable in monadic second order logic.

4.3 Frame Definability and $\mathcal{HLON}$

In this section we show that any frame property definable by a formula of the strong hybrid language can be expressed using pure formulas of $\mathcal{HLON}$. We start with an example motivating our formalism.

Example 6. (**Right directed**). Right directness can be defined by the following formula of the strong hybrid language $\mathcal{HLON}(@, \forall, \exists)$:

$$\phi = \forall v, u \exists t \ (@_v \diamond t \wedge @_u \diamond t)$$

Let us assume that ϕ is valid on a frame $F = (W, R)$. Then we can expand the given frame by adding a new binary function symbol F^f, obtaining the sentence $\phi^{'}$: $\forall v, u \ @_v \diamond \mathbf{f}(u, v) \wedge @_u \diamond \mathbf{f}(u, v)$ such that:

$$\phi \text{ is valid on } \mathcal{F} \text{ iff } \phi^{'} \text{ is valid on } \mathcal{F}^{'}$$

where $\mathcal{F}^{'} = (W, R, F^f)$. By proposition 2 we get:

$$\phi \text{ is valid on } \mathcal{F} \text{ iff } \phi^{nom} \text{ is valid on } \mathcal{F}^{'}$$

where $\phi^{nom} = @_i \diamond f(i, j) \wedge @_j \diamond f(i, j)$.

We assume that i and j are particular names, but their interpretation in the model is arbitrary, so the universal quantification is implicit. Note that no orthodox modal formula defines the condition of $right - directed$.

Indeed the process can be shown to be completely general. Essentially, to express an $\mathcal{HLOV}(@, \forall, \exists)$ formula, we introduce functions symbols to act as Skolem functions and obtain a pure $\mathcal{HLON}$ formula whose validity is equivalent to that of the given $\mathcal{HLOV}$. In [24] we proved a completeness result for functional relational frames and $\mathcal{HLON}$ pure nominal formulas.

It is a routine exercise to show that every formula of $\mathcal{HLOV}(@, \forall, \exists)$ can be rewritten in to a PNF equivalent. Recall that, a formula is on Prenex normal formula (PNF) if it has the form $Q_1 x_1 \ Q_2 x_2 \ \cdots \ Q_n x_n \ \phi$ where $Q_i \in \{\forall, \exists\}$ are quantifiers and ϕ is quantifier free.

Lemma 1. *Given the sentence $\exists y \ G(y)$ ($G(y)$ is a quantifier free formula) of $\mathcal{H}(@, \forall, \exists)$, augment the language with a new constant c and form the sentence $G(c)$ such that:*

$$G(y) \ \textit{is valid on} \ \mathcal{F} \ \textit{iff} \ G(c) \ \textit{is valid on} \ \mathcal{F}^{'}$$

Given the sentence $\forall x_1, \cdots, x_n \exists y \ G(\overline{x}, y)$ of $\mathcal{H}(@, \forall, \exists)$, augment the language with a new $n-$ ary function symbol f and form the sentence $\forall x_1, \cdots, x_n \ G^{'}(\overline{x}, \mathbf{f}(\overline{x}))$. Then,

$$\forall x_1, \cdots, x_n \exists y \ G(\overline{x}, y) \ \textit{is valid on} \ \mathcal{F} \ \textit{iff} \ \forall x_1, \cdots, x_n \ G^{'}(\overline{x}, \mathbf{f}(\overline{x})) \ \textit{is valid on} \ \mathcal{F}^{'}$$

where $\mathcal{F}$ and $\mathcal{F}^{'}$ are frames of $\mathcal{H}(@, \forall, \exists)$.

Proof. In each step we can expand a frame of the sentence on the left hand side to one on the right hand side by adding the appropriately designated element for the constant, or by adding an appropriate function $F^{\mathbf{f}}$, where F, is the interpretation of $\mathbf{f}$ in $\mathcal{F}$. Conversely, we can take a frame of the sentence on the right hand side and remove the constant symbol, respectively function symbol and we get a frame for the left hand side.

We call the procedure of removing one existential quantification and replacing the variable it bound by a functional term, as in the Lemma 1, a *Skolemization step*. We use the term *Skolemization* for the removal of all existential quantifications in this manner:

Theorem 2. *Given a formula ϕ of the strong hybrid language $\mathcal{H}(@, \forall, \exists)$ there is an effective procedure for finding a universal sentence $\phi^{\forall}$ of $\forall - \mathcal{HLOV}$ such that:*

$$\phi \text{ is valid on } \mathcal{F} \text{ iff } \phi^{\forall} \text{is valid on } \mathcal{F}^{'}$$

Furthermore, we can choose $\phi^{\forall}$ such that every frame of ϕ can be expanded to a frame of $\phi^{\forall}$ and every frame $\phi^{\forall}$ can be reducted to frame of ϕ.

Proof. First we put ϕ in prenex form (this is straightforward). Then we just apply the lemma 1 repeatedly until there are no existential quantifiers.

Then by proposition 2,

Theorem 3. *For any frame $\mathcal{F}$ definable by a formula ϕ of the strong hybrid language $\mathcal{HLOV}(@, \forall, \exists)$ we can associate a pure formula of $\mathcal{HLON}$ such that:*

$$\phi \text{ is valid on } \mathcal{F} \text{ iff } \phi^{nom} \text{ is valid on } \mathcal{F}^{'}$$

To sum up, we have shown that the traditional fragment of the hybrid logic becomes in a rather unorthodox interpretation equivalent to first order logic by extending the basic hybrid language with functions working on nominals denoting worlds in the frame.

5 Conclusions

We hope that we have convinced the reader that is useful to add function symbols in the context of hybrid logics. Roughly speaking, we have shown that $\mathcal{HLON}$ is strong enough to express, any equationally definable class of algebras or the property of 2− colorability of a graph (not definable in the first order logic).

I am indebted to Robin Hirsch for his assistance and guidance throughout. I would like to thank the three anonymous referees for their useful comments and suggestions.

References

1. Carlos Eduardo Areces. Logic Engineering, The case of Description and Hybrid Logics, PhD thesis, ILLC, University of Amsterdam, 2000.
2. C. Areces, P. Blackburn, and M. Marx, Hybrid logics, Characterization, interpolation and complexity. Journal of Symbolic Logic, 66(3), 977-1010, 2001.
3. G. Birkhoff, On the structure of abstract algebras, Proc. Cambr. Philos. Soc. 31 (1935), 433-454.
4. P. Blackburn, Internalizing labelled deduction. Journal of Logic and Computation,10(1): 137 - 168, 2000.
5. Patric Blackburn, Representation, Reasoning, and Relational Structures: a Hybrid Logic Manifesto. Proc. of the 1st. Method for Modalities Workshop. Amsterdam. Areces, C., Franconi, E., Gor, R., de Rijke, M., Schlingloff, H., editors. Special Issue of the Logic Journal of the IGPL. Vol 8:3, 339-625, 2000.
6. Patrick Blackburn, Maarten de Rijke and Yde Venema, Modal Logic, Cambridge University Press, 2001.
7. Patrick Blackburn and Balder ten Cate, Beyond Pure Axioms: Node Creating Rules in Hybrid Tableaux. In Carlos Areces, Patrick Blackburn, Maarten Marx and Ulrike Sattler, editors, Proceedings of the 4th Workshop on Hybrid Logics (HyLo 2002).
8. P. Blackburn and M. Tzakova. Hybrid languages and temporal logic. Logic Journal of the IGPL, 7(1):2754, 1999.
9. A. Blass and Y. Gurevich. Henkin quantifiers and complete problems. Ann. Pure Appl. Logic 32:1, 1-16 (1986).
10. P. Blackburn and J. Seligman. Hybrid Languages. Journal of Logic, Langauge and Information, 4(3): 251-272, 1995.
11. R. Bull. An approach to tense logic. Theoria, 36(3):282300, 1970.
12. Stanley Burris, H P Sankappanavar, A course in universal algebra, Springer-Verlag, New York,1981.
13. Balder ten Cate, Model theory for extended modal languages. PhD thesis, University of Amsterdam, 2005.
14. B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, Handbook of Theoretical Computer Science, volume Volume B: Formal Models and Sematics, pages 193-242. Elsevier and MIT Press, 1990.
15. Bruno Courcelle. The Expression of Graph Properties and Graph Transformations in Monadic Second-Order Logic. Handbook of Graph Grammars 1997: 313-400
16. George Gargov and Valentin Goranko. Modal Logic with Names. Journal of Philosophical Logic, 22(6):607636, 1993.
17. Valentin Goranko and Dimiter Vakarelov. Universal Algebra and Modal Logic, in: Advances in Modal Logic, vol. II, CSLI publications, Stanford, 2000, 265-292.
18. Melvin Fitting, First Order logic and automated theorem proving. Graduate Texts in Computer Science. Springer Verlag, Berlin, second edition, 1996.
19. Hylo page. http://www.hylo.net/, Papers, Bibliography, Course and Slides....
20. O.Kutz, E-Connections and Logics of Distance. Ph.D. thesis, University of Liverpool, 2004.
21. Ahti-Veikko Pietarinen, Tero Tulenheimo. An introduction to IF logic, Course material, 16th European Summer School in Logic, Language and Information, Universite Henri Poincare, Nancy, France.
22. A. Prior, Past, Present and Future. Oxford: Clarendon Press, 1967.
23. A. Prior, Papers on Time and Tense. Oxford: Clarendon Press, 1968.

24. Evangelos Tzanis, Algebraizing Hybrid Logic. Msc Thesis, University of Amsterdam, 2005. (http://www.illc.uva.nl/Publications/ResearchReports/MoL-2005-03.text.pdf)
25. Yoad Winter, Functional Quantification. Research on Language and Computation 2:331-363,2004.

A Modal View on Polder Politics

Wouter van Atteveldt, Stefan Schlobach

Vrije Universtiteit Amsterdam, The Netherlands

{wva,schlobac}@few.vu.nl

Abstract

To get a better understanding of the influence of the media on current political events, researchers in Communication Science have recently reported on several case studies, in which they analyse certain patterns in newspaper articles, such as internal disagreement or criticism within a political party. In this paper we explain how Hybrid Logic can be used as a query language in such an analysis, and show that we can use existing model checkers to reproduce results of a recently published case study about the Dutch election campaign in 2002. [1]

1 Introduction

The media play an increasing role in the functioning of our society, often being the main informant of citizens and policy-makers. Controversies easily arise when the media are accused of providing inaccurate or biased information, such as the BBC-affair leading to the suicide of David Kelly in the UK and the alleged demonisation of Pim Fortuyn before his assassination in the Netherlands. The messages of politicians reach most citizens through the media rather than directly, making the media image of a party or candidate and their reported positions on key issues an important factor in election campaigns. Therefore, it is important to analyse the content of the media and its effects on media consumers, as done in for example election studies, agenda setting, and framing research (3; 11). These studies generally annotate news items using Content Analysis which has resulted in a large number of annotated articles over the past decades.

In this paper we provide a uniform representation of these media data by formalising them as Kripke structures. Not only do we get unified representations with formal semantics, we also inherit well studied query languages in the form of modal logics. This allows us to use off-the-shelf model checking

[1] The Polder model is a phenomenon in politics in the Netherlands to reach an agreement despite differences, however great these might be.(15)

software to query newspaper articles, or at least the formalisation of these articles. In order to reproduce recent results of a media study about the Dutch elections in 2002 (9), we show that quite expressive modal languages (such as Hybrid Logic $H(@, \downarrow)$) are needed to capture important political terms, such as internal debate or criticism. This approach is verified experimentally by transforming around 7,500 newspaper articles into Kripke structures. Using an existing hybrid logics model checker (6), we query a number of politically interesting patterns and compare them to the original study in (9).

There are three significant advantages of our approach: it offers an explicit semantics for the formal representation of newspaper documents, which makes the study much more transparent and therefore reproducible. Secondly, by using existing tools, the amount of purpose-written and ad-hoc software is greatly reduced, and, finally, it becomes straightforward to integrate formally represented background knowledge.

2 Analysing the News

Quantitative Content Analysis is a field of Communication Science that analyses media messages by categorising units of text into scientifically relevant concepts (10). Network or Relational Content Analysis methods, such as the NET method described below, additionally construct a network from these texts where the nodes are the categories mentioned above and the vertices determine certain relations between these nodes as expressed in the text (12; 5). These relations are often of a fairly subjective or opinionative nature, such as (dis)like between actors and issue positions, but can also include causal relations or actions, such as passing a law or going on strike.

In this paper, a simplified version of the Network Content Analysis method called the NET method will be used. The NET method (Network analysis for Evaluative Texts; 9), assumes that all relevant information in a sentence can be expressed as $\langle$`subject`, `relation`, `object`$\rangle$ triples. The `subject` and `object` of these statements are drawn from a (generally fixed) list of relevant concepts, and the `relation` is a binary predicate indicating dissociation or association.

Automatically extracting these relations from text is an open problem on the intersection of Communication Science and Computational Linguistics. A full discussion of this is beyond the scope of this paper; the interested reader is referred to (14; 13) for two examples of work in this field. In this paper we assume that we have such a network to start with, either from one of the many manual studies, or from a (future) automated analysis.

The resulting network can be queried in different ways depending on the research question. In general, the researcher (implicitly) defines relevant concepts as patterns over these networks. In election studies, such as the study that provided the data underlying this paper, it is interesting to know what the general tone and type of the news is, who is portrayed positively or negatively, and what issue positions the different actors have according to the

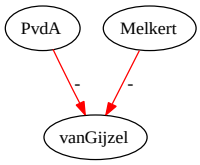
media. Moreover, more specific concepts such as internal dissent, unclear issue positions and criticism from political actors can also be important (9).

2.1 NET by example: the Rise and Fall of Fortuyn

The examples in this paper are all drawn from the media coverage of the 2002 parliamentary elections in the Netherlands as described in (9). Initially, these elections appeared to become a duplicate of the 1998 polarisation between the ruling parties, the PvdA (labour) and VVD (conservative), with both criticising each other but generally intending to continue the coalition. The PvdA, however, was hindered by internal dispute, as exemplified in an article published in the Dutch newspaper "Trouw" on 27 November 2001. Figure (1) shows two sentences of this article which are indicative of this internal strife, and which are coded as ⟨`Melkert`,–, `vanGijzel`⟩ and ⟨`PvdA`,–, `vanGijzel`⟩.

Fig. 1: 'Rob van Gijzel fights back' *(Trouw, 27 November 2001)*

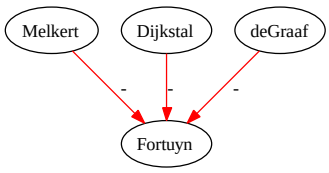
PvdA-leader Melkert took away the spokesmanship on the construction fraud affair from Rob van Gijzel. [..] This shows that he can no longer assume support within his party



Not all went as expected by the ruling parties. On 9 February 2002, the newcomer Pim Fortuyn gave an interview in de Volkskrant which caused him to be seen as extreme right, and forced him to establish his own party to continue in Dutch politics. These radical issue positions caused him to receive heavy criticism from the established parties. The sentences quoted in figure 2 are all direct criticism on Fortuyn by the other party leaders ⟨`Dijkstal`,–, `Fortuyn`⟩, ⟨`Melkert`,–, `Fortuyn`⟩ and ⟨`De Graaf`,–, `Fortuyn`⟩. Note, that any meaningful interpretation of this graph requires some background knowledge, for example, that Dijkstal, Melkert and De Graaf were the party leaders of the VVD, PvdA and D66 at the time.

Fig. 2: "Public urinator" *(Trouw, 13 February 2002)*

[Before the interview] people had some second thoughts about Fortuyn, but now he is said to 'encroach on the heart of our civilisation' (Dijkstal), 'have crossed a line, that you cannot cross' (Melkert) [..]. De Graaf could not help citing Anne Frank and calling him a fascist and a racist.

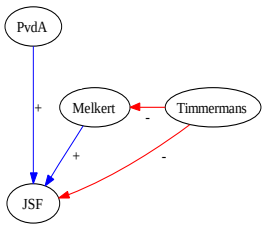


Fortuyn did surprisingly well in the local elections in Rotterdam later that month, which caused the leader of the PvdA, Melkert, to come under heavy

fire. Moreover, the PvdA was still plagued with internal disagreement, for example about the participation in the Joint Strike Fighter project (Figure 3). The first sentence is coded as ⟨`Timmermans`,–, `JSF`⟩, ⟨`PvdA`,+, `JSF`⟩, the second as ⟨`Timmermans`,–, `Melkert`⟩ and ⟨`Melkert`,+, `JSF`⟩.

Fig. 3: 'Difficult, but I never threatened to resign' *(de Volkskrant, 10 April 2002)*

PvdA parliamentary Frans Timmermans remains skeptical about the JSF decision of his party [..] after the PvdA-fraction finally said 'Yes' to the JSF. Timmermans played the main part and diametrically oppose party leader Melkert on this issue



The tragic end of the campaign is known: Pim Fortuyn was assassinated nine days prior to the elections. His party (the List Pim Fortuyn) made an unprecedented debut and was made part of the government, but it was plagued by internal dissent over the succession and was diminished in the next election after the cabinet fell, less than three months after its formation.

2.2 Healthy Conflict or Internal Dissent

In campaign time, conflict and criticism are the rule rather than the exception. However, there is an important distinction between different types of conflict: conflicting or disputing issues with opponents can be beneficial as it allows a party to create a distinct profile. Internal criticism, however, is generally harmful, as is criticism from societal actors. Also, if different members of a party adopt opposing viewpoints on an issue, the position of the party is unclear to the voter and can be seen as indicating internal dissent or lack of leadership.

Thus, it is important to identify whether the criticiser is a member of the same party or a different party, and whether opposing viewpoints are adopted by members of different parties or of the same party. This means we need to know the party membership of the actors that are under investigation. Using this background knowledge, we can see that figure 1 indicates *internal criticism*, figure 3 indicates *internal disagreement*, while figure 2 indicates *criticism by opponents* of Fortuyn. These concepts are therefore defined as patterns over a combination of media data and background knowledge.

3 Querying opinions through model checking

In the previous section we introduced a prototypical problem exemplifying the way social scientists use information about opinions to analyse political situations. We now show that NET graphs can be mapped to particular Kripke

structures, and that hybrid modal languages are useful query languages over these structures. In particular, we show that we can reformulate the case study of the previous section as a hybrid model checking problem.

Not only can we reproduce the published results with our approach, as will be shown in the following section, but the expressiveness of the querying languages allows one to extend the complexity of possible sociological analysis of the underlying newspaper articles. Furthermore, it provides a formal interpretation to the experiments. Moreover, it makes it easier to do meta-research over different communication scientific studies by formalising the semantics and hence allows for improved theory building.

3.1 NET graphs and background knowledge

In section 2 we loosely introduced NET graphs as a result of a specific method of content analysis. These graphs can be characterised more formally as follows: let $\mathcal{N} = \{n_1, \ldots, n_l\}$ be a finite set of names, which usually represent political actors, groups and issues. A *NET graph* is then defined as a finite set of triples $\langle n_i, *, n_j \rangle$, where $n_i, n_j \in \mathcal{N}$ are names, and $* \in \{+, -\}$ is an opinionated relation, denoting consent and dissent, respectively. As an example, consider the NET graph NET_{ex} which was introduced to exemplify internal disagreement between two members of the same party, and which is repeated on the left-hand side of figure 4. Here, we have four names, Melkert, Timmermans, PvdA and JSF, with triples $\langle$`Timmermans`,–, `JSF`$\rangle$, $\langle$`PvdA`,+, `JSF`$\rangle$, $\langle$`Timmermans`,–, `Melkert`$\rangle$ and $\langle$`Melkert`,+, `JSF`$\rangle$.

Background knowledge is usually given as a simple ontology over political actors and concepts. Formally, let $\mathcal{N}$ again denote the set of names, $\{R_1, \ldots, R_n\}$ a finite set of binary relations, and $\mathcal{P} = \{P_1, \ldots, P_m\}$ a finite set of propositional variables. The *domain ontology* $\mathcal{O}$ is then defined as a finite set of axioms of the form $(n_i : P), P \sqsubseteq Q$ or $R(n_i, n_j)$, where $n_i, n_j \in \mathcal{N}, \{P, Q\} \subseteq \mathcal{P}$ and $R \in \{R_1, \ldots, R_n\}$. In our example, both Timmermans and Melkert are politicians, and members of the PvdA, which is a party, and the JSF is an issue; finally, politicians and parties are political actors. This can be represented formally in the ontology $\mathcal{O}_{ex} = \{$Timmermans:Politician, Timmermans:memberOf PvdA, Melkert:Politician, Melkert:memberOf PvdA, JSF:Issue, PvdA:Party, Politician $\sqsubseteq$ PolActor$\}$.

3.2 NET as Kripke structures

Kripke structures over a set of propositional variables $\mathcal{P}$, are triples of the form $\langle M, \mathcal{R}, V \rangle$, where M is a set of states, $\mathcal{R}$ a family of binary accessibility relations on M, and V is a valuation function from $\mathcal{P}$ to the power-set of M. We can translate a NET structure NET and background ontology $\mathcal{O}$ in the form described above into a kripke structure as follows:

For each name n_i in $\mathcal{N}$, we create a state m_i in M. Let us define two relations $\mathcal{R}_+$ and $\mathcal{R}_-$, where $(m_i, m_j) \in \mathcal{R}_+$ iff $\langle n_i, +, n_j \rangle \in NET$ and an

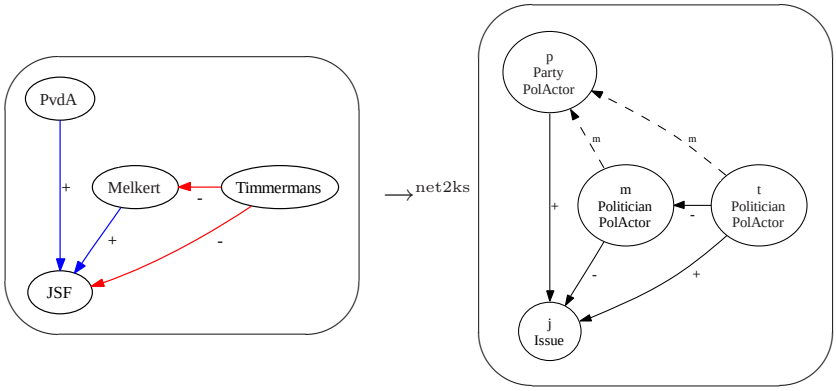


Fig. 4. The NET graph of figure 3 and its Kripke structure $\mathcal{K}_{ex}$

analogous definition for $\mathcal{R}_-$. Further, we define a relation $\mathcal{R}_i$ for each relation R_i in the ontology $\mathcal{O}$, where $(m_i, m_j) \in \mathcal{R}_i$ iff $(n_i, n_j) \in R_i$. Now, we can equate $\mathcal{R} = \{\mathcal{R}_-, \mathcal{R}_+, R_1, \ldots, R_n\}$. Finally, we define the valuation function V as the smallest set $V(P)$ such that $\{n \mid n : P \in \mathcal{O}\} \subseteq V(P)$ and $\bigcup_{(P \sqsubseteq Q) \in \mathcal{O}} V(Q) \subseteq V(P)$ for each $P \in \mathcal{P}$.

Please note that the way we translate the $P \sqsubseteq Q$ ontological statements into our valuation function effectively means unfolding these axioms into propositional symbols attached to the states, which is an implicit Closed World Assumption. This is a necessary assumption if we wish to use a model checker to evaluate formulas, as a model needs to have all valuations specified. Moreover, we feel that in our application this is an acceptable assumption because of the restricted domain.

Now, we apply this translation to the NET graph NET_{ex} and the ontology $\mathcal{O}_{ex}$. From the set of names $\{Melkert, Timmer, PvdA, JSF\}$ we create the set of states $M_{ex} = \{m, t, p, j\}$. The family of accessibility relations is defined as $\mathcal{R}_{ex} = \{\mathcal{R}_-, \mathcal{R}_+, R_m\}$, where the set of NET triples is translated into the definition of $\mathcal{R}_+ = \{(p, j), (m, j)\}$ and $\mathcal{R}_- = \{(t, j), (t, m)\}$; and the ontological relation $memberOf$ is translated into $\mathcal{R}_m = \{(t, p), (m, p)\}$. Finally, the valuation function V_{ex} is derived from taking the instances and unfolding the 'subclass' relation in the ontology: $V_{ex}(Politician = \{t, m\}$, $V_{ex}(Party) = \{p\}$, $V_{ex}(PolActor) = \{t, m, p\}$, and $V_{ex}(Issue) = \{j\}$. This yields the Kripke structure $\mathcal{K}_{ex} = \langle M_{ex}, \mathcal{R}_{ex}, V_{ex} \rangle$ visualized on the right hand side of figure 4.

This translation might seem to be mainly a change from normal to calligraphic font, i.e. the syntactic translation is hardly enervating. However, with this translation we also define the semantics of the joint network of NET and the background ontology. Since NET has no formal semantics attached to it, and no previous work exists to provide a formal framework for interpreting the (social scientific) background knowledge, this is a conceptually interesting and important step. It is worth mentioning that this semantic explicitation is based on a number of assumptions that Social Scientists might agree upon or

not. But by providing an explicit semantics to NET (with background knowlege) by means of a translation into Kripke structures we introduce a formal way of interpreting communication scientific concepts or theories. This can provide insight into the (often implicit) discrepancies between the assumptions behind studies.

3.3 Modal logics as query languages over Kripke structures

Interpreting NET graphs and their domain ontologies as Kripke structures has two big advantages: first, it provides a unified representation about all the knowledge that is represented about a certain political statement. Secondly, Kripke structures come with a number of very powerful and well studied query languages, namely modal logics (1). In our case, multi-modal logics seem to be the obvious choice for querying multi-relation structures. They extend propositional logic by adding a family of modal operators $\{\Diamond_1, \ldots, \Diamond_d\}$. Formally, let $\mathsf{PROP} = \{p, q, \ldots\}$ be a set of propositional variables. The syntax of multi-modal logic MML is as usual: $\varphi := \top \mid p \mid \neg\varphi \mid \varphi \wedge \psi \mid \Diamond_i$. To simplify matters, $\varphi \vee \psi$ and $\Box_i\varphi$ are defined as abbreviations for $\neg(\neg\varphi \wedge \neg\psi)$ and $\neg\Diamond_i\neg\varphi$. Let $\mathcal{K}$ be a Kripke structure and $m \in M$. The semantics of multi-modal logic is defined as usual:

$$
\begin{array}{lll}
\mathcal{K}, m \models \top & & \\
\mathcal{K}, m \models p & \text{iff} & m \in V(p),\ p \in \mathsf{PROP} \\
\mathcal{K}, m \models \neg\varphi & \text{iff} & \mathcal{K}, m \not\models \varphi \\
\mathcal{K}, m \models \varphi \wedge \psi & \text{iff} & \mathcal{K}, m \models \varphi \text{ and } \mathcal{K}, m \models \psi \\
\mathcal{K}, m \models \Diamond_i\varphi & \text{iff} & \exists m'\ (R_i m m' \wedge \mathcal{K}, m' \models \varphi)
\end{array}
$$

To see the potential of modal logic as a query language, let us study an example: *give me politicians criticising other politicians?* Asking this query over the Kripke structure $\mathcal{K}_{ex}$ should return the name Timmermans, because $\langle$`Timmermans`,–, `Melkert`$\rangle \in NET_{ex}$, as well as Melkert:PolActor$\in \mathcal{O}_{ex}$ and Timmermans:PolActor$\in \mathcal{O}_{ex}$. Formally, the set of answers to this query is the set of all worlds n for which $\mathcal{K}_{ex}, n \models$PolActor$\wedge\Diamond_$PolActor, where $\Diamond\_$ is interpreted as the relation –. More generally, given any Kripke structure $\mathcal{K}$, and an arbitrary formula φ in the basic language, it is well known that it can be checked in linear time whether φ is true at a state in $\mathcal{K}$ or not.

Unfortunately, the logic is not yet expressive enough for some important queries which are needed in our case study. First, we are unable to ask for adversaries of particular political actors, or opinions of politicians with respect to specific political issues. This is because modal logics usually do not allow to address worlds by their names, i.e. they usually don't have support for nominals. So, if we were to ask about those politicians opposing immigration, we would need the name of the issue immigration in our language. Then the query (Politician$\wedge\Diamond_$immigration) would capture our intuition.

But we need even more. A type of query that is required for the political investigations are about internal disagreement, that also cannot be expressed, even with nominals. In the following example we want to query for a political party, which has suffered criticism on a particular issue it supports from one of its members. Basically the query should include the proposition Party, i.e. we search for a party, and an issue this party supports, i.e. $\Diamond_m \Diamond_+ \mathsf{Issue}$. Finally, we need to find a member of the party, which is critical of the same issue. But here we fail, as we cannot state equality of two issues or two parties, respectively. What is needed is a further extension of our logic with weak quantification, or variable binding.

3.4 Hybrid Logics

Hybrid Logic (HL, for short) extends multi-modal logic with devices for naming states and accessing states by names. Let $\mathsf{NOM} = \{i, j, \ldots\}$ and $\mathsf{WVAR} = \{x, y, \ldots\}$ be sets of nominals and state variables, respectively. HL's syntax is: $\varphi := \mathrm{MML} \mid i \mid x \mid @_t\varphi \mid \downarrow x.\varphi \mid \exists x.\varphi$, with $i \in \mathsf{NOM}$, $x \in \mathsf{WVAR}$, $t \in \mathsf{NOM} \cup \mathsf{WVAR}$. The operators in $\{@, \downarrow, \exists\}$ are called *hybrid operators*. We call $\mathsf{WSYM} = \mathsf{NOM} \cup \mathsf{WVAR}$ the set of *state symbols*, and $\mathsf{ATOM} = \mathsf{PROP} \cup \mathsf{NOM} \cup \mathsf{WVAR}$ the set of *atoms*. For simplicity, we omit parenthesis after the $\downarrow$. For instance, $\downarrow x.p \wedge @_x q$ should be read as $\downarrow x.(p \wedge @_x q)$.

Hybrid Logic is interpreted over *hybrid Kripke structures*, i.e., Kripke structures $\langle M, R, V\rangle$ where the valuation function V assigns singleton subsets of M to nominals $i \in \mathsf{NOM}$. To give meaning to the formulas, we also need the notion of *assignment*. An assignment g is a mapping $g : \mathsf{WVAR} \to M$. Given an assignment g, we define g^x_m by $g^x_m(x) = m$ and $g^x_m(y) = g(y)$ for $x \neq y$. For any atom a, let $[V, g](a) = \{g(a)\}$ if a is a state variable, and $V(a)$ otherwise.

The semantics of hybrid logic is defined over a hybrid Kripke structure $\mathcal{K} = \langle M, R, V\rangle$, where $m \in M$, and g is an assignment; the semantics for Boolean and modal operators is as before.

$$
\begin{array}{lll}
\mathcal{K}, g, m \models a & \text{iff} & m \in [V, g](a),\ \ a \in \mathsf{ATOM} \\
\mathcal{K}, g, m \models @_t\varphi & \text{iff} & \mathcal{K}, g, m' \models \varphi, \text{ where } [V, g](t) = \{m'\},\ \ t \in \mathsf{WSYM} \\
\mathcal{K}, g, m \models \downarrow x.\varphi & \text{iff} & \mathcal{K}, g^x_m, m \models \varphi \\
\mathcal{K}, g, m \models \exists x.\varphi & \text{iff} & \text{there is } m' \in M \text{ such that } \mathcal{K}, g^x_{m'}, m \models \varphi
\end{array}
$$

This means that $@_t$ shifts evaluation to the state named by t, where t is a nominal or a variable. The down-arrow $\downarrow x$ binds the state variable x to the *current* state, while the existential binder $\exists x$ binds to *some* state in the model; $\downarrow$ and $\exists$ do not shift evaluation away from the current state. The definition of hybrid logic is almost literally taken from (7). We also refer to (2) for an introduction to hybrid logics.

By adding one or more of these hybrid operators to our modal language, we can obtain a number of hybrid logics, such as $H(@)$ and $H(@, \downarrow)$, with different expressivity and computational complexity. Which of these logics is

needed depends on the specific query to be formulated.

For example, to know if any politician is taking a negative issue position, simple modal logic suffices: $\mathsf{Politician}\Diamond\_\mathsf{Issue}$. If we also want to know which politician is taking this position, we need to interpret the hybrid structure, as we want to bind to a specific world rather than the whole Kripke structure, and thus arrive in $H(@)$, the least expressive hybrid logic.

If we want to formulate *internal disagreement*, we also need the $\downarrow$ binder to be able to require two accessibility 'branches' to reach the same state: $\downarrow x.\mathsf{Party} \wedge (\Diamond_m\Diamond_+(\mathsf{Issue} \wedge \downarrow y.(@_x\Diamond_m\Diamond_- y)))$. The outer binder binds the current state, i.e. the party we query for, to a variable. Inside the formula, we then enforce a criticism relation between a party member and the issue in question.

Hybrid Logics as query languages over Kripke structures stemming from NET graphs with background knowledge offer sufficient expressiveness to emulate realistic investigations in content analysis. Moreover, as there is a fully-fledged model checker publicly available, these results almost come for free, as we will show in the following section.

4 Experiments and results

The experimental validation of our proposed method is a secondary analysis based on articles coded for the investigation of the Dutch 2002 parliamentary campaign by (9). This data set consists of 7,478 newspaper articles and television items from 20 November 2001 to 14 May 2002 of which the headline and leading paragraph were coded, yielding 35,031 triples.

Each article was translated into a Kripke structure as described in 3.2, using a background ontology formlaising party membership of all politicians. Finally, propositions were added that identify politicians, parties, political actors (comprising politicians and parties), and issues. We then formalised the main political concepts as modal or hybrid logic formulas. Using the model checker described in (6, see also 7), these formal HL queries were then checked in each of these Kripke structures, resulting in a list of parties, politicians or issues for which these formulas were true.

These outcomes were summarised per period and compared to the results presented in the original study. Small discrepancies occur because in the original study the number of edges were counted rather than the number of worlds. In fact the difference occurs because the way we pose our queries: we search for an issue that is disputed by politicians, and not for politicians disputing an issue.

Horserace news:	$\mathsf{reality} \wedge (\Diamond_+ \mathsf{PolActor} \vee \Diamond_- \mathsf{PolActor})$
Conflict news:	$\mathsf{Actor} \wedge (\Diamond_+ \mathsf{PolActor} \vee \Diamond_- \mathsf{PolActor})$
Issue news:	$\mathsf{PolActor} \wedge (\Diamond_+ \mathsf{Issue} \vee \Diamond_- \mathsf{Issue})$

Concept	until interview	until loc. elections	until murder	after murder	total	n	*Orig.*
Conflict	40%	44%	46%	57%	45%	4,613	*49%*
Horserace	14%	13%	19%	24%	17%	1,721	*20%*
Issue	47%	42%	35%	19%	39%	3,968	*31%*

Table 1
Relative attention to news types

4.1 Modal logic: General political frames

To measure the general tone of the campaign news, three modal logic queries were used to discover the three general news frames [2] used for the original study. Table 1 shows the relative attention for these frames, i.e. the number of Kripke structures in which there was a world in which the corresponding formula was true. The total percentages are compared to the percentages published in (9, table 2.2), which we restate in the last column of the table for convenience.

4.2 $H(@)$: Issue positions of politicians and parties

To discover the average position of politicians on one of the key issues, immigration, two $H(@)$ queries were used. Note that this requires interpreting the hybrid Kripke structures as we want to know the names of the worlds (i.e. politicians) in which the formula holds [3] . Table 2 reports the average direction of the position on immigration per party per period, i.e. $(anti - pro)/(anti + pro)$, compared to table 4.3 from (9).

Interesting to see is the tempering of Fortuyn's position in the period between the local elections and his assassination, also reported in figure 4.1 of the original study. Note also the 'jerk to the right' by both incumbent parties (VVD and PvdA) after the interview while the CDA actually becomes more positive about immigration.

4.3 $H(@, \downarrow)$ with background knowledge: Internal dissent

To discover different forms of internal strife, three $H(@, \downarrow)$ formulas were used on the combined media-data and background knowledge. In these formulas

[2] In communication science, *framing* indicates the way an actor or issue is depicted (or *framed*) in the news.
[3] Note that to bind to the parties rather than the politicians, we would have to use the inverse membership relation , defined as $\Diamond_i^{inv}\varphi \equiv \exists x.(\Diamond_i x \wedge \varphi)$, which requires $H(@, \downarrow)$

Pro-immigration: $\mathsf{PolActor} \wedge \Diamond_{+}\mathsf{immigration}$
Anti-immigration: $\mathsf{PolActor} \wedge \Diamond_{-}\mathsf{immigration}$

Party	until interview	until loc. elections	until elections	total	(n)	*Original*
VVD	-0.42	-0.50	-0.76	-0.53	(60)	*-0.66*
PvdA	0.29	-0.45	-0.11	-0.07	(97)	*-0.11*
CDA	-0.71	-0.56	0.00	-0.57	(41)	*-0.66*
Fortuyn	-1.00	-1.00	-0.48	-0.62	(104)	*-0.52*

Table 2
Average position on immigration

$\Diamond_m$ represents the memberOf relation between a party and a person. Table 3 gives the total number of occurrences of these three frames, as summarised in figure 5.1 from (9), the *support−criticism* per week is also given and compared to the original number. The difference in scale is a result of the counting of nodes (parties) rather than edges as explained above. An interesting result is that no party seems to be able to have internal discussion without internal (personal) criticism, although the PvdA shows in the last period that the internal discussion need not keep step with the criticism.

Internal criticism: $\downarrow x.\mathsf{Party} \wedge (\Diamond_m^{inv}\Diamond_{-}(x \vee \Diamond_m x) \vee \Diamond_{-}(x \vee \Diamond_m x))$
Internal support: $\downarrow x.\mathsf{Party} \wedge (\Diamond_m^{inv}\Diamond_{+}(x \vee \Diamond_m x) \vee \Diamond_{+}(x \vee \Diamond_m x))$
Internal disagreement: $\downarrow x.\mathsf{Party} \wedge (\Diamond_m\Diamond_{+}(\mathsf{Issue} \wedge \downarrow y.(@_x\Diamond_m\Diamond_{-}y)))$

Party	until interview					until elections					until murder				
	disa	crit	supp	diff	*orig*	disa	crit	supp	diff	*orig*	disa	crit	supp	diff	*orig*
PvdA	59	64	42	-1.9	*-3*	9	12	14	0.6	*1*	46	110	76	-4.0	*-8*
VVD	39	55	66	1.0	*3*	11	6	16	2.8	*5*	55	66	70	0.5	*6*
CDA	7	6	7	0.1	*0*	0	8	15	2.0	*4*	5	7	9	0.2	*0*
LPF	3	1	3	0.2	*0*	7	4	11	2.0	*4*	34	28	40	1.4	*1*

Table 3
Internal Criticism and Debate

5 Conclusion

In this paper we describe first attempts at using modal logics, with their well studied formalisms and tools in media studies. As a proof of concept, we reproduce relevant results of a case study about the Dutch election campaign of 2002. We do this by formalising newspaper articles as Kripke structures, and by using an existing hybrid model checker to search for instances of politically interesting patterns. This approach has some nice properties: we can use an existing logic with well-studied semantics and out-of-the-box tools. We can do this, because of the close relation of the social scientist's intermediate NET

representation with Kripke structures. As a result, we can offer explicit formal semantics, easy integration of background knowledge, and an expressive query language, as well as, with the hybrid model checker, a generic tool for querying.

5.1 Relevance to Communication Science

Using this method has three advantages to communication sciences. First, it allows for an easy way to formally describe the concepts used in a study. This can make a study more transparent and hence more repeatable.

Second, by making the investigated concepts and domain knowledge formal and explicit, the amount of specific code or SPSS syntax that has to be written to conduct an analysis is greatly reduced. In fact, apart from the general code to translate NET articles to Kripke structures and to invoke the model checker on those structures with the queries, no code or syntax was written at all for this study.

Finally, it allows the separation of the domain specific knowledge (in the ontology) from the definition of the concepts/frames. This makes it possible to define and maintain the domain ontology without knowing the specific queries that will be asked and, more importantly, to formulate the concepts in general enough terms to compare concepts in different domains. The current queries could also be used on the Dutch election campaigns of 1994 and 1998 or even the American presidential campaigns.

Of course, whether the findings of one study, or even the concepts used in those findings, are useful in different circumstances is another question, one that is crucial for many studies: How far can the proposed concepts and theories be generalised? The method described here can be a tool to help investigate such questions, thus allowing for more systematic meta-research and theory-building.

5.2 Shortcomings, limitations and challenges

Obviously, the streamlining of the analysis process into this particular modal logic formalism comes at a price. The reduction of querying NET graphs with background knowledge from complex reasoning to model checking is only possible because of two relatively strong assumptions: first, we assume complete knowledge about the actors and issues (at least w.r.t. the propositions in our domain), and we restrict our background knowledge to very simple "unfoldable" axioms. There has recently been a surge in interest in simple terminologies to reduce the complexity of reasoning (for example (4)), and we would claim that it should be reasonably simple to extend our approach to more expressive ontologies.

To simplify this presentation, we did not combine the semantics of the NET relations into the modal reasoning, such as transitivity or combination of arrows. Instead we applied explicit constraint propagation before translating to Kripke structures. This, however, is somehow inconsistent with our unifying

approach taken otherwise. Another limitation is our failure to capture the quantitative aspects of NET representations. In part, we avoid the issue by considering individual documents rather than aggregated NET graphs. Thus, the statistical variance is expressed in the results of the analysis, rather than explicitly in the representation itself. Nevertheless, a certain level of approximation and fuzziness is lost in the process.

Another strong restriction of our current approach is the lack of any temporal or epistemic dimension: queries for *changes of mind over time* or *reactions on someone*, let alone *differences between newspapers* cannot be expressed. Nevertheless, we take this shortcoming as a challenge for future work, rather than a limitation, as there is extensive work on multi-dimensional modal logics to start with (just to mention (8)).

References

[1] Blackburn, P., M. de Rijke and Y. Venema, "Modal Logic," Cambridge University Press, 2001.

[2] Blackburn, P. and J. Seligman, *What are hybrid languages?*, Advances in Modal Logic **1**, CSLI Publications, Stanford University, 1998 pp. 41–62.

[3] Bryant, J. and D. Zillman, editors, "Media Effects: Advances in Theory and Research," Lawrence Erlbaum, Mahwah, NJ, 2002.

[4] Calvanese, D., G. De Giacomo, D. Lembo, M. Lenzerini and R. Rosati, *Dl-lite: Tractable description logics for ontologies*, in: *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, 2005.

[5] Carley, K., *Content analysis*, in: R. Asher, editor, *The Encyclopedia of Language and Linguistics*, Pergamon, Edinburgh, 1990 .

[6] Dragone, L., *Hybrid logics model checker*, C implementation published on http://www.luigidragone.com/hlmc/ (2005-07-06) (2005).

[7] Franceschet, M. and M. de Rijke, *Model checking for hybrid logics (with an application to semistructured data)*, Journal of Applied Logic (2005).

[8] Gabbay, D. M., A. Kurucz, F. Wolter and M. Zakharyaschev, "Many-Dimensional Modal Logics: Theory and Applications," Elsevier, 2003.

[9] Kleinnijenhuis, J., D. Oegema, J. de Ridder, A. van Hoof and R. Vliegenthart, "De puinopen in het nieuws," Communicatie Dossier **22**, Kluwer, Alpen aan de Rijn (Netherlands), 2003.

[10] Krippendorff, K., "Content Analysis: An Introduction to Its Methodology (second edition)," Sage Publications, 2004.

[11] McCombs, M. E. and D. L. Shaw, *The agenda-setting function of mass media*, Public Opinion Quarterly **36** (1972), pp. 176–187.

[12] Roberts, C., editor, "Text Analysis for the Social Sciences; Methods for

Drawing Statistical Inferences from Texts and Transcripts," Lawrence Erlbaum Associate, Mahwah, New Jersey, 1997.

[13] Van Atteveldt, W., N. Ruigrok and J. Kleinnijenhuis, *Associative framing: A unified method for measuring media frames and the media agenda*, in: *Submitted to the 56th Annual Conference of the International Communication Association (ICA2006), Dresden, Germany (June 19 – 23, 2006)*, forthcoming.

[14] Wiebe, J., T. Wilson, R. Bruce, M. Bell and M. Martin, *Learning subjective language*, Computational Linguistics **30 (3)** (2004).

[15] *Wikipedia, the free encyclopedia*, URL://www.wikipedia.org (2005).

Cut–Free Axiomatizations for Stratified Modal Fixed Point Logic

Gerhard Jäger, Mathis Kretz*, and Thomas Studer

Institut für Informatik und angewandte Mathematik, Universität Bern,
Neubrückstrasse 10, CH–3012 Bern, Switzerland
{jaeger, kretz, tstuder}@iam.unibe.ch

Abstract. We present an infinitary and a finitary cut–free axiomatization for a fragment of the modal μ–calculus in which nesting of fixed points is restricted to non–interleaving occurrences. In this study we prove soundness and completeness of both axiomatizations. Completeness is established by constructing a canonical countermodel to any non–provable formula using an extension of the method of saturated sequents. Soundness of the finitary axiomatization is a consequence of the small model property and well–known results about monotone operators while completeness follows from the corresponding result for the infinitary case.

1 Introduction

Stratified modal fixed point logic, henceforth abbreviated as SFL, is an extension of standard multi–modal logic with syntactic constructs for representing least and greatest fixed points of stratified positive formulae. As such SFL is contained in Kozen's well known modal μ–calculus which – along with many of its fragments – is an important tool for the specification and verification of properties of programs. It has been studied extensively, for example, with respect to model–checking. Classical studies in this direction include Kozen [11], Streett and Emerson [14], Winskel [17] and Stirling and Walker [13].

Interesting efforts have also been undertaken to obtain sound and complete deductive systems for the modal μ–calculus. Completeness of the (trivially sound) axiomatization proposed by Kozen [11] was addressed by Walukiewicz [15], making use of some deep results from the theory of automata on infinite words. While certainly being interesting from a foundational point of view the proposed deductive systems have so far left one important question largely unaddressed, namely the possibility of obtaining cut–free axiomatizations. All previously proposed complete deductive systems for the modal μ–calculus crucially include a cut–rule and semantic or syntactic cut–elimination procedures were not known.

In this paper we present an infinitary and a closely related finitary cut–free axiomatization of SFL and show that these axiomatizations are sound and complete with respect to the standard Kripke semantics. The language of SFL captures –

* Research partly supported by the Swiss National Science Foundation

as we shall see later – many important logics such as PDL, CTL and the logic of common knowledge. It is stratified in the following sense: Consider a formula $\mu\mathsf{X}.\mathcal{A}[\mathsf{X}]$, where $\mathcal{A}[\mathsf{X}]$ is positive in the variable X. $\mathcal{A}[\mathsf{X}]$ may contain a subformula $\mu\mathsf{Y}.\mathcal{B}[\mathsf{Y}]$ and $\nu\mathsf{Y}.\mathcal{B}[\mathsf{Y}]$ only if X does not appear free in $\nu\mathsf{Y}.\mathcal{B}[\mathsf{Y}]$. This allows us to compute the meaning of $\mu\mathsf{Y}.\mathcal{B}[\mathsf{Y}]$ and $\nu\mathsf{Y}.\mathcal{B}[\mathsf{Y}]$ and then use it to determine the meaning of $\mu\mathsf{X}.\mathcal{A}[\mathsf{X}]$. Stratification guarantees that inner fixed points do not depend on the outer ones. Hence it is possible to determine the meaning of any formula by a simple induction on the levels of its fixed points and its complexity.

This is not possible when interleaving of fixed points is allowed. Consider the formula of the form $\mu\mathsf{X}.\nu\mathsf{Y}.\mathcal{A}[\mathsf{X},\mathsf{Y}]$. Here the meaning of the inner fixed point $\nu\mathsf{Y}.\mathcal{A}[\mathsf{X},\mathsf{Y}]$ depends on the value assigned to X by the interpretation of the outer fixed point $\mu\mathsf{X}.\nu\mathsf{Y}.\mathcal{A}[\mathsf{X},\mathsf{Y}]$ which in turn depends on $\nu\mathsf{Y}.\mathcal{A}[\mathsf{X},\mathsf{Y}]$. Hence interleaving has the effect that the meaning of nested fixed points cannot be determined one after another, but has to be treated in a more complicated way.

Turning to our two deductive systems for SFL we show (i) completeness of the infinitary version $\mathsf{T}^{\omega}_{\mathsf{SFL}}$ and (ii) the soundness of the finitary system $\mathsf{T}_{\mathsf{SFL}}$. Furthermore, it will be obvious that everything provable in $\mathsf{T}^{\omega}_{\mathsf{SFL}}$ is also provable in $\mathsf{T}_{\mathsf{SFL}}$ and, consequently, $\mathsf{T}^{\omega}_{\mathsf{SFL}}$ as well as $\mathsf{T}_{\mathsf{SFL}}$ are sound and complete.

The completeness of $\mathsf{T}^{\omega}_{\mathsf{SFL}}$ is proved by extending a method also used in Alberucci and Jäger [1]. For establishing the soundness of the finitary system $\mathsf{T}_{\mathsf{SFL}}$, we heavily rely on the small model property of the modal μ–calculus; see [14, 5, 9, 16]. Generalizing an idea of Jäger, Kretz and Studer [10], we can exploit the fact that with respect to provability only finitely many stages of greatest fixed points are relevant.

The focus of this paper is on proof-theoretic aspects of an important subsystem of the modal μ–calculus and the question whether there are cut–free formalizations. We do *not* claim that the systems presented in this article are a direct basis for implementations which perform better, for example, with respect to average case behavior. However, we think that it is important to know that finite cut–free calculi do exist so that it makes sense to search for cut–free frameworks whose structural proof theory is more developed than that of $\mathsf{T}_{\mathsf{SFL}}$. Furthermore, the proof systems studied in this paper allow direct and logic–oriented completeness proofs, a fact which is interesting in its own right.

2 Monotone Operators

To facilitate the introduction of SFL we will first briefly review some basic notions and results from the theory of monotone operators; for all details we refer, for example, to Moschovakis [12].

Definition 1 (Monotone Operators). *Let A be a set. A function F is called an operator on A if $F : \mathcal{P}(A) \longrightarrow \mathcal{P}(A)$. F is called monotone if for all $B, C \subset A$ such that $B \subset C$ we have $F(B) \subset F(C)$.*

An important notion which we will require in various forms is that of a (potentially transfinite) iteration of a given operator. There are two natural ways of defining iterations. Firstly, we may start from the empty set and collect up all elements which are possibly added by repeated applications of a given operator. Secondly, we may start with the whole domain and throw away any elements which are possibly removed by a repeated application of the operator. Both these possibilities are reflected in the next definition.

Definition 2 (Iterations). *Let A be a set, F a monotone operator on A and α an ordinal. Define the sets I_F^α, I_F, $I_F^{<\alpha}$ J_F^α, J_F and $J_F^{<\alpha}$ as follows:*

$$\begin{array}{lll} I_F^{<\alpha} = \bigcup_{\beta<\alpha} I_F^\beta, & I_F^\alpha = F(I_F^{<\alpha}), & I_F = \bigcup_\beta I_F^\beta. \\ J_F^{<\alpha} = \bigcap_{\beta<\alpha} J_F^\beta, & J_F^\alpha = F(J_F^{<\alpha}), & J_F = \bigcap_\beta J_F^\beta. \end{array}$$

The next two theorems state that the sequences of sets determined by iterating a monotone operator transfinitely many times in the two ways described in Definition 2 always converge to the least and greatest fixed point of this operator and that the number of iterations at which convergence is reached is bounded by the cardinality of the domain.

Theorem 1 (Least Fixed Point). *Let A be a set and F a monotone operator on A. Then the following statements hold:*

(i) *For any ordinals α, β, if $\beta \leq \alpha$, then $I_F^\beta \subset I_F^\alpha$.*
(ii) *There exists an ordinal α such that $\alpha < |A|^+$ and $I_F = I_F^\alpha = I_F^{<\alpha}$.*
(iii) $F(I_F) = I_F = \bigcap\{B : F(B) \subset B\} = \bigcap\{B : F(B) = B\}$.

A proof of this theorem is given in Moschovakis [12]. This takes care of the least fixed point case. The analogous situation also holds for greatest fixed points and their approximations.

Theorem 2 (Greatest Fixed Point). *Let A be a set and F a monotone operator on A. Then the following statements hold:*

(i) *For any ordinals α, β, if $\beta \leq \alpha$, then $J_F^\alpha \subset J_F^\beta$.*
(ii) *There exists an ordinal α such that $\alpha < |A|^+$ and $J_F = J_F^\alpha = J_F^{<\alpha}$.*
(iii) $F(J_F) = J_F = \bigcup\{B : B \subset F(B)\} = \bigcup\{B : B = F(B)\}$.

Finally, for reasons of duality we shall also mention the following statement.

Theorem 3 (Fixed Point Duality). *Let F be a monotone operator on a set A. Then the operator $G(X) := A \setminus F(A \setminus X)$ is monotone and the following properties hold for all $B \subset A$:*

(i) *If $F(B) = B$, then $G(A \setminus B) = A \setminus B$.*
(ii) *If $G(B) = B$, then $F(A \setminus B) = A \setminus B$.*
(iii) *$I_G = A \setminus J_F$ and $J_G = A \setminus I_F$.*

3 Stratified Modal Fixed Point Logic

Starting off from the usual language of multi-modal logic, we introduce the language of SFL in a level-by-level fashion, beginning at level 0, at which we allow only formulae of modal logic possibly containing a fresh propositional variable X and its negation $\sim\mathsf{X}$. For all formulae $\mathcal{A}$ which are positive in X we then add constants $P_{\mathcal{A}}$ and $Q_{\mathcal{A}}$ for the least and greatest fixed point of the associated operator. At level 1 we allow formulae of modal logic possibly containing X and $\sim\mathsf{X}$ as well as any constant $P_{\mathcal{A}}$ or $Q_{\mathcal{A}}$ from level 0. Again, for those formulae $\mathcal{B}$ of level 1, which are positive in X we add new constants $P_{\mathcal{B}}$ and $Q_{\mathcal{B}}$ as before and iterate this procedure inductively, thus obtaining levels $2, 3, 4$ and so on. Each formula A of the language is therefore assigned a level in a natural way, namely the minimal level of this construction at which A appears. At each level of the language we also define the negation $\neg A$ for an arbitrary formula A using De Morgan's laws, the law of double negation and the law of fixed point duality stated in Theorem 3.

Definition 3 (Language, Level, Length). *Let $\Phi = \{\mathsf{p}, \sim\mathsf{p}, \mathsf{q}, \sim\mathsf{q}, \mathsf{r}, \sim\mathsf{r}, \ldots\}$ be a countable set of atomic propositions, $\mathsf{V} = \{\mathsf{X}, \sim\mathsf{X}\}$ a set containing one variable and its negation, $\mathsf{T} = \{\top, \bot\}$ a set containing symbols for truth and falsehood and M a set of indices.*

1. *Define $\mathcal{L}_0$ as being the least superset of $\Phi \cup \mathsf{V} \cup \mathsf{T}$ closed under $\wedge, \vee, \Box_i, \Diamond_i$ where $i \in \mathsf{M}$. Given a formula $A \in \mathcal{L}_0$, we inductively define $\neg A$ by:*

$$\begin{aligned}
&\neg\mathsf{p} := \sim\mathsf{p}, \quad \neg\mathsf{X} := \sim\mathsf{X}, \quad \neg\sim\mathsf{p} := \mathsf{p}, \quad \neg\sim\mathsf{X} := \mathsf{X}, \quad \neg\top := \bot, \quad \neg\bot := \top,\\
&\neg(B \wedge C) := \neg B \vee \neg C, \quad \neg(B \vee C) := \neg B \wedge \neg C,\\
&\neg\Box_i B := \Diamond_i \neg B, \quad \neg\Diamond_i B := \Box_i \neg B.
\end{aligned}$$

2. *A formula $A \in \mathcal{L}_0$ is called X-positive if $\sim\mathsf{X}$ does not occur in A. In the following, X-positive formulae will be denoted by symbols $\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$ (possibly with primes and subscripts). Given formulae $\mathcal{A}$ and B of $\mathcal{L}_0$ where $\mathcal{A}$ is X-positive we write $\mathcal{A}[B]$ for the formula which is obtained by replacing every occurrence of X in $\mathcal{A}$ by B. We define the dual $\overline{\mathcal{A}}$ of an X-positive formula $\mathcal{A}$ as $\overline{\mathcal{A}} = \neg(\mathcal{A}[\sim\mathsf{X}])$. Furthermore, define the sets*

$$L_0 := \{P_{\mathcal{A}} : \mathcal{A} \in \mathcal{L}_0 \ \mathsf{X}\text{-positive}\} \text{ and } G_0 := \{Q_{\mathcal{A}} : \mathcal{A} \in \mathcal{L}_0 \ \mathsf{X}\text{-positive}\}.$$

3. *Define $\mathcal{L}_{k+1}$ as the least superset of $\Phi \cup \mathsf{V} \cup \mathsf{T} \cup L_k \cup G_k$ closed under the symbols $\wedge, \vee, \Box_i, \Diamond_i$ where $i \in \mathsf{M}$. Again a formula $\mathcal{A} \in \mathcal{L}_{k+1}$ is called X-positive if $\sim\mathsf{X}$ does not occur in $\mathcal{A}$. For any formulae $A \in \mathcal{L}_{k+1}$ define $\neg A$ as before and adding the clauses $\neg P_{\mathcal{A}} := Q_{\overline{\mathcal{A}}}$ and $\neg Q_{\mathcal{A}} := P_{\overline{\mathcal{A}}}$. For X-positive formulae $\mathcal{A} \in \mathcal{L}_{k+1}$ we also define $\overline{\mathcal{A}}$ as before. Similar to the base case we also define sets of fixed point constants*

$$L_{k+1} := \{P_{\mathcal{A}} : \mathcal{A} \in \mathcal{L}_k \ \mathsf{X}\text{-positive}\} \text{ and } G_{k+1} := \{Q_{\mathcal{A}} : \mathcal{A} \in \mathcal{L}_k \ \mathsf{X}\text{-positive}\}.$$

4. *Define the language $\mathcal{L}$ by setting $\mathcal{L} = \bigcup_{k \in \omega} \mathcal{L}_k$.*

5. *For every formula* $A \in \mathcal{L}$ *define* level(A) *to be the least* k *so that* $A \in \mathcal{L}_k$.
6. *The length* $|A|$ *of an* $A \in \mathcal{L}$ *is simply the number of symbols occurring in* A *with the proviso that fixed point constants* $P_{\mathcal{A}}$ *and* $Q_{\mathcal{A}}$ *count (recursively) as* $|\mathcal{A}| + 1$ *many symbols.*

Turning to the semantics of $\mathcal{L}$, a Kripke structure, as usual, is a triple $\mathsf{K} = (S, R, \pi)$ where S is a non–empty set, R a mapping from M to $\mathcal{P}(S \times S)$ and π a mapping from $\Phi \cup \mathsf{V}$ to $\mathcal{P}(S)$; we also require that $\pi(\sim\mathsf{X}) = S \setminus \pi(\mathsf{X})$ and $\pi(\sim\mathsf{p}) = S \setminus \pi(\mathsf{p})$ for all $\mathsf{p} \in \Phi$. The function R assigns an accessibility relation to each $i \in \mathsf{M}$ where we write R_i for the relation $R(i)$. Furthermore, given a set $T \subset S$ we define $\mathsf{K}[\mathsf{X} := T]$ to be the the Kripke structure (S, R, π'), where π' maps X to T, $\sim\mathsf{X}$ to $S \setminus T$ and otherwise agrees with π.

Given a Kripke structure K we now define the denotations of the formulae of $\mathcal{L}$ by main induction on their levels and side induction on their lengths.

Definition 4 (Denotation). *Let* $\mathsf{K} = (S, R, \pi)$ *be a Kripke structure. For every* $A \in \mathcal{L}_k$ *we define the set* $\|A\|_{\mathsf{K}} \subset S$ *inductively as follows:*

$$\|P\|_{\mathsf{K}} := \pi(P) \text{ for all } P \in \Phi \cup \mathsf{V}, \quad \|\top\|_{\mathsf{K}} := S, \quad \|\bot\|_{\mathsf{K}} := \emptyset,$$
$$\|B \wedge C\|_{\mathsf{K}} := \|B\|_{\mathsf{K}} \cap \|C\|_{\mathsf{K}}, \quad \|B \vee C\|_{\mathsf{K}} := \|B\|_{\mathsf{K}} \cup \|C\|_{\mathsf{K}},$$
$$\|\Box_i B\|_{\mathsf{K}} := \{w \in S : v \in \|B\|_{\mathsf{K}} \text{ for all } v \text{ such that } wR_iv\},$$
$$\|\Diamond_i B\|_{\mathsf{K}} := \{w \in S : v \in \|B\|_{\mathsf{K}} \text{ for some } v \text{ such that } wR_iv\}.$$

For every $P_{\mathcal{A}}$ *and* $Q_{\mathcal{A}}$ *we define*

$$\|P_{\mathcal{A}}\|_{\mathsf{K}} := \bigcap\{T \subset S : T \supset F^{\mathsf{K}}_{\mathcal{A}}(T)\} \text{ and } \|Q_{\mathcal{A}}\|_{\mathsf{K}} := \bigcup\{T \subset S : T \subset F^{\mathsf{K}}_{\mathcal{A}}(T)\}$$

where $F^{\mathsf{K}}_{\mathcal{A}}$ *is the operator on* S *given by* $F^{\mathsf{K}}_{\mathcal{A}}(T) := \|\mathcal{A}\|_{\mathsf{K}[\mathsf{X}:=T]}$ *for every subset* T *of* S.

Obviously by the main induction hypothesis the operator $F^{\mathsf{K}}_{\mathcal{A}}$ is defined and monotone and $\|P_{\mathcal{A}}\|_{\mathsf{K}}$ and $\|Q_{\mathcal{A}}\|_{\mathsf{K}}$ are the least and greatest fixed points of $F^{\mathsf{K}}_{\mathcal{A}}$ respectively.

Definition 5 (Satisfaction and Validity). *Let* $\mathsf{K} = (S, R, \pi)$ *be a Kripke structure. We say a formula* $A \in \mathcal{L}$ *is satisfied in* K *if* $\|A\|_{\mathsf{K}} \neq \emptyset$ *and valid in* K *if* $\|A\|_{\mathsf{K}} = S$. *We say* A *is satisfiable if there exists a Kripke structures in which* A *is satisfied. Furthermore, we say that* A *is valid if it is valid in all Kripke structures.*

The next definition introduces formulae describing the finite iterations of monotone operators, both from below and above. We also define the stages of such inductive definitions over a given Kripke structure.

Definition 6 (Iterations). *Let* $\mathcal{A}$ *be an* X*-positive formula.*

1. *For every $k \in \omega$ define the formulae $Q^k_{\mathcal{A}}$ and $P^k_{\mathcal{A}}$ inductively as follows:*

$$P^0_{\mathcal{A}} := \bot, \quad P^{k+1}_{\mathcal{A}} := \mathcal{A}[P^k_{\mathcal{A}}], \qquad Q^0_{\mathcal{A}} := \top, \quad Q^{k+1}_{\mathcal{A}} := \mathcal{A}[Q^k_{\mathcal{A}}].$$

2. *Let $\mathsf{K} = (S, R, \pi)$ be a Kripke structure. For every ordinal α define the subsets $I^{<\alpha}_{\mathcal{A},\mathsf{K}}$, $I^{\alpha}_{\mathcal{A},\mathsf{K}}$ $J^{<\alpha}_{\mathcal{A},\mathsf{K}}$ and $J^{\alpha}_{\mathcal{A},\mathsf{K}}$ of S as follows:*

$$I^{<\alpha}_{\mathcal{A},\mathsf{K}} := I^{<\alpha}_{F^{\mathsf{K}}_{\mathcal{A}}}, \qquad I^{\alpha}_{\mathcal{A},\mathsf{K}} := I^{\alpha}_{F^{\mathsf{K}}_{\mathcal{A}}}, \qquad J^{<\alpha}_{\mathcal{A},\mathsf{K}} := J^{<\alpha}_{F^{\mathsf{K}}_{\mathcal{A}}}, \qquad J^{\alpha}_{\mathcal{A},\mathsf{K}} := J^{\alpha}_{F^{\mathsf{K}}_{\mathcal{A}}}.$$

Remark 1. We note two facts, which are evident from Definition 6 and which will be used several times in the subsequent argument. Let $\mathcal{A}$ be a formula of $\mathcal{L}$ which is X-positive. Then for all natural numbers k we have $\mathrm{level}(Q^k_{\mathcal{A}}) < \mathrm{level}(Q_{\mathcal{A}})$ and $\mathrm{level}(P^k_{\mathcal{A}}) < \mathrm{level}(P_{\mathcal{A}})$ and, furthermore, for all Kripke structures K we have $\|P^k_{\mathcal{A}}\|_{\mathsf{K}} = I^{<k}_{\mathcal{A},\mathsf{K}}$ and $\|Q^k_{\mathcal{A}}\|_{\mathsf{K}} = J^{<k}_{\mathcal{A},\mathsf{K}}$.

Our language $\mathcal{L}$ is built up in layers and at each layer we add a set of fixed point constants for certain formulae of the layer below. The following definition is so that the rank of a formula at a higher level is strictly greater than the rank of any formula in a level below.

Definition 7 (Rank). *The rank* $\mathrm{rk}(A)$ *of a formula $A \in \mathcal{L}$ is an ordinal defined inductively as follows:*

1. *If A is an element of $\Phi \cup \mathsf{V} \cup \mathsf{T}$, then* $\mathrm{rk}(A) := 0$.
2. *If A is a fixed point constant $P_{\mathcal{A}}$ or $Q_{\mathcal{A}}$ and* $\mathrm{level}(A) = n$, *then* $\mathrm{rk}(A) := \omega n$.
3. *If A is a formula $B \wedge C$ or $B \vee C$, then* $\mathrm{rk}(A) := \max(\mathrm{rk}(B), \mathrm{rk}(C)) + 1$.
4. *If A is a formula $\Box_i B$ or $\Diamond_i B$ for some $i \in \mathsf{M}$, then* $\mathrm{rk}(A) := \mathrm{rk}(B) + 1$.

The following lemma summarizes some important properties of the rank function. Its proof is routine and left to the reader.

Lemma 1. *For all formulae $A, B \in \mathcal{L}$ the following statements hold:*

(i) $\omega \cdot \mathrm{level}(A) \leq \mathrm{rk}(A) < (\omega \cdot \mathrm{level}(A)) + \omega$.
(ii) $\mathrm{rk}(A) = \mathrm{rk}(\neg A)$.
(iii) $\mathrm{rk}(A), \mathrm{rk}(B) < \mathrm{rk}(A \wedge B), \mathrm{rk}(A \vee B)$.
(iv) $\mathrm{rk}(A) < \mathrm{rk}(\Box_i A), \mathrm{rk}(\Diamond_i A)$ *for all $i \in \mathsf{M}$.*

The right hand inequality of the following Lemma is a straightforward consequence of Remark 1 and Lemma 1, the left hand one is an induction on k.

Lemma 2. *For all fixed point constants $Q_{\mathcal{A}}$ and all $k \in \omega$ we have*

$$\mathrm{rk}(Q^k_{\mathcal{A}}) \leq \mathrm{rk}(Q^{k+1}_{\mathcal{A}}) < \mathrm{rk}(Q_{\mathcal{A}}).$$

The preceding considerations show that the rank of a formula A as given in Definition 7 allow us to prove certain claims – for example the next one – by a simple induction on $\mathrm{rk}(A)$.

Lemma 3. *For all formulae A of $\mathcal{L}$ and all Kripke structures $\mathsf{K} = (S, R, \pi)$ we have $\|\neg A\|_{\mathsf{K}} = S \setminus \|A\|_{\mathsf{K}}$.*

4 A Hilbert System for SFL

In this section we briefly introduce the Hilbert system $\mathsf{H}_{\mathsf{SFL}}$ for stratified modal fixed point logic. We mention $\mathsf{H}_{\mathsf{SFL}}$ because it is simple and easily accessible from an intuitive point of view. The system is basically Kozen's axiomatization of the modal μ–calculus [11] adapted to our more restrictive setting. It consists of the standard axioms and rules for multi–modal logic plus axioms and rules for the fixed point constants. Given X-positive formulae $\mathcal{A}$ the additional axioms express the fact that the constants $P_{\mathcal{A}}$ stand for fixed points and the additional induction rules state that these fixed points are minimal. Indeed, as we are about to see we only require additional axioms and rules for least fixed points. Their counterparts with respect to greatest fixed points may be derived due to the syntactic duality of least and greatest fixed points in our language.

Definition 8 (The System $\mathsf{H}_{\mathsf{SFL}}$). *The system $\mathsf{H}_{\mathsf{SFL}}$ is defined by adding the following axioms and rules to any suitable Hilbert–style axiomatization for a multi–modal formulation of the logic K:*

Closure axioms *For every X-positive formula $\mathcal{A}$:*

$$\mathcal{A}[P_{\mathcal{A}}] \to P_{\mathcal{A}}$$

Induction rules *For every X-positive formula $\mathcal{A}$ and every formula B:*

$$\frac{\mathcal{A}[B] \to B}{P_{\mathcal{A}} \to B}$$

It is easily checked that for greatest fixed points the duals of the closure axioms and induction rules are derivable in $\mathsf{H}_{\mathsf{SFL}}$. Basically all we need is the fixed point duality formulated in Theorem 3.

Lemma 4. *The system $\mathsf{H}_{\mathsf{SFL}}$ derives the formula $Q_{\mathcal{A}} \to \mathcal{A}[Q_{\mathcal{A}}]$ and the rule:*

$$\frac{B \to \mathcal{A}[B]}{B \to Q_{\mathcal{A}}}$$

We may use the system $\mathsf{H}_{\mathsf{SFL}}$ to gain some intuition about the expressive strength of SFL. To this end we will briefly discuss three notable fragments of SFL, namely logic of common knowledge, computational tree logic CTL and propositional dynamic logic PDL. Logic of common knowledge [6] is a multi–modal logic which can be used to talk about certain epistemic situations among a group of agents. The index set M is interpreted as standing for a finite set of agents and a modal formula $\Box_i A$ is taken to mean that agent i knows the statement A. If A is known by all agents in M, we write $\mathsf{E}A$ which, formally speaking, is just an abbreviation for the conjunction $\bigwedge_{i \in \mathsf{M}} \Box_i A$. If A is common knowledge among all agents – all agents know A and all agents know that all agents know A and so on ad infinitum – then we write $\mathsf{C}A$. More formally, the logic of common knowledge can be axiomatized as a multi–modal version of K plus the following axioms and rules to treat the common knowledge operator C:

Closure axioms For every formula A:

$$\mathsf{C}A \rightarrow \mathsf{E}(A \wedge \mathsf{C}A)$$

Induction rules For every formula A and B:

$$\frac{B \rightarrow \mathsf{E}(A \wedge B)}{B \rightarrow \mathsf{C}A}$$

It can easily be seen that the logic of common knowledge is a fragment of SFL by defining a syntactic embedding from the former into the latter: we translate atomic, propositional and modal formulae as themselves and a formula $\mathsf{C}A$ as $Q_{\mathsf{E}(A^* \wedge \mathsf{X})}$ where A^* denotes the translation of A. By Lemma 4 it is clear that we then obtain the following theorem stating that the translations of the axioms and rules for common knowledge are derivable in $\mathsf{H}_{\mathsf{SFL}}$. Since the rest of the axioms and rules of logic of common knowledge are also a part of $\mathsf{H}_{\mathsf{SFL}}$, this already takes care of the embedding result.

Theorem 4. *For all formulae A and B of logic of common knowledge we have:*

1. $\mathsf{H}_{\mathsf{SFL}}$ *derives* $(\mathsf{C}A \rightarrow \mathsf{E}(A \wedge \mathsf{C}A))^*$.
2. *If* $\mathsf{H}_{\mathsf{SFL}}$ *derives* $(B \rightarrow \mathsf{E}(A \wedge B))^*$, *then* $\mathsf{H}_{\mathsf{SFL}}$ *also derives* $(B \rightarrow \mathsf{C}A)^*$.

The next fragment of SFL we consider is computational tree logic or CTL for short [4]. CTL is based on mono–modal logic and may be used to talk about the set of all possible runs of a system. Using this logic we may express such properties as "in all runs extending from the current state A holds in the next state", written as $\Box A$, or "in some runs extending from the current state A holds in the next state", written as $\Diamond A$. More importantly we may also express behavior which is in a sense unbounded like "in all runs extending from the current state A holds until B is the case", denoted by $\forall(A\mathsf{U}B)$ or "in some runs extending from the current state A holds until B is the case", written as $\exists(A\mathsf{U}B)$. Axiomatically we obtain CTL by extending a suitable axiomatization for K by the following axioms and rules governing the use of the $\forall(A\mathsf{U}B)$ and $\exists(A\mathsf{U}B)$ constructs:

Closure axioms For every formula A and B

(1) $B \vee (A \wedge \Box\forall(A\mathsf{U}B)) \rightarrow \forall(A\mathsf{U}B)$

(2) $B \vee (A \wedge \Diamond\exists(A\mathsf{U}B)) \rightarrow \exists(A\mathsf{U}B)$

Induction rules For all formulae A, B and C

(3) $$\frac{C \rightarrow (\neg B \wedge \Diamond C)}{C \rightarrow \neg\forall(A\mathsf{U}B)}$$

(4) $$\frac{C \rightarrow (\neg B \wedge (A \rightarrow \Box C))}{C \rightarrow \neg\exists(A\mathsf{U}B)}$$

An embedding of CTL into SFL can be obtained again by translating atomic, propositional and modal constructs as themselves and using in this case least fixed point constants to translate formulae of the form $\forall(A\mathsf{U}B)$ and $\exists(A\mathsf{U}B)$. More precisely we translate a formula of the form $\forall(A\mathsf{U}B)$ into $P_{B^* \vee (A^* \wedge \Box\mathsf{X} \wedge \Diamond\top)}$ and a formula of the form $\exists(A\mathsf{U}B)$ into $P_{B^* \vee (A^* \wedge \Diamond\mathsf{X})}$ where in both cases A^* and B^* stand for the translations of A and B respectively. Using this translation and Definition 8 we again immediately get the following theorem which ensures that CTL is embeddable into $\mathsf{H}_{\mathsf{SFL}}$.

Theorem 5. *For all formulae A, B and C of* CTL *we have:*

1. $\mathsf{H}_{\mathsf{SFL}}$ *derives* $[B \vee (A \wedge \Box\forall(A\mathsf{U}B)) \rightarrow \forall(A\mathsf{U}B)]^*$.
2. $\mathsf{H}_{\mathsf{SFL}}$ *derives* $[B \vee (A \wedge \Diamond\exists(A\mathsf{U}B)) \rightarrow \exists(A\mathsf{U}B)]^*$.
3. *If* $\mathsf{H}_{\mathsf{SFL}}$ *derives* $[C \rightarrow (\neg B \wedge \Diamond C)]^*$, *then also* $[C \rightarrow \neg\forall(A\mathsf{U}B)]^*$.
4. *If* $\mathsf{H}_{\mathsf{SFL}}$ *derives* $[C \rightarrow (\neg B \wedge (A \rightarrow \Box C))]^*$, *then also* $[C \rightarrow \neg\exists(A\mathsf{U}B)]^*$.

The last fragment of SFL which we mention is propositional dynamic logic, abbreviated as PDL [7, 8]. It is once again a multi–modal logic this time featuring an infinite set M of indices. The logic is primarily used for reasoning about programs in the following sense: a formula $\Box_i A$ is interpreted as the statement "whenever an i action is executed in the current state, we terminate in a state which satisfies A". Consequently, $\Diamond_i A$ is taken to mean "in the current state it is possible to execute an i action and terminate in a state which satisfies A". Similar to our two previous examples PDL also features constructs for expressing "unbounded" properties. $\Box_i^* A$ is used to state that for any finite iteration of the action i we end up in a state satisfying A. Dually, $\Diamond_i^* A$ states that there exists a finite iteration of i actions after which we end up in a state satisfying A. Again we may axiomatize PDL by taking suitable axioms for multi–modal K and extending them by the following axioms and rules for the $\Box_i^*$ and $\Diamond_i^*$ operators:

Closure axioms For every formula A

(1) $\Box_i^* A \rightarrow (A \wedge \Box_i \Box_i^* A)$
(2) $(A \vee \Diamond_i \Diamond_i^* A) \rightarrow \Diamond_i^* A$

Induction rules For all formulae A, B

(3) $$\frac{B \rightarrow (A \wedge \Box_i B)}{B \rightarrow \Box_i^* A}$$
(4) $$\frac{(A \vee \Diamond_i B) \rightarrow B}{\Diamond_i^* A \rightarrow B}$$

The above axioms and rules suggest a translation of a formula $\Box_i^* A$ as a greatest and $\Diamond_i^* A$ as a least fixed point. More precisely, in order to embed PDL into SFL we translate atomic, propositional and modal formulae as themselves, $\Box_i^* A$ as $Q_{A^* \wedge \Box_i \mathsf{X}}$ and $\Diamond_i^* A$ as $P_{A^* \vee \Diamond_i \mathsf{X}}$ where A^* stands for the translation of A. Using this translation, Definition 8 and Lemma 4 we prove the following embedding theorem.

Theorem 6. *For all formulae A and B of* PDL *we have:*

1. $\mathsf{H}_{\mathsf{SFL}}$ *derives* $[\Box_i^* A \rightarrow (A \wedge \Box_i \Box_i^* A)]^*$.
2. $\mathsf{H}_{\mathsf{SFL}}$ *derives* $[(A \vee \Diamond_i \Diamond_i^* A) \rightarrow \Diamond_i^* A]^*$.
3. *If* $\mathsf{H}_{\mathsf{SFL}}$ *derives* $[B \rightarrow (A \wedge \Box_i B)]^*$, *then* $\mathsf{H}_{\mathsf{SFL}}$ *derives* $[B \rightarrow \Box_i^* A]^*$.
4. *If* $\mathsf{H}_{\mathsf{SFL}}$ *derives* $[(A \vee \Diamond_i B) \rightarrow B]^*$, *then* $\mathsf{H}_{\mathsf{SFL}}$ *derives* $[\Diamond_i^* A \rightarrow B]^*$.

5 The Infinitary Calculus $\mathsf{T}^{\omega}_{\mathsf{SFL}}$

In the following we introduce the system $\mathsf{T}^{\omega}_{\mathsf{SFL}}$ for deriving valid formulae of stratified modal fixed point logic. For ease of presentation we shall restrict the remainder of our account to the case where we are dealing with just one modality, that is to say where M is a singleton set. A generalization of the subsequent arguments to the full multi–modal case is an easy exercise. Consequently, we drop the indices when writing the symbols $\Box$ and $\Diamond$. Furthermore, given a Kripke structure $\mathsf{K} = (S, R, \pi)$ we henceforth treat R as a single relation $R \subset S \times S$.

The calculus $\mathsf{T}^{\omega}_{\mathsf{SFL}}$ is designed in Tait–style, that is to say, using it we derive finite sets of formulae of $\mathcal{L}$. It is infinitary in the sense that in order to apply the rule for a greatest fixed point constant $Q_{\mathcal{A}}$ we must derive infinitely many premises. As a consequence of this, proofs in $\mathsf{T}^{\omega}_{\mathsf{SFL}}$ can become infinite in length. Our calculus will not include a cut rule although such a rule could be added for convenience.

Definition 9 (Sequents). *A sequent is a finite set of formulae of $\mathcal{L}$. Henceforth, unless otherwise stated capital Greek letters $\Gamma, \Delta, \Sigma, \ldots$ (possibly with primes and subscripts) shall be used to denote sequents. Given a formula A we write Γ, A for $\Gamma \cup \{A\}$. By $\bigvee \Gamma$ we denote the formula $((\ldots(A_1 \vee A_2) \vee \ldots) \vee A_n)$ if Γ is the set $\{A_1, \ldots, A_n\}$ or the formula $\bot$ if Γ is empty. Furthermore, by $\Diamond\Gamma$ we denote the sequent obtained by prefixing each formula in Γ by $\Diamond$.*

We are now in a position to state the rules of $\mathsf{T}^{\omega}_{\mathsf{SFL}}$. These rules are to be read in the following way: if all sequents displayed in the premise have already been derived, then the sequent displayed in the conclusion may also be derived. In this sense, rules with an empty premise correspond to sequents which may always be derived, that is to say they are axioms of $\mathsf{T}^{\omega}_{\mathsf{SFL}}$.

Definition 10 (The System $\mathsf{T}^{\omega}_{\mathsf{SFL}}$). *The system $\mathsf{T}^{\omega}_{\mathsf{SFL}}$ is defined by the following set of inference rules:*

$$\frac{}{\Gamma, \mathsf{p}, \sim\mathsf{p}} \quad (ID1) \qquad \frac{}{\Gamma, \mathsf{X}, \sim\mathsf{X}} \quad (ID2) \qquad \frac{}{\Gamma, \top} \quad (ID3)$$

$$\frac{\Gamma, A, B}{\Gamma, A \vee B} \quad (\vee) \qquad \frac{\Gamma, A \quad \Gamma, B}{\Gamma, A \wedge B} \quad (\wedge)$$

$$\frac{\Gamma, A}{\Diamond\Gamma, \Box A, \Sigma} \quad (\Box)$$

$$\frac{\Gamma, \mathcal{A}[P_{\mathcal{A}}]}{\Gamma, P_{\mathcal{A}}} \quad (P_{\mathcal{A}}) \qquad \frac{\Gamma, Q^{k}_{\mathcal{A}} \quad \text{for all } k \in \omega}{\Gamma, Q_{\mathcal{A}}} \quad (Q^{\omega}_{\mathcal{A}})$$

Provability of a sequent Γ in $\mathsf{T}^{\omega}_{\mathsf{SFL}}$ is defined as usual and denoted by $\mathsf{T}^{\omega}_{\mathsf{SFL}} \vdash \Gamma$. It is obvious from the formulation of the system that $\mathsf{T}^{\omega}_{\mathsf{SFL}}$ satisfies weakening in the sense that if Γ is provable and $\Gamma \subset \Delta$, then Δ is also provable.

However, it is not immediately obvious that $\mathsf{T}^{\omega}_{\mathsf{SFL}}$ is sound. Problems might occur in connection with the infinitary rule $(Q^{\omega}_{\mathcal{A}})$ whose premises are exactly the finite stages of greatest fixed points, whereas in arbitrary Kripke structures transfinite stages cannot be ruled out. However, in Section 7 we will prove the soundness of a system $\mathsf{T}_{\mathsf{SFL}}$ which contains $\mathsf{T}^{\omega}_{\mathsf{SFL}}$. Thus we do not elaborate on the soundness proof for $\mathsf{T}^{\omega}_{\mathsf{SFL}}$ here.

From the completeness of $\mathsf{T}^{\omega}_{\mathsf{SFL}}$ which will be shown in Section 6 we can immediately deduce that the Hilbert system $\mathsf{H}_{\mathsf{SFL}}$ is contained in $\mathsf{T}^{\omega}_{\mathsf{SFL}}$. More formally and moving from Hilbert–style to a Tait–style framework, completeness implies the following Theorem

Theorem 7. *For all formulae $\mathcal{A}$ and B of $\mathcal{L}$ where $\mathcal{A}$ is X-positive we have:*

(i) $\mathsf{T}^{\omega}_{\mathsf{SFL}} \vdash \neg\mathcal{A}[P_{\mathcal{A}}], P_{\mathcal{A}}$.
(ii) $\mathsf{T}^{\omega}_{\mathsf{SFL}} \vdash \neg\mathcal{A}[B], B \implies \mathsf{T}^{\omega}_{\mathsf{SFL}} \vdash \neg P_{\mathcal{A}}, B$.

6 A Completeness Proof for $\mathsf{T}^{\omega}_{\mathsf{SFL}}$

We now aim to prove completeness for the system $\mathsf{T}^{\omega}_{\mathsf{SFL}}$, that is the claim that any valid formula of $\mathcal{L}$ must be provable in $\mathsf{T}^{\omega}_{\mathsf{SFL}}$. In order to arrive at this result, we will in fact show its contrapositive. That is we will show that given a non–provable formula A, we may always construct a Kripke structure in which A is not satisfied. Such a formula A can thus never be valid. To show completeness we use an extension of the method of saturated sequents. A sequent is saturated if it is in a sense maximally non–provable. The first major step in proving the completeness of $\mathsf{T}^{\omega}_{\mathsf{SFL}}$ is thus to show that any non–provable sequent can be expanded to a saturated sequent. The second step is then to show that from the set of all saturated sequents we may construct a suitable countermodel for a non–provable formula. We thus first need to state what saturated sequents are.

Definition 11 (Saturated Sequents). *Let k be a natural number.*

1. *A sequent $\Gamma \subset \mathcal{L}$ is called k–presaturated if all of the following properties hold:*
   (i) $\mathsf{T}^{\omega}_{\mathsf{SFL}} \nvdash \Gamma$
   (ii) For all formulae $A \wedge B$ with $\mathrm{level}(A \wedge B) \geq k$*, if $A \wedge B \in \Gamma$, then $A \in \Gamma$ or $B \in \Gamma$.*
   (iii) For all formulae $A \vee B$ with $\mathrm{level}(A \vee B) \geq k$*, if $A \vee B \in \Gamma$, then $A \in \Gamma$ and $B \in \Gamma$.*
   (iv) For all formulae $Q_{\mathcal{A}}$ with $\mathrm{level}(Q_{\mathcal{A}}) \geq k$*, if $Q_{\mathcal{A}} \in \Gamma$, then $Q^{n}_{\mathcal{A}} \in \Gamma$ for some $n \in \omega$.*
2. *Γ is called k–saturated if it is k–presaturated and in addition the following property holds*

(v) For all formulae $P_{\mathcal{A}}$ with $\mathrm{level}(P_{\mathcal{A}}) \geq k$*, if* $P_{\mathcal{A}} \in \Gamma$*, then* $\mathcal{A}[P_{\mathcal{A}}] \in \Gamma$*.*

In general Γ is simply called saturated if it is 0–saturated. To show that any non–provable sequent Γ may be expanded to a saturated one we will use the strategy of choosing a formula in Γ which violates one of the conditions (ii) to (v) of Definition 11, adding suitable formulae to make the respective condition satisfied, then iterating this with a new formula which violates the conditions and so on. The essential step in the proof is then to show that this procedure converges after finitely many steps, thus yielding a saturated sequent. What makes matters complicated is condition (v) since satisfying this condition – in contrast to conditions (ii) to (iv) – means increasing formula complexity. In order to tackle these complications we need to make two technical definitions. The first one introduces a notation for the set of all those formulae in a sequent Γ which prevent Γ from being k–presaturated. The second definition introduces the set $\mathrm{csub}(A)$ of critical subformulae of formula A of $\mathcal{L}$. The intended meaning of $\mathrm{csub}(A)$ is being the set of all subformulae of A which could be considered during the process of saturation.

Definition 12 (k–Deficiency Set). *Let Γ be a sequent such that $\mathsf{T}^{\omega}_{\mathsf{SFL}} \nvdash \Gamma$ and k a natural number. Define the k–deficiency set $\mathrm{ds}_k(\Gamma)$ of Γ as the empty set if Γ is k–presaturated and otherwise as the set of all elements of Γ of level k which violate one of the conditions (ii) – (iv) of Definition 11.*

Definition 13 (Critical Subformulae). *For any $A \in \mathcal{L}$ define the set $\mathrm{csub}(A)$ of critical subformulae of A inductively as follows:*

1. *If A is an element of $\Phi \cup \mathsf{V} \cup \mathsf{T}$, a fixed point constant $P_{\mathcal{A}}$, a formula $\Diamond B$ or a formula $\Box B$, then define $\mathrm{csub}(A)$ as the set $\{A\}$.*
2. *If A is a fixed point constant $Q_{\mathcal{A}}$, then $\mathrm{csub}(A) := \{Q_{\mathcal{A}}\} \cup \{Q^n_{\mathcal{A}} : n \in \omega\}$.*
3. *If A is a formula $B \vee C$ or $B \wedge C$, then $\mathrm{csub}(A) := \{A\} \cup \mathrm{csub}(B) \cup \mathrm{csub}(C)$.*

For any sequent $\Gamma = \{A_1, \ldots, A_n\}$ set $\mathrm{csub}(\Gamma) := \mathrm{csub}(A_1) \cup \ldots \cup \mathrm{csub}(A_n)$.

In order to saturate an arbitrary non–provable sequent Γ we proceed as follows: Start with Γ and find the least number k for which Γ is k–saturated. If $k = 0$, then Γ is saturated and we are done. Thus assume $k = l + 1$ for some number l. Now we iteratively satisfy conditions (ii) to (iv) of Definition 11, producing in a finite number of steps a sequent Γ' which is l–presaturated, but not necessarily l–saturated. To achieve the latter, we add $\mathcal{A}[P_{\mathcal{A}}]$ to Γ' for each fixed point constant of level l and thus obtain a sequent Γ''. Unfortunately Γ'' need not necessarily be l–presaturated any longer but we notice that it is still $l + 1$–saturated. Thus we again iteratively satisfy conditions (ii) to (iv) of Definition 11 where after we arrive at an l–saturated sequent Γ'''. Repeating this procedure yields a strictly decreasing sequence of saturation numbers and thus after finitely many steps of adding formulae we reach a sequent which is saturated. We will now set about formalizing the saturation argument just described in the shape of the next three lemmata.

Lemma 5. *For every sequent Γ with $\mathsf{T}^{\omega}_{\mathsf{SFL}} \nvdash \Gamma$ there exists a natural number k such that Γ is k–saturated.*

Proof. By Definition 11 it is clear that the claim holds if we take k as the maximum level of all formulae in Γ plus 1. □

Lemma 6. *Suppose that k is a natural number and Γ a $(k+1)$–saturated sequent. Then there exists a k–presaturated sequent Δ so that*

$$\Gamma \subset \Delta \text{ and } (\Delta \setminus \Gamma) \subset \mathrm{csub}(\mathrm{ds}_k(\Gamma)).$$

Proof. The proof of this lemma is routine and will thus not be carried out. An almost identical claim is shown by Alberucci and Jäger [1] in full detail. □

Lemma 7. *Suppose that k is a natural number and Γ a $(k+1)$–saturated sequent. Then there exists a k–saturated sequent Δ such that $\Gamma \subset \Delta$.*

Proof. In a first step we apply Lemma 6 in order to obtain a k–presaturated sequent Δ_0 so that $\Gamma \subset \Delta_0$ and $(\Delta_0 \setminus \Gamma) \subset \mathrm{csub}(\mathrm{ds}_k(\Gamma))$. This sequent need not necessarily be k–saturated as condition (v) could be violated for some fixed point constants $P_{\mathcal{A}}$ of level k. To rectify this problem, our next step is to define the sequent $\Delta_1 := \Delta_0 \cup \{\mathcal{A}[P_{\mathcal{A}}] : \mathrm{level}(P_{\mathcal{A}}) = k \text{ and } P_{\mathcal{A}} \in \Delta_0\}$. Now, in turn, Δ_1 need not be k–presaturated since we have no guarantee that

$$\mathrm{ds}_k(\Delta_1) = \mathrm{ds}_k(\{\mathcal{A}[P_{\mathcal{A}}] : \mathrm{level}(P_{\mathcal{A}}) = k \text{ and } P_{\mathcal{A}} \in \Delta_0\})$$

is empty. However, again by using Lemma 6 we can extend Δ_1 to a k–presaturated sequent Δ with the properties $\Delta_1 \subset \Delta$ and $(\Delta \setminus \Delta_1) \subset \mathrm{csub}(\mathrm{ds}_k(\Delta_1))$. Thus all elements of $(\Delta \setminus \Delta_1)$ belong to the set

$$\mathrm{csub}(\{\mathcal{A}[P_{\mathcal{A}}] : \mathrm{level}(P_{\mathcal{A}}) = k \text{ and } P_{\mathcal{A}} \in \Delta_0\})$$

with the consequence that all fixed point constants of level k which are elements of Δ are already elements of Δ_0. Hence Δ is k–saturated and the lemma is shown. □

Combining Lemmata 5 and 7 now takes care of the first part of our completeness proof for $\mathsf{T}^{\omega}_{\mathsf{SFL}}$.

Lemma 8. *For every sequent Γ which is not provable in $\mathsf{T}^{\omega}_{\mathsf{SFL}}$ there exists a saturated sequent Δ such that $\Gamma \subset \Delta$.*

Proof. Assume $\mathsf{T}^{\omega}_{\mathsf{SFL}} \nvdash \Gamma$. Then by Lemma 5 we know that Γ is k–saturated for a suitably chosen k. Moreover, according to Lemma 7 there are sequents $\Gamma_{k-1}, \ldots, \Gamma_1, \Gamma_0$ so that each Γ_i is i–saturated for $0 \leq i \leq k-1$ and

$$\Gamma \subset \Gamma_{k-1} \subset \ldots \subset \Gamma_1 \subset \Gamma_0.$$

To conclude the proof simply set $\Delta := \Gamma_0$. □

Based on the collection of all saturated sequents we now define a Kripke structure $\mathsf{K}_{\mathrm{sat}}$ which will turn out to be a suitable countermodel for any non–provable formula of $\mathcal{L}$. The worlds of $\mathsf{K}_{\mathrm{sat}}$ are just the saturated sequents themselves. Accessibility is defined to treat formulae of the form $\Box B$ and $\Diamond B$ correctly and the valuation function makes a primitive proposition true in any world which does not contain it. This is possible since the non–provability of any saturated sequent Γ guarantees that not both p and $\sim\mathsf{p}$ are elements of Γ at once. The same is also true for the symbols X and $\sim\mathsf{X}$.

Definition 14 (Canonical Countermodel). *Set* $\mathsf{K}_{\mathrm{sat}} := (S_{\mathrm{sat}}, R_{\mathrm{sat}}, \pi_{\mathrm{sat}})$ *and*

$$\begin{aligned} S_{\mathrm{sat}} &:= \{\Gamma \subset \mathcal{L} : \Gamma \text{ saturated}\},\\ R_{\mathrm{sat}} &:= \{(\Gamma, \Delta) \in S_{\mathrm{sat}} \times S_{\mathrm{sat}} : \{B \in \mathcal{L} : \Diamond B \in \Gamma\} \subset \Delta\},\\ \pi_{\mathrm{sat}}(P) &:= \{\Gamma \in S_{\mathrm{sat}} : P \notin \Gamma\} \text{ for } P \in \Phi \cup \mathsf{V}. \end{aligned}$$

It is easily verified that $\mathsf{K}_{\mathrm{sat}}$ *is a Kripke structure in our sense. Given a formula* $A \in \mathcal{L}$ *and a set* $T \subset S_{\mathrm{sat}}$ *we will write* $\|A\|_{\mathrm{sat}}$ *for* $\|A\|_{\mathsf{K}_{\mathrm{sat}}}$ *and* $\|A\|_{\mathrm{sat}[\mathsf{X}:=T]}$ *for* $\|A\|_{\mathsf{K}_{\mathrm{sat}}[\mathsf{X}:=T]}$.

The completeness proof for $\mathsf{T}^{\omega}_{\mathsf{SFL}}$ will effectively be finished once we have shown for an arbitrary formula A of $\mathcal{L}$ that if A is an element of a saturated set Γ, then $\mathsf{K}_{\mathrm{sat}}$ is a countermodel for A at the world Γ. The next lemma shows that with respect to this property the construction of $\mathsf{K}_{\mathrm{sat}}$ consistently treats formulae of the form $\Box B$ and $\Diamond B$.

Lemma 9. *Assume* $\Gamma \subset \mathcal{L}$ *is a saturated sequent.*

(i) If $\Box A \in \Gamma$*, then there exists a sequent* Δ *such that* $\Gamma R_{\mathrm{sat}} \Delta$ *and* $A \in \Delta$.
(ii) If $\Diamond A \in \Gamma$*, then* $A \in \Delta$ *for all sequents* Δ *such that* $\Gamma R_{\mathrm{sat}} \Delta$.

Proof. To show (i) consider that since Γ is saturated, we have $\mathsf{T}^{\omega}_{\mathsf{SFL}} \nvdash \Gamma$ and thus also $\mathsf{T}^{\omega}_{\mathsf{SFL}} \nvdash \{B \in \mathcal{L} : \Diamond B \in \Gamma\}, A$. Thus by Lemma 8 there exists a saturated sequent Δ such that $\{B \in \mathcal{L} : \Diamond B \in \Gamma\}, A \subset \Delta$ and thus $A \in \Delta$ and $\Gamma R_{\mathrm{sat}} \Delta$. Part (ii) of the claim is obvious by Definition 14. □

The proofs of the lemmata leading up to the completeness theorem formally proceed as an induction along the levels of the language $\mathcal{L}$. That is to say we prove that for all formulae A of level 0 if A is an element of a saturated sequent Γ, then $\mathsf{K}_{\mathrm{sat}}$ is a countermodel for A at Γ. Then we show that if this claim holds for all formulae at level k, then it also holds for all formulae at level $k+1$. For this purpose we introduce the notion of k–adequacy of the Kripke structure $\mathsf{K}_{\mathrm{sat}}$.

Definition 15 (k–Adequacy). *Let* k *be a natural number. We call the Kripke structure* $\mathsf{K}_{\mathrm{sat}}$ *k–adequate if for all saturated sequents* Γ *and all formulae* A *of* $\mathcal{L}$ *we have*

$$\mathrm{level}(A) \leq k \text{ and } A \in \Gamma \implies \Gamma \notin \|A\|_{\mathrm{sat}}.$$

The naive approach to showing that if a formula A is in a saturated sequent Γ, then $\mathsf{K}_{\mathrm{sat}}$ is a countermodel for A at world Γ would be an induction on $\mathrm{rk}(A)$. However, this argument breaks down in the case where A is a constant $P_{\mathcal{A}}$. For this reason we must treat the case of constants $P_{\mathcal{A}}$ separately in the shape of the next lemma.

Lemma 10. *Suppose that* $\mathsf{K}_{\mathrm{sat}}$ *is* k*–adequate for the natural number* k *and let* $P_{\mathcal{A}}$ *be a fixed point constant of level* $k+1$*. Then we have for all* X*-positive formulae* $\mathcal{B}$ *such that* $\mathrm{level}(\mathcal{B}) \leq k$*, all saturated sequents* Γ *and all ordinals* α *that*

$$\mathcal{B}[P_{\mathcal{A}}] \in \Gamma \implies \Gamma \notin \|\mathcal{B}\|_{\mathrm{sat}[\mathsf{X}:=I^{<\alpha}_{\mathcal{A},\mathsf{K}_{\mathrm{sat}}}]}.$$

Proof. We prove this claim by main induction on α and side induction on $\mathrm{rk}(\mathcal{B})$. The atomic and truth value symbol cases are trivial, the propositional cases follow by hypothesis of the side induction and the modal cases by Lemma 9 and the hypothesis of the side induction. We are thus left with the fixed point and variable cases: In case $\mathcal{B} = P_{\mathcal{C}}$ or $\mathcal{B} = Q_{\mathcal{C}}$ we have $\mathcal{B}[P_{\mathcal{A}}] = \mathcal{B}$ and since $\mathrm{level}(\mathcal{B}) \leq k$ and $\mathsf{K}_{\mathrm{sat}}$ is k–adequate, we get

$$\Gamma \notin \|\mathcal{B}\|_{\mathrm{sat}} = \|\mathcal{B}\|_{\mathrm{sat}[\mathsf{X}:=I^{<\alpha}_{\mathcal{A},\mathsf{K}_{\mathrm{sat}}}]}.$$

In the case where $\mathcal{B} = \mathsf{X}$ from $\mathcal{B}[P_{\mathcal{A}}] \in \Gamma$, i.e. $P_{\mathcal{A}} \in \Gamma$, we immediately obtain $\mathcal{A}[P_{\mathcal{A}}] \in \Gamma$. Now we apply the hypothesis of the main induction and conclude that we have

$$\Gamma \notin \|\mathcal{A}\|_{\mathrm{sat}[\mathsf{X}:=I^{<\beta}_{\mathcal{A},\mathsf{K}_{\mathrm{sat}}}]}$$

for all $\beta < \alpha$. Semantic reasoning yields

$$\Gamma \notin \|\mathsf{X}\|_{\mathrm{sat}[\mathsf{X}:=I^{\beta}_{\mathcal{A},\mathsf{K}_{\mathrm{sat}}}]}$$

for all $\beta < \alpha$. Consequently we have

$$\Gamma \notin \|\mathsf{X}\|_{\mathrm{sat}[\mathsf{X}:=I^{<\alpha}_{\mathcal{A},\mathsf{K}_{\mathrm{sat}}}]},$$

and the claim is shown. □

Corollary 1. *Suppose that* $\mathsf{K}_{\mathrm{sat}}$ *is* k*–adequate for the natural number* k *and let* $P_{\mathcal{A}}$ *be a fixed point constant of level* $k+1$*. Then we have for all saturated sequents* Γ *that*

$$P_{\mathcal{A}} \in \Gamma \implies \Gamma \notin \|P_{\mathcal{A}}\|_{\mathrm{sat}}.$$

The next lemma takes care of the induction step, showing that if the Kripke structure $\mathsf{K}_{\mathrm{sat}}$ is k–adequate, then it is also $(k+1)$–adequate. This property will shortly lead us to the statement that indeed $\mathsf{K}_{\mathrm{sat}}$ is k–adequate for all natural numbers k.

Lemma 11. *Suppose that* $\mathsf{K}_{\mathrm{sat}}$ *is* k*–adequate for the natural number* k*. Then for all formulae* A *of* $\mathcal{L}$ *and all saturated sequents* Γ *we have*

$$A \in \Gamma \text{ and } \mathrm{level}(A) \leq k+1 \implies \Gamma \notin \|A\|_{\mathrm{sat}}.$$

Proof. We show this lemma by induction on $\mathrm{rk}(A)$. The atomic, variable and truth value symbol cases are trivial. The propositional and modal cases follow by induction hypothesis, the latter using Lemma 9. We are thus left with the fixed point cases: For the first case assume that $A = Q_{\mathcal{B}}$, $Q_{\mathcal{B}} \in \Gamma$ and that $\mathrm{level}(Q_{\mathcal{B}}) \leq k+1$. Then by saturation of Γ we have $Q^l_{\mathcal{B}} \in \Gamma$ for some $l \in \omega$ and $\mathrm{level}(Q^l_{\mathcal{B}}) < \mathrm{level}(Q_{\mathcal{B}}) \leq k+1$. Thus by induction hypothesis $\Gamma \notin \|Q^k_{\mathcal{B}}\|_{\mathrm{sat}}$ and thus also $\Gamma \notin \|Q_{\mathcal{B}}\|_{\mathrm{sat}}$. For the second case assume that $A = P_{\mathcal{B}}$, $P_{\mathcal{B}} \in \Gamma$ and $\mathrm{level}(P_{\mathcal{B}}) \leq k+1$. In the case where $\mathrm{level}(P_{\mathcal{B}}) \leq k$ it follows that $\Gamma \notin \|P_{\mathcal{B}}\|_{\mathrm{sat}}$ by k–adequacy of $\mathsf{K}_{\mathrm{sat}}$, in the case where $\mathrm{level}(P_{\mathcal{B}}) = k+1$ the claim follows by Corollary 1. □

We are now ready to prove the crucial lemma establishing the fact that any formula contained in a saturated sequent Γ is not satisfied in $\mathsf{K}_{\mathrm{sat}}$ at Γ.

Lemma 12. *For any natural number k, the Kripke structure $\mathsf{K}_{\mathrm{sat}}$ is k–adequate; i.e. for all formulae $B \in \mathcal{L}$ and all saturated sequents $\Gamma \subset \mathcal{L}$ we have*

$$B \in \Gamma \implies \Gamma \notin \|B\|_{\mathrm{sat}}.$$

Proof. This lemma is shown by a straightforward induction on k using Lemma 11 in the induction step. □

Theorem 8 (Completeness of $\mathsf{T}^{\omega}_{\mathsf{SFL}}$). *The system $\mathsf{T}^{\omega}_{\mathsf{SFL}}$ is complete, that is for all formulae $A \in \mathcal{L}$ if A is valid, then $\mathsf{T}^{\omega}_{\mathsf{SFL}} \vdash A$.*

Proof. We show the claim by contraposition. Assume $\mathsf{T}^{\omega}_{\mathsf{SFL}} \nvdash A$. Then by Lemma 8 there exists a saturated set $\Gamma \subset \mathcal{L}$ such that $A \in \Gamma$. Therefore by Lemma 12 we have $\Gamma \notin \|A\|_{\mathrm{sat}}$ and thus A cannot be valid. Hence we have shown the completeness of $\mathsf{T}^{\omega}_{\mathsf{SFL}}$. □

7 Finitizing $\mathsf{T}^{\omega}_{\mathsf{SFL}}$

We will now use the so–called small model property of SFL to reduce the number of premises of the greatest fixed point rule $(Q^{\omega}_{\mathcal{A}})$ down to a single premise. Doing so will result in a truly finitary system in which all proofs are finite in length. Before we address the soundness proof of $\mathsf{T}^{\omega}_{\mathsf{SFL}}$ we state the small model property in its customary form.

Remark 2 (Small model property). There exists an exponential function f on the natural numbers such that for every formula $A \in \mathcal{L}$ if A is satisfiable, then there exists a Kripke structure $\mathsf{K} = (S, R, \pi)$ with $|S| < f(|A|)$ which satisfies A.

The small model property holds for stratified modal fixed point logic since it holds for the modal μ–calculus. A candidate for the exponential function f mentioned in Remark 2 can be reconstructed from results presented in [9] or [16]. The exact shape of f or indeed a minimal candidate with respect to our framework shall not concern us in the current study.

Definition 16 (The System $\mathsf{T}_{\mathsf{SFL}}$). *The system $\mathsf{T}_{\mathsf{SFL}}$ is defined by replacing the rule $(Q^{\omega}_{\mathcal{A}})$ in the system $\mathsf{T}^{\omega}_{\mathsf{SFL}}$ by the rule*

$$\frac{\Gamma, Q^{k}_{\mathcal{A}}}{\Gamma, Q_{\mathcal{A}}, \Sigma} \quad (Q_{\mathcal{A}})$$

where $k = f(|\bigvee(\Gamma, Q_{\mathcal{A}})|)$.

The function f guaranteed to exists by Remark 2 is used to bound the number of iterations $Q^{k}_{\mathcal{A}}$ for which we need to check the derivability of $\Gamma, Q^{k}_{\mathcal{A}}$ before applying the rule $(Q_{\mathcal{A}})$ to conclude $\Gamma, Q_{\mathcal{A}}$. Indeed f supplies us with the only such iteration we need to check explicitly, as the subsequent argument will show. In order to obtain weakening the conclusion of the rule $(Q_{\mathcal{A}})$ needs to weakened by an arbitrary sequent Σ explicitly.

The structure of the rule $(Q_{\mathcal{A}})$ makes it clear that completeness of $\mathsf{T}_{\mathsf{SFL}}$ is implied by completeness of $\mathsf{T}^{\omega}_{\mathsf{SFL}}$. With completeness of $\mathsf{T}_{\mathsf{SFL}}$ taken care of the only task that remains is to show soundness. Before we proceed to show the soundness of $\mathsf{T}_{\mathsf{SFL}}$ we establish some consequences of the small model property. The first fact we note is essentially just the contraposition of Remark 2.

Lemma 13. *Let $A \in \mathcal{L}$. If A is valid in all Kripke structures K with $|\mathsf{K}| \leq f(|A|)$, then A is valid.*

Proof. The proof of this claim is trivial by the fact that $|\neg A| = |A|$. □

The second fact we require in order to show the soundness of $\mathsf{T}_{\mathsf{SFL}}$ states that in order to check the validity of greatest fixed point in all Kripke structures with at most k worlds, we only need to check the validity of its k-th approximation. The essential ingredient to proving this claim is the boundedness of closure ordinals established in Theorem 2.

Lemma 14. *If the formula $\bigvee(\Gamma, Q^{k}_{\mathcal{A}})$ is valid, then the formula $\bigvee(\Gamma, Q_{\mathcal{A}})$ is valid in all Kripke structures K with $|\mathsf{K}| \leq k$.*

Proof. Let $k \in \omega$ be arbitrary and K be a Kripke structure with $|\mathsf{K}| \leq k$. Since the formula $\bigvee(\Gamma, Q^{k}_{\mathcal{A}})$ is valid it is also valid in K. Since $|\mathsf{K}| \leq k$, Theorem 2 guarantees that the formula $Q^{k}_{\mathcal{A}} \leftrightarrow Q_{\mathcal{A}}$ is valid in K. Thus $\bigvee(\Gamma, Q_{\mathcal{A}})$ is valid in K and the claim is shown. □

We are now ready to state and prove the soundness of the system $\mathsf{T}_{\mathsf{SFL}}$.

Theorem 9 (Soundness of $\mathsf{T}_{\mathsf{SFL}}$). *The system $\mathsf{T}_{\mathsf{SFL}}$ is sound, that is for all sequents $\Gamma \subset \mathcal{L}$ if $\mathsf{T}_{\mathsf{SFL}} \vdash \Gamma$, then the formula $\bigvee \Gamma$ is valid.*

Proof. We must show that all axioms of $\mathsf{T}_{\mathsf{SFL}}$ are valid and that all rules of $\mathsf{T}_{\mathsf{SFL}}$ preserve validity. However, in view of the definition of $\mathsf{T}_{\mathsf{SFL}}$ we merely need to check the rules $(P_{\mathcal{A}})$ and $(Q_{\mathcal{A}})$. For the rule $(P_{\mathcal{A}})$ assume the formula $\bigvee(\Gamma, \mathcal{A}[P_{\mathcal{A}}])$ representing the premise of $(P_{\mathcal{A}})$ is valid. By the semantics of $\mathcal{L}$ the

formula $\mathcal{A}[P_{\mathcal{A}}] \leftrightarrow P_{\mathcal{A}}$ is also valid and hence the formula $\bigvee(\Gamma, P_{\mathcal{A}})$ representing the conclusion of $(P_{\mathcal{A}})$ is valid and thus the rule is sound. For the rule $(Q_{\mathcal{A}})$ assume that the formula $\bigvee(\Gamma, Q^k_{\mathcal{A}})$ where $k = f(|\bigvee(\Gamma, Q_{\mathcal{A}})|)$ representing the premise of $(Q_{\mathcal{A}})$ is valid. Therefore by Lemma 14 the formula $\bigvee(\Gamma, Q_{\mathcal{A}})$ is valid in all Kripke structures K with $|\mathsf{K}| \leq k$. Therefore by Lemma 13 the formula $\bigvee(\Gamma, Q_{\mathcal{A}})$ representing the conclusion of $(Q_{\mathcal{A}})$ is valid and thus the rule is sound. This concludes the soundness proof for the system $\mathsf{T}_{\mathsf{SFL}}$. □

8 Concluding Remarks

The approach to stratification presented in this paper can be easily generalized to a form of stratification which permits, at each level, the introduction of simultaneously defined least and greatest fixed points. To give a simple example, extend our language $\mathcal{L}_0$ so that it comprises countably many variables $\mathsf{X}_1, \mathsf{X}_2, \mathsf{X}_3, \ldots$ and their negations $\sim\mathsf{X}_1, \sim\mathsf{X}_2, \sim\mathsf{X}_3, \ldots$. Now assume that we are given a system of, say, three formulas

$$\mathcal{A}_1[\mathsf{X}_1, \mathsf{X}_2, \mathsf{X}_3], \quad \mathcal{A}_2[\mathsf{X}_1, \mathsf{X}_2, \mathsf{X}_3], \quad \mathcal{A}_3[\mathsf{X}_1, \mathsf{X}_2, \mathsf{X}_3],$$

all positive in $\mathsf{X}_1, \mathsf{X}_2, \mathsf{X}_3$. Then we add three new constants $P_{\mathcal{A}_1}$, $P_{\mathcal{A}_2}$ and $P_{\mathcal{A}_3}$ as well as three constants $Q_{\mathcal{A}_1}$, $Q_{\mathcal{A}_2}$ and $Q_{\mathcal{A}_3}$ to represent the least and greatest fixed points simultaneously generated by these three formulas.

In the Hilbert system for stratified simultaneous modal fixed point logic the corresponding closure axioms read as follows:

$$\begin{aligned}
\mathcal{A}_1[P_{\mathcal{A}_1}, P_{\mathcal{A}_2}, P_{\mathcal{A}_3}] &\rightarrow P_{\mathcal{A}_1}, \\
\mathcal{A}_2[P_{\mathcal{A}_1}, P_{\mathcal{A}_2}, P_{\mathcal{A}_3}] &\rightarrow P_{\mathcal{A}_2}, \\
\mathcal{A}_3[P_{\mathcal{A}_1}, P_{\mathcal{A}_2}, P_{\mathcal{A}_3}] &\rightarrow P_{\mathcal{A}_3}.
\end{aligned}$$

Furthermore, for all formulas B_1, B_2, B_3 we have as induction rules:

$$\frac{(\mathcal{A}_1[B_1, B_2, B_3] \rightarrow B_1) \wedge (\mathcal{A}_2[B_1, B_2, B_3] \rightarrow B_2) \wedge (\mathcal{A}_3[B_1, B_2, B_3] \rightarrow B_3)}{P_{\mathcal{A}_i} \rightarrow B_i}$$

where $i \in \{1, 2, 3\}$. This can be done for arbitrary (finite) systems of positive formulas and analogously carried through for all levels of stratification.

Even though the languages thus obtained and the corresponding systems are strictly more expressive than the ones studied in this article (cf. e.g. [2, 3]), all our methods remain applicable in this more general framework.

References

1. Luca Alberucci and Gerhard Jäger. About cut–elimination for logics of common knowledge. *Annals of Pure and Applied Logic*, 133:73–99, 2005.

2. Dietmar Berwanger, Erich Grädel, and Giacomo Lenzi. On the variable hierarchy of the modal μ–calculus. In *CSL '02: Proceedings of the 16th International Workshop and 11th Annual Conference of the EACSL on Computer Science Logic*, pages 352–366. Springer, 2002.
3. Dietmar Berwanger and Giacomo Lenzi. The variable hierarchy of the modal μ–calculus is strict. In *STACS 2005, Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science*, LNCS. Springer, 2005. (to appear).
4. Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logics of Programs*, volume 131 of *LNCS*, pages 52–71. Springer, 1981.
5. E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu–calculus and determinacy (extended abstract). In *Proceedings of the 32th Annual IEEE Symposium on Foundations of Computer Science*, pages 3368–3377. IEEE Computer Society Press, 1991.
6. Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.
7. Michael J. Fischer and Richard E. Ladner. Propositional modal logic of programs. In *Proc. 9th ACM Annual Symposium on Theory of Computing*, pages 286 – 294, 1977.
8. Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computing and System Science*, 18(2):194 – 211, 1979.
9. Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.
10. Gerhard Jäger, Mathis Kretz, and Thomas Studer. Cut–free common knowledge. submitted, 2005.
11. Dexter Kozen. Results on the propositional modal μ–calculus. *Theoretical Computer Science*, 27:333–354, 1983.
12. Yannis Moschovakis. *Elementary Induction on Abstract Structures*. Studies in Logic and the Foundations of Mathematics. North-Holland, 1974.
13. Colin Stirling and David Walker. Local model–checking in the modal mu–calculus. *Theoretical Computer Science*, 89:161–177, 1991.
14. Robert S. Streett and E. Allen Emerson. An automata theoretic decision procedure for the propositional modal mu-calculus. *Information and Computation*, 81(3):249–264, 1989.
15. Igor Walukiewicz. Completeness of Kozen's axiomatization of the propositional μ–calculus. *Information and Computation*, 157:142–182, 2000.
16. Thomas Wilke. Alternating tree automata, parity games, and modal μ-calculus. *Bull. Soc. Math. Belg.*, 8(2), 2001.
17. Glynn Winskel. A note on model checking the modal ν–calculus. *Theoretical Computer Science*, 83:157–167, 1991.

Natural deduction for belief *at most*

Johan W. Klüwer[1] and Arild Waaler[2]

[1] Dep. of Philosophy, University of Oslo, johanw@filosofi.uio.no
[2] Finnmark College and Dep. of Informatics, University of Oslo, arild@ifi.uio.no

Abstract. We present a new approach to the logic of *at most*, introducing the notion of *parametric propositions* to modal natural deduction proofs. We apply the method with a natural deduction formulation of the doxastic logic $\mathrm{N}^{\text{Æ}}$, a new system in the "only knowing" family. Using parametric proof rules, we give introduction and elimination rules for belief *at most* that directly match the natural second-order formulation of the concept. $\mathrm{N}^{\text{Æ}}$ is sound and complete with respect to the class of intended models. We conjecture that it weakly normalizes and that it satisfies the subformula property.

1 Introduction

The aim of this paper is to provide a new approach to modal logic with *at most* modalities. To the basis of a bimodal natural deduction system, we add introduction and elimination rules for an *at most* modality that directly match its natural definition as a second-order concept. In the rules, second-order quantifiers are reflected in an original use of *parametric propositions*, the role of which is reminiscent of the way parameters are used in the standard analysis of first-order quantifiers. The according pattern of *parametric proposition proof* enable us to directly translate second-order definitions of logical constants into proof rules.[3]

We do not here attempt a general analysis of proof with parametric propositions, but focus on the application to doxastic logic. For this purpose, we introduce the natural deduction system $\mathrm{N}^{\text{Æ}}$ of belief, conceivability, and belief *at most*. In $\mathrm{N}^{\text{Æ}}$ the modal rules for belief and necessity are standard, providing a well understood analysis of basic concepts, to which parametric rules for belief *at most* are added. We show that on this approach, the introspection properties we take to be valid for belief and conceivability are inherited in the appropriate way by belief *at most*. A consequence of this is that the introspection schemas adopted for belief *at most* in existing "all I know" logics are not just stipulations, but follow from conceptually adequate second-order assumptions that can readily be given intuitive interpretation.

Current approaches to *at most* modalities employ complementary logics, first addressed by Humberstone [3]. These logics adopt one modality which, in the

[3] Our deployment of parametric propositions in proof rules is inspired by Frank Pfenning's use of parametric judgments in his treatment of negation [5, 1078 f].

semantics, quantifies over a preferred set of points, and another modality which intuitively quantifies over a complementary set of points. The framework was, in particular, successfully applied by Levesque [4] to the study of default reasoning. However, a weakness of the standard approach to *at most* is that the systems have unintended models (see [3, p. 350]), in which the *at most* modality complementary to belief can not be fully accounted for by reference to strength of belief alone (cf. [8, section 3.2] for a detailed discussion). Perhaps this has not been seen as problematic because the default representation pattern introduced by Levesque employs only formulae that require for their truth that models are of the intended kind (see [8, lemma 2]).

There is however every reason to believe that the scope of *at most* modalities goes beyond the representation of defaults, so finding the correct logic is still of interest. We will in this paper remedy the situation by providing a logic of *at most* that (i) is based only on notions – belief and conceivability – that can be given solid common-sense interpretations, and (ii) whose natural models are of the intended kind.

The object language of our target system $\mathrm{N}^{\text{Æ}}$ is defined as follows: It contains a stock of propositional variables, the constants $\top$ and $\bot$, and connectives $\neg, \vee, \wedge, \supset$, and $\equiv$. Modal operators are $\Diamond$ (conceivability), B (belief) and $\breve{B}$, and their duals: $\Box$ (necessity) is $\neg\Diamond\neg$, b (plausibility) is $\neg B\neg$, and $\breve{b}\varphi$ is $\neg\breve{B}\neg\varphi$. The desired interpretation of $\breve{B}\varphi$ is that at most $\neg\varphi$ is believed. Giving conceptual priority to belief and strength of propositions, we take this to mean that φ is at least as strong as the strongest proposition that is believed. Observe that "φ is at least as strong as ψ" has a natural expression, using the notion of conceivability, in the formula $\Box(\varphi \supset \psi)$. This means that we can specify the meaning of $\breve{B}\varphi$, using a propositional quantifier, as follows.

$$\breve{B}\varphi =_{\text{def}} \forall p(Bp \supset \Box(\neg\varphi \supset p))\,. \tag{1}$$

$\breve{B}\neg\varphi$ is true whenever φ is a proposition that necessarily implies every believed proposition. Its negation $\breve{b}\varphi$ then means that some belief is held, the negation of which is compatible with φ. In terms of possible worlds, $\breve{b}\varphi$ expresses that there is a conceivable world in which φ is true, and which is believed not to be the actual world.

$$\breve{b}\varphi =_{\text{def}} \exists p(Bp \wedge \Diamond(\varphi \wedge \neg p))\,. \tag{2}$$

Overview of the paper: In section 2, we present a Fitch-style framework for natural deduction with modalities. The framework is applied in section 3, introducing the natural deduction system $\mathrm{N}^{\text{Æ}}$. Section 4 recounts the semantics of the system Æ [8], with restriction to models that can properly be said to be intended for the logic of *at most*. $\mathrm{N}^{\text{Æ}}$ is sound and complete with regard to these models, and proofs of this are given in sections 4, 5, and 6. Section 7 addresses normalization and the subformula property, which is essential for proof-search applications.

2 The natural deduction framework

We here present the framework of Fitch-style natural deduction by means of which the system $\mathrm{N}^{\text{Æ}}$ is defined in section 3. We assume that a standard set of rules for the propositional operators $\bot$, $\wedge$, $\vee$, and $\supset$ is in place (see e.g. [6]).

Parametric proofs. With the aim of representing second-order quantifiers, we introduce a new kind of subproof pattern, *parametric proof.* A parametric proof may be opened anywhere, and is annotated with a propositional variable which is not previously used in the proof (we will use p, q, r as names for variables). At the opening of a parametric proof, the formula p employed is said to be *parametric*, and should be seen as standing for any proposition. p remains parametric until used in an inference rule that requires it to have an unchanging value, after which it is said to be *fixed* (analogously to the binding of a new constant in elimination of a first-order existential quantifier). A subproof annotated with p is closed by an application of PP.

$$\begin{array}{l|l|ll}
 & p & \vdots & \\
 & & \varphi & p \text{ not in } \varphi \quad \text{PP} \\
\rhd & \varphi & &
\end{array}$$

PP is a sound rule: if φ follows from the mere assumption that p is a proposition, but does not involve p itself, then φ must be true.

Negation rules. The rules for negation match the second-order definition of negation as "explosion", $\neg\varphi$ iff $\forall\psi(\varphi \supset \psi)$. Explosion obtains if every proposition is provable. Thus, if we can prove that $\varphi \supset p$ is provable on the mere assumption that p is a proposition $\neg\varphi$ may be introduced.

$$\begin{array}{l|ll}
 & \varphi \supset p & (p \text{ parametric}) \\
\rhd & \neg\varphi & \text{fixing } p
\end{array} \quad \neg\text{I} \qquad\qquad \begin{array}{l|ll}
 & \neg\varphi & \\
\rhd & \varphi \supset \psi &
\end{array} \quad \neg\text{E}$$

In order to obtain a classical system, we add the rule of excluded middle.

$$\begin{array}{l|ll}
\rhd & \varphi \vee \neg\varphi & \text{tnd}
\end{array}$$

Modal rules. For reasoning with standard modalities, we add the device of *modal subproof.* The presentation here is intended as schematic, with L standing for any universal modality (here, $\Box$ or B) and M for any existential modality ($\Diamond$ or b). We take the rules for L as basic; rules for M can easily be derived.[4]

[4] The rules follow the format introduced by Fitch [e.g., 1]; cf. Siemens [7] and Fitting [2] for application to a variety of modal systems. In the terminology of Fitting, we have *A-style* rules as basic, and see *I-style* rules as abbreviations. In naming the rules, we have followed Siemens [7].

An L-subproof may be opened at any point in a proof. Introduction of $L\varphi$ requires φ to be proven in an L-modal subproof. Iteration into modal subproofs is restricted to formulae φ s.t. $L\varphi$.

$$\begin{array}{l|l|l} & L & \vdots \\ & & \varphi \\ \rhd & L\varphi & \end{array} \quad L\mathrm{I} \qquad\qquad \begin{array}{l|l|l} & L\varphi & \\ L & \vdots & \\ \rhd & \varphi & \end{array} \quad L\mathrm{R}$$

Adding the rules LI and LR to the classical propositional logic yields the normal system K. Rules may be added to yield stronger systems, corresponding to taking axioms as valid. Here, we will be considering the rules LE (corresponding to the truth schema $L\varphi \supset \varphi$), WLE (corresponding to $L\varphi \supset M\varphi$), LLR (corresponding to $L\varphi \supset LL\varphi$), and MLR (corresponding to $M\varphi \supset LM\varphi$).

$$\begin{array}{l|l} & L\varphi \\ \rhd & \varphi \end{array} \quad L\mathrm{E} \qquad\qquad \begin{array}{l|l} & L\varphi \\ \rhd & M\varphi \end{array} \quad \mathrm{W}L\mathrm{E}$$

$$\begin{array}{l|l|l} & L\varphi & \\ L & \vdots & \\ \rhd & L\varphi & \end{array} \quad LL\mathrm{R} \qquad\qquad \begin{array}{l|l|l} & M\varphi & \\ L & \vdots & \\ \rhd & M\varphi & \end{array} \quad ML\mathrm{R}$$

3 The system $\mathrm{N}^{\text{Æ}}$

We propose a system for reasoning about belief B, conceivability $\Diamond$, and belief *at most* $\breve{B}\neg$: Using the naming patterns for rules given in section 2, the system $\mathrm{N}^{\text{Æ}}$, and its provability relation $\vdash$, is given by the following.

For the non-modal fragment, adopt a standard set of rules for $\bot$, $\wedge$, $\vee$, and $\supset$, to which is added PP, $\neg$I, $\neg$E, and tnd. Rules for B are BI, BR, BBR, and bBR, which yields a K45 modality. Rules for $\Box$ are $\Box$I, $\Box$R, $\Box$E, $\Box\Box$R and $\Diamond\Box$R, for an S5 modality. Rules for interaction between B and $\Box$ are defined below: they amount to $\Box B$E, weak introspection $\Box B$R and $\Diamond B$R, and strong introspection $B\Box$R and $b\Box$R. Parametric rules $\breve{B}$E and $\breve{B}$I for the $\breve{B}$ modality conclude the definition of the system.

The fundamental assumption about the relationship between belief and conceivability is that what is not conceivably false (i.e., what is necessary) is also believed. This is captured in $\Box B$E.

$$\begin{array}{l|l} & \Box\varphi \\ \rhd & B\varphi \end{array} \quad \Box B\mathrm{E}$$

$\Box B$R and $\Diamond B$R are weak introspection principles, representing assumptions that what is necessary, resp. conceivable is believed to be so; they may be seen as weakened versions of $4_\Box$ resp. $5_\Box$.

$$\begin{array}{l|l} & \Box\varphi \\ B & \vdots \\ \rhd & \Box\varphi \end{array} \qquad \Box B\mathrm{R} \qquad\qquad \begin{array}{l|l} & \Diamond\varphi \\ B & \vdots \\ \rhd & \Diamond\varphi \end{array} \qquad \Diamond B\mathrm{R}$$

$B\Box$R and $b\Box$R are strong introspection principles, saying it is inconceivable for the doxastic subject that beliefs and non-beliefs could be different from what they are.

$$\begin{array}{l|l} & B\varphi \\ \Box & \vdots \\ \rhd & B\varphi \end{array} \qquad B\Box\mathrm{R} \qquad\qquad \begin{array}{l|l} & \neg B\varphi \\ \Box & \vdots \\ \rhd & \neg B\varphi \end{array} \qquad b\Box\mathrm{R}$$

The rules for belief *at most* directly reflect the modality's second-order definition (1). We take the pair of rules $\breve{B}$E and $\breve{B}$I to be primitive, and consider the rules $\breve{b}$E and $\breve{b}$I to be derived rules. In $\breve{B}$I and $\breve{b}$E q is assumed to be parametric.

$$\begin{array}{l|l} & \breve{B}\varphi \\ \rhd & B\psi \supset \Box(\neg\varphi \supset \psi) \end{array} \qquad \breve{B}\mathrm{E} \qquad\qquad \begin{array}{l|l} & \breve{b}\varphi \\ \rhd & Bq \wedge \Diamond(\varphi \wedge \neg q) \end{array} \qquad \text{fixing } q \quad \breve{b}\mathrm{E}$$

$$\begin{array}{l|l} & Bq \supset \Box(\neg\varphi \supset q) \\ \rhd & \breve{B}\varphi \end{array} \qquad \text{fixing } q \quad \breve{B}\mathrm{I} \qquad\qquad \begin{array}{l|l} & B\psi \wedge \Diamond(\varphi \wedge \neg\psi) \\ \rhd & \breve{b}\varphi \end{array} \qquad \breve{b}\mathrm{I}$$

4 Soundness

A *model* M is a quadruple (U, U^+, U^-, V), where U is a non-empty set of *points*, U^+ and U^- are subsets of U such that $U^+ \cup U^- = U$, and V is an valuation function which assigns a subset of U to each propositional variable in the language. A satisfaction relation can be defined for each point x in M,

$$\begin{aligned} M \vDash_x p &\quad \text{iff } x \in V(p) \text{ for a propositional variable } p \\ M \vDash_x \Box\varphi &\text{ iff } M \vDash_y \varphi \text{ for each } y \in U \\ M \vDash_x B\varphi &\text{ iff } M \vDash_y \varphi \text{ for each } y \in U^+ \\ M \vDash_x \breve{B}\varphi &\text{ iff } M \vDash_y \varphi \text{ for each } y \in U^- \ , \end{aligned}$$

and as usual for Boolean connectives. A formula is *satisfied* in a model if it is true at one of its points. If $M \vDash_x \varphi$ for all $x \in U$, we write $M \vDash \varphi$ and say that φ is *true in M*. The *truth set* of φ, denoted $||\varphi||$, is the set of points in U at which φ is true.

Observe that all points in a model agree on the truth value of every completely modalized formula. Hence, for such formulae the notions of satisfaction and truth in a model coincide. This justifies use of the notation $M \vDash \varphi$ whenever a completely modalized φ is satisfied in M.

A model of this kind, further exposed in [8, section 3.2], is intended to represent a doxastic subject, where U is the *space of conceivability*, i.e., the range of states of affairs that the subject can conceive of, and the subject has a *belief state*, modeled by U^+, and a *co-belief state*, modeled by U^-. Any point in U^+ is, on the perspective of the doxastic subject, a plausible candidate for being the actual world. U^- is intended to capture the set of *implausible* points, i.e, the set of points that are ruled out (by the subject's belief) as not actual.

Semantically, the intended interpretation requires that the models satisfy a notion of bisectedness to distinguish the belief state from the co-belief state. We will say a model M is *strongly bisected* if $U^+ \cap U^- = \emptyset$ and *weakly bisected* if the following condition holds for all formulae φ: if $||\varphi|| \subseteq ||\psi||$ for every ψ such that $U^+ \subseteq ||\psi||$, then $||\varphi|| \cap U^- = \emptyset$.

That is, in a weakly bisected model $U^+ \cap U^-$ can be non-empty, but there is no formula in the language which we can use to define these points. Although $\mathrm{N}^{\text{Æ}}$ is characterized by the class of strongly bisected models, it is also characterized by the set of weakly bisected models, which, in light of Lemma 1 below, seems to be the more natural class. The two notions of bisectedness coincide for finite models. For infinite models the syntax is in general not expressive enough to enforce a complete separation of the plausible points from the implausible ones; the reason being that the language is countable while the set of propositions in general is uncountable.

We can recast the stipulative definition of the $\breve{B}$ operator in section 1 into semantical terms. A model M satisfies the $\breve{B}$*-property* if for each formula φ, $\breve{B}\varphi$ is true in M iff $B\psi \supset \Box(\neg\varphi \supset \psi)$ is true in M for every formula ψ.

Lemma 1. *A model is weakly bisected if and only if it satisfies the* $\breve{B}$*-property.*

Proof. Assume that M is weakly bisected. If $\breve{B}\varphi$ is true in M, it follows from the model condition $(U \setminus U^-) \subseteq U^+$ that (†) $B\psi \supset \Box(\neg\varphi \supset \psi)$ is true for every ψ. Conversely, if the latter condition (†) holds, it follows from weak bisection that $||\neg\varphi|| \cap U^- = \emptyset$ and hence that $\breve{B}\varphi$ is true in M. Assume that M is not weakly bisected. There is then a formula φ such that (1) $||\varphi|| \subseteq ||\psi||$ for every ψ such that $U^+ \subseteq ||\psi||$ and (2) $||\varphi|| \cap U^- \neq \emptyset$. (1) implies that the (†) condition holds, while (2) implies that $\breve{B}\varphi$ is not true in M. M hence fails to satisfy the $\breve{B}$-property. □

Theorem 1. *Assume that there is a proof π of φ. Then φ is true at all points in all weakly bisected models.*

Proof. We argue by induction over π and show that all formulae in π are satisfied in all appropriate models at all appropriate points. Initially, all weakly bisected models and all points in any of these models are appropriate, but this may change as we inductively progress through the proof structure. When a hypothetical subproof is opened the appropriate set of points (wrt. the model at hand) is narrowed to those that satisfy the assumption of the subproof. When a $\Box$-subproof (B-subproof) is opened, the set of appropriate points is U (U^+). When a subproof of any of these three types is closed its constraint is suppressed and we reassume the notion of appropriateness that was operational immediately before the subproof was opened. When a parametric proposition is fixed by a $\neg$I ($\breve{B}$I) inference, the set of appropriate *models* is narrowed to those that assign $\emptyset$ (U^+) to p; this model constraint is suppressed when the parametric subproof annotated with p is closed by an application of PP. Given these notions the inductive soundness argument is straightforward. The only non-trivial cases are the $\breve{B}$E rule, for which we use Lemma 1, and the PP rule, in which case we can prove by a simple induction that removing the model constraint does not disturb the truth value of a formula which satisfies the side condition of the rule. $\Box$

A system in which there is an unprovable formula is *consistent*. It follows trivially from Theorem 1 that $\mathrm{N}^{\text{Æ}}$ (and all its subsystems) is consistent.

5 Derivability in $\mathrm{N}^{\text{Æ}}$

For the purpose of comparison to existing doxastic logics, in particular Æ [8], let us look at the strength of the *at most* modality $\breve{B}$ in $\mathrm{N}^{\text{Æ}}$.

The following two lemmas establish the relationship between B, $\breve{B}$, and $\Box$, and the fact that $\breve{B}$ is a normal modality. These results can naturally be considered basic, as they are independent of any introspection schemas for B or $\Box$, relying only on their being normal modalities. (Justification for standard proof steps is abbreviated "ml" for "modal logic".)

Lemma 2. $\vdash \Box\varphi$ *if and only if* $\vdash B\varphi \wedge \breve{B}\varphi$

Proof. Assume $\Box\varphi$; $B\varphi$ follows by $\Box B$E. For $\breve{B}\varphi$, open a parametric subproof with p as parameter. $\Box\varphi$ implies $Bp \supset \Box(\neg\varphi \supset p)$ by ml. Then $\breve{B}\varphi$ may be introduced, and returned from the subproof by PP because p doesn't appear in $\breve{B}\varphi$. Assume $B\varphi$ and $\breve{B}\varphi$; from the latter, infer $B\varphi \supset \Box(\neg\varphi \supset \varphi)$ by $\breve{B}$E. Then $\Box\varphi$ follows by ml. $\Box$

Lemma 3. $\breve{B}$ *is a normal modality.*

Proof. Lemma 2 implies that the necessitation rule $\varphi/\breve{B}\varphi$ is valid. Assume $\breve{B}(\varphi \supset \psi)$ and $\breve{B}\varphi$. It suffices for normality of $\breve{B}$ that we can show $\breve{B}\psi$. Open a

subproof with p as parameter; we want to show $Bp \supset \Box(\neg\psi \supset p)$ (with p parametric). Applying $\breve{B}$E twice, we infer $Bp \supset \Box(\neg(\varphi \supset \psi))$ and $Bp \supset \Box(\neg\varphi \supset p)$, and then $Bp \supset \Box(\neg\psi \supset p)$ by ml. Then $\breve{B}\psi$ follows by $\breve{B}$I, and $\breve{B}\psi$ may be returned from the subproof by PP. □

The introspection schemas for B and $\Box$ imply corresponding schemas for $\breve{B}$. $\breve{B}\neg\varphi$ expresses that no proposition stronger than φ is believed, and $\breve{b}\varphi$ that some proposition is believed, the negation of which is compatible with φ. This means that just as B is a universal belief modality and b an existential modality of non-belief, so $\breve{B}$ is a universal non-belief modality, and $\breve{b}$ an existential belief modality [cf. 8, section 3.3]. In $\mathrm{N}^{\text{Æ}}$, bBR and $b\Box$R are negative introspection rules, preserving non-belief, while BBR and $B\Box$R are positive introspection rules, preserving belief.

The following lemma demonstrates that, as intuitively desired, the rules preserving non-belief imply $\breve{B}$ iteration schemas, and the rules preserving belief imply $\breve{b}$ iteration. The rules bBR and BBR, together with weak introspection for $\Box$, validate introspection of $\breve{B}$ and $\breve{b}$ statements into B contexts. With S5 properties of $\Box$ plus strong introspection rules $b\Box$R and $B\Box$R, we gain full introspection for $\breve{B}$, i.e., the schemas $\breve{B}\varphi \supset \breve{B}\breve{B}\varphi$ and $\breve{b}\varphi \supset \breve{B}\breve{b}\varphi$.

Lemma 4.

1. $\breve{B}\varphi \vdash B\breve{B}\varphi$ *given* bBR, $\Box BR$.
2. $\breve{b}\varphi \vdash B\breve{b}\varphi$ *given* BBR, $\Diamond BR$.
3. $\breve{B}\varphi \vdash \breve{B}\breve{B}\varphi$ *given* $b\Box R$, $\Box\Box R$.
4. $\breve{b}\varphi \vdash \breve{B}\breve{b}\varphi$ *given* $B\Box R$, $\Diamond\Box R$.

Proof.

1.

1		$\breve{B}\varphi$	
2	p	$Bp \supset \Box(\neg\varphi \supset p)$	1, $\breve{B}$E
3		$B(Bp \supset \Box(\neg\varphi \supset p))$	2, ml (bBR, $\Box B$R)
4	B	$Bp \supset \Box(\neg\varphi \supset p)$	3, BR
5		$\breve{B}\varphi$	4, $\breve{B}$I fixing p
6		$B\breve{B}\varphi$	BI
7		$B\breve{B}\varphi$	PP

2. Assume $\breve{b}\varphi$ and eliminate into a parametric subproof as Bp and $\Diamond(\varphi \wedge \neg p)$. These iterate by BBR and $\Diamond B$R into a B subproof, where $\breve{b}$I is applied; return by BI and PP.
3. Assume $\breve{b}\breve{b}\varphi$, and eliminate into a subproof parametric in p as Bp and $\Diamond(\breve{b}\varphi \wedge p)$. Open a subproof parametric in q, and a $\Diamond$-subproof with $\breve{b}\varphi$ as the opening formula. Apply $\breve{b}$E to obtain Bq and $\Diamond(\varphi \wedge \neg q)$, which are returned from the

$\Diamond$-subproof by appeal to $b\Box$R and $\Box\Box$R. Apply $\breve{b}$I to get $\breve{b}\varphi$, then PP twice to return this to the initial context.

4. As in 2. above, assume $\breve{b}\varphi$ and eliminate into a subproof parametric in p. Iterate by $B\Box$R and $\Diamond\Box$R into a $\Box$ subproof, apply $\breve{b}$I, then return using $\Box$I and PP.

$\Box$

The introduction of $\breve{B}\varphi$ in line 5 of lemma 4.1 takes place in a different subproof (indeed, in a different modal context) from that in which the parameter p is introduced. The elimination of a parameter (in PP) has been separated from its use in an inference rule that fixes it (here, with the implicit generalization of BI). Indeed, the need to allow such proof steps, which are clearly valid, is the rationale behind our approach of making parametric proof a separate proof rule.

A comparison to quantified logic helps illustrate how this is significant. One standard form of universal generalization can be formulated as, "if Gc has been established in a subproof from the assumption that Fc, where c is a new parameter, then the subproof may be closed and $\forall x(Fx \supset Gx)$ introduced". The employed parameter c is always dropped simultaneously with the introduction of $\forall$. If $\breve{B}$I were modeled directly on the standard form of $\forall$I from quantified logic, it would have a form such as the following: "if $\Box(\neg\varphi \supset p)$ has been proven in a subproof starting with the assumption that p is a fresh parameter s.t. Bp, then the subproof may be closed and $\breve{B}\varphi$ introduced". With this form of $\breve{B}$I, iteration principles for $\breve{B}$ would not be derivable from those for B and $\Box$.

The logic Æ [8] is defined as the smallest system which contains all tautologies, the following axioms,

Def$\Box$	$\Box\varphi \equiv (B\varphi \wedge \breve{B}\varphi)$	T	$\Box\varphi \supset \varphi$
K_B	$B(\varphi \supset \psi) \supset (B\varphi \supset B\psi)$	$K_{\breve{B}}$	$\breve{B}(\varphi \supset \psi) \supset (\breve{B}\varphi \supset \breve{B}\psi)$
$B_\Box$	$B\varphi \supset \Box B\varphi$	$\breve{B}_\Box$	$\breve{B}\varphi \supset \Box\breve{B}\varphi$
$\overline{B}_\Box$	$\neg B\varphi \supset \Box\neg B\varphi$	$\overline{\breve{B}}_\Box$	$\neg\breve{B}\varphi \supset \Box\neg\breve{B}\varphi$,

and is closed under all instances of the necessitation rule and modus ponens.[5] The axioms of Æ are all derivable in $\mathrm{N}^{\text{Æ}}$. K_B follows from BI and BE by a standard proof. $\breve{B}$I and $\breve{B}$E imply validity of the basic Def$\Box$ (lemma 2) and $K_{\breve{B}}$ (lemma 3). T, $B_\Box$, and $\overline{B}_\Box$ follow immediately from rules $\Box$E, $B\Box$R, and $b\Box$R, respectively. $b\Box$R implies $\breve{B}\varphi \supset B\breve{B}\varphi$ and $\breve{B}\varphi \supset \breve{B}\breve{B}\varphi$, and hence validity of $\breve{B}_\Box$ (lemma 4.1 and 4.2). $B\Box$R implies validity of $\overline{\breve{B}}_\Box$, from $\breve{b}\varphi \supset B\breve{b}\varphi$ and $\breve{b}\varphi \supset \breve{B}\breve{b}\varphi$ (lemma 4.3 and 4.4).

[5] In the full version of Æ, the modalities B and $\breve{B}$ are indexed with priorities. We have chosen not to present a prioritized system here, on the belief that it would contribute little to the main issues of the paper.

6 Completeness

A $N^{Æ}$*-maximal* set s is a saturated set of formulae (i.e., a consistent set with every of its proper extensions inconsistent) which satisfies the following property for each formula φ: $\breve{B}\varphi \in s$ iff $B\psi \supset \Box(\neg\varphi \supset \psi) \in s$ for every formula ψ.

Lemma 5. *Let ρ be $N^{Æ}$-consistent. Then $\rho \in s$ for a $N^{Æ}$-maximal set s.*

Proof. Maximal consistent sets are, as usual, constructed on the basis of an enumeration $\varphi_1, \varphi_2, \ldots$ of formulae. We successively construct consistent sets $S_0, S_1, S_2 \ldots$ where $S_0 = \{\rho\}$ and S_{k+1} arises from S_k by adding φ_{k+1} to S_k whenever it is consistent to do so. We do, however, require that if φ_{k+1} is of the form $\neg\breve{B}\varphi$ and is consistent with S_k, there is a propositional variable q at hand which does not occur in any formula in S_k, and that S_{k+1} in this case is $S_k \cup \{\neg\breve{B}\varphi, Bq \wedge \Diamond(\neg\varphi \wedge \neg q)\}$. To see that S_{k+1} is consistent, assume not. We then have that (‡) $S_k, Bq \wedge \Diamond(\neg\varphi \wedge \neg q) \vdash \breve{B}\varphi$. Moreover, since q does not occur in S_k, it follows by an application of $\breve{b}$E that $S_k, \neg\breve{B}\varphi \vdash Bq \wedge \Diamond(\neg\varphi \wedge \neg q)$. Together with (‡) this gives $S_k \vdash \breve{B}\varphi$, contradicting the assumption. Clearly $s = \cup_{k\geq 0} S_k$ is a saturated set which contains ρ. If $\breve{B}\varphi \in s$, $\breve{B}$E along with saturation gives that $B\psi \supset \Box(\neg\varphi \supset \psi) \in s$ for every formula ψ. If $\neg\breve{B}\varphi \in s$, there is by construction of s a q such that $Bq \wedge \Diamond(\neg\varphi \wedge \neg q) \in s$, hence s is $\mathrm{N}^{Æ}$-maximal. □

Let s be $\mathrm{N}^{Æ}$-maximal. M_s, the *canonical model wrt. s*, is $(U_s^{\Box}, U_s^{B}, U_s^{\breve{B}}, V_s)$, where $U_s^{\Box}$ is the set of $\mathrm{N}^{Æ}$-maximal sets such that $t \in U_s^{\Box}$ iff $\{\varphi \mid \Box\varphi \in s\} \subseteq t$. Likewise, $t \in U_s^{B}$ iff $\{\varphi \mid B\varphi \in s\} \subseteq t$ and $t \in U_s^{\breve{B}}$ iff $\{\varphi \mid \breve{B}\varphi \in s\} \subseteq t$. $V_s(p) = \{t \mid t \in U_s^{\Box} \text{ and } p \in t\}$. In the completeness proof below we make extensive use of the observation in the previous section to the effect that $\mathrm{N}^{Æ}$ extends Æ.

Theorem 2. *$N^{Æ}$ is complete wrt. the class of weakly bisected models.*

Proof. Let φ be consistent. By lemma 5, φ occurs in a $\mathrm{N}^{Æ}$-maximal set s. By using the introspection principles of Æ (i.e., $B_{\Box}$, $\breve{B}_{\Box}$, $\overline{B}_{\Box}$ and $\overline{\breve{B}}_{\Box}$) it is straightforward to prove that for all t in $U_s^{\Box}$ the following three properties hold: (i) $U_s^{\Box} = U_t^{\Box}$, (ii) $U_s^{B} = U_t^{B}$ and (iii) $U_s^{\breve{B}} = U_t^{\breve{B}}$, while the T-axiom entails that (iv) $s \in U_s^{\Box}$. On the basis of these properties and the fact that the three modalities are all normal (this immediate for $\Box$ and B and established in lemma 3 for $\breve{B}$), it is straightforward to prove the Truth lemma: *$M_s \vDash_t \varphi$ iff $\varphi \in t$ for each $t \in U_s^{\Box}$*. The Truth lemma entails both that M_s satisfies φ and, together with lemma 5 and the construction of $\mathrm{N}^{Æ}$-maximal sets, that M_s satisfies the $\breve{B}$-property. It remains to be shown that $U_s^{\Box} = U_s^{B} \cup U_s^{\breve{B}}$ and that M_s is weakly bisected. The former property follows easily from the Def$\Box$ scheme of Æ, while the latter follows from lemma 1. □

Corollary 1. *$N^{Æ}$ is characterized by the set of strongly bisected models.*

Proof. Since $\mathrm{N}^{Æ}$ is sound wrt. the class of weakly bisected models (Theorem 1) it is clearly sound wrt. the smaller class of strongly bisected models. Conversely, if φ is consistent, it has by Theorem 2 a weakly bisected model $M = (U, U^+, U^-, V)$. If we enrich the language with a new propositional variable q we can construct a new model M_q from M as follows: $U_q = U_q^+ \cup U_q^-$, $U_q^+ = \{\langle x, q\rangle \mid x \in U^+\}$, $U_q^- = \{\langle x, \overline{q}\rangle \mid x \in U^-\}$, $V_q(q) = U_q^+$, and $V_q(p) = \{\langle x, \cdot\rangle \mid x \in V(p)\}$ for $p \neq q$. M_q is a model of $\varphi \wedge Bq \wedge \breve{B}\neg q$ and is strongly bisected. □

7 Conjectures

The system satisfies local reduction; in particular, the introduction of $\breve{B}$ formulae does not increase deductive power, which means that the elimination rule for $\breve{B}$ is sound. To see this, consider the following pattern, where the notation $\delta[\psi/p]$ refers to the derivation resulting from replacing every occurrence of the formula p in derivation δ by the formula ψ.

$$
\begin{array}{lll}
\vdots\ \ \delta & & \\
Bp \supset \Box(\neg\varphi \supset p) & & \\
\breve{B}\varphi & \breve{B}\mathrm{I}, \text{ fixing } p & \\
B\psi \supset \Box(\neg\varphi \supset \psi) & \breve{B}\mathrm{E} & \\
\vdots\ \ \epsilon & & \\
\chi & &
\end{array}
\quad \Longrightarrow_R \quad
\begin{array}{l}
\vdots\ \ \delta[\psi/p] \\
B\psi \supset \Box(\neg\varphi \supset \psi) \\
\vdots\ \ \epsilon \\
\chi
\end{array}
$$

Because introduction of $\breve{B}\varphi$ requires a fresh proposition symbol p, i.e., a proposition about which nothing is known, derivation of $B\psi \supset \Box(\neg\varphi \supset \psi)$ can not require greater deductive resources than the formula $Bp \supset \Box(\neg\varphi \supset p)$ on which the introduction of $\breve{B}\varphi$ is based. A similar pattern holds for $\breve{b}$. On the basis of this we conjecture interesting proof-theoretical properties.

Conjecture 1. $\mathrm{N}^{Æ}$ satisfies weak normalization.

Conjecture 2. If φ is $\mathrm{N}^{Æ}$-provable from no assumptions, it has a proof in which each instance of $\breve{B}\mathrm{E}$ is instantiated with a formula ψ which is a subformula of φ (and correspondingly for the derived rule $\breve{b}\mathrm{I}$).

The natural way to prove the latter property is to exploit the structure of normal form natural deduction proofs. If it holds, $\mathrm{N}^{Æ}$ is an analytic system. From the point of view of automated deduction, this is of course a desirable property.

References

[1] Frederic Brenton Fitch. *Symbolic Logic: An Introduction*. The Ronald Press Company, New York, 1952.

[2] Melvin Fitting. *Proof Methods for Modal and Intuitionistic Logics*. Reidel, Dordrecht, 1983.

[3] I. L. Humberstone. Inaccessible worlds. *Notre Dame Journal of Formal Logic*, 24:346–352, 1983.

[4] Hector J. Levesque. All I know: A study in autoepistemic logic. *Artificial Intelligence*, 42:263–309, 1990.

[5] Frank Pfenning. Logical frameworks. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 1065–1147. Elsevier Science Publishers, 2001.

[6] Dag Prawitz. *Natural Deduction: A Proof-Theoretical Study*. Almquist & Wiksell, Stockholm, 1965.

[7] David F. Siemens, Jr. Fitch-style rules for many modal logics. *Notre Dame Journal of Formal Logic*, 18:631–636, 1977.

[8] Arild Waaler, Johan W. Klüwer, Tore Langholm, and Espen H. Lian. Only knowing with degrees of confidence. *Journal of Applied Logic*, 2005. To appear. A preprint is available at `http://folk.uio.no/johanw/ok-doc.pdf`.

Modal logic and navigational XPath: an experimental comparison

Massimo Franceschet[1,2] and Enrico Zimuel[1,2]

[1] Informatics Institute, University of Amsterdam, Amsterdam, The Netherlands
[2] Dipartimento di Scienze, Università "Gabriele D'Annunzio", Pescara, Italy

Abstract. XPath is the core retrieval language of XQuery, the official query language for XML data. We empirically compare three query evaluation strategies for the navigational fragment of XPath known as Core XPath: a bottom-up algorithm based on model checking techniques for multi-modal logic, a first top-down procedure based on a technique to eliminate XPath filters, and a second top-down procedure that takes advantage of the pre/post plane representation of an XML tree. We implement the three methods and we benchmark the resulting XPath processors using a fragment of XPathMark, a recently proposed benchmark for XPath.

1 Introduction

The *Extensible Markup Language* (XML) [1] is a popular representation language for semistructured data [2], which are data that do not necessarily possess a regular schema. The XML Path Language (XPath) [3] is a simple retrieval language for XML data. XPath in the core retrieval fragment of the XML Query Language (XQuery) [4], the standard query language for XML.

XPath and modal logic are similar in many respects. Syntactically, the XPath language contains navigational axes that closely resemble modal logic *modalities*. Semantically, XPath queries are evaluated on XML trees, which are tree-shaped *Kripke structures* whose states (nodes) are labelled with XML tags. Finally, the query evaluation problem for XPath can be reinterpreted as a *model checking problem* for multi-modal logic. XPath queries have the form $q[\alpha]$, where q is called *path* and α is called *filter*. A path is a sequence of axis steps and it is interpreted according to the following *query semantics*: it retrieves those nodes that are reachable from the current one through the axes used in q. A filter is similar to a *modal logic formula* and it is interpreted according to the standard *modal logic semantics*: it selects the current node if it satisfies the filter α. These two semantics are orthogonal, and they are mixed in the semantics of XPath. This orthogonality is the cause of the exponential complexity of a naive implementation of the semantics of XPath [5]. There exist two main strategies to avoid this exponential behavior. The first translates the path q into a modal logic formula α_q and then applies modal logic semantics and methods. The second reduces the filter α to a query q_α and then applies query semantics and tree-search

algorithms. However, it is not clear which of the two contexts, either the modal logic context or the database one, is more appropriate for the implementation of efficient evaluation algorithms for XPath.

In this paper, we neatly isolate two evaluation strategies for the navigational fragment of XPath known as Core XPath [5]. The first algorithm, that we called BottomXPath, translates a Core XPath query into a modal logic formula and then applies model checking procedures in order to retrieve the answer set of the original query. This algorithm works *bottom-up* with respect to the parse tree of the input query: it processes the sub-queries of the input query from the leaves of the parse tree up to the tree root. The second algorithm, that we called TopXPath, replaces the filters present in the input query with query paths and then applies a node retrieval procedure in order to compute the answer set of the original query. This procedure works *top-down* with respect to the parse tree of the input query: it processes the sub-queries of the input query from the root of the parse tree down to the leaves of the tree. The XQuery formal semantics requires that the result of an XPath expression is a sequence of document nodes that is document sorted and duplicate free. The *document order* corresponds to the total order of nodes given by a preorder visit of the nodes of an XML tree. With reference to the time when the sorting of the XPath expression results is performed, we specify two versions of the top-down algorithm. The first version, that we named TopXPath1, does not care about the order of the nodes in the intermediate node sequences and it document sorts the final result only. The second version, that we named TopXPath2, maintains document sorted all the intermediate node sequences and hence it does not need to sort the final result. More importantly, it takes advantage of the hypothesis that the intermediate results are document sorted in order to speed-up the XPath axis evaluation. This happens by pruning the intermediate results as much as possible before starting a new step evaluation.

A theoretical analysis of the worst-case asymptotic computational complexity of the outlined algorithms does not help in evaluating their real-life performance: all the procedures run in *linear time* with respect to the product of the size of the XML tree and the length of the query. In order to better understand the computational differences between the proposed strategies, which is our main goal in this paper, we performed an *experimental analysis*. We implemented the algorithms in standard C language and we used a fragment of the XPath benchmark XPathMark [6] to assess the empirical complexity of the discussed strategies.

This paper relates to previous work as follows. Gottlob et al. [5] propose a bottom-up polynomial-time XPath processing algorithm for full XPath, which runs in linear time for Core XPath. Moreover, they discuss a general mechanism for translating the bottom-up algorithm into a top-down one. The relation between XPath query evaluation and model checking has been investigated in [7, 8], where the authors embed a fragment of Core XPath into temporal logic and use an existing model checker to solve the query evaluation problem. The idea of maintaining document sorted the intermediate answers in order to speed-up

the axis evaluation has been proposed in [9], a work that is mostly inspired by the results in [10]. However, none of these papers has empirically compared the different strategies for XPath query evaluation. This is our main task in this work. Our bottom-up procedure BottomXPath borrows from ideas in [7], while our first top-down algorithm TopXPath1 has been inspired by the work in [5]. Finally, our second top-down algorithm TopXPath2 is an simplified version of the procedure proposed in [9].

The rest of the paper is as follows. In Section 2 we introduce XPath and we relate it to modal logic. In Sections 3 and 4 we describe the bottom-up and top-down evaluation strategies, respectively. In Section 5 we perform the experimental analysis of the proposed algorithms and in Section 6 we show how to rewrite the standard bottom-up model checking algorithm for modal logic into a more efficient top-down procedure. Finally, we sum-up in Section 7. For space reasons, we omit part of the pseudo-code. The interested reader may consult the full version of this paper [11] for more details.

2 XPath and modal logic

In this section we introduce XPath and we relate it to modal logic. Core XPath [5] is defined on a tag set Σ including the special symbol $*$. Let χ be the set of Core XPath axes. Axes are: child, parent, descendant, ancestor, descendant-or-self, ancestor-or-self, following-sibling, preceding-sibling, following, preceding, and self. We say that child is the inverse of parent and viceversa, descendant is the inverse of ancestor and viceversa, and so on. Notice that self is the inverse of itself. A *Core XPath query* is defined by the query clause of the following grammar (where $\texttt{axis} \in \chi$ and $\texttt{a} \in \Sigma$):

```
query  = /path
path   = step | step/path
step   = axis :: a | axis :: a[filter]
filter = path | filter and filter | filter or filter |
         not(filter) | (filter)
```

The Boolean XPath language extends the Core XPath language with Boolean operators at path level. More precisely, a *Boolean XPath query* is defined by the query clause of the following grammar (where $\texttt{axis} \in \chi$ and $\texttt{a} \in \Sigma$):

```
query = path | /path | query and query | query or query |
        not(query) | (query)
path  = step | path/path | path and path | path or path |
        not(path) | (path)
step  = axis :: a | axis :: a[path]
```

Notice that each Core XPath query is a Boolean XPath query, but not viceversa. For a query q, we define its length, denoted by $length(q)$, as the sum of the number of Boolean operators and the number of *atomic* steps appearing in q. An atomic step has the form `axis::a`.

Our target in this paper is Core XPath, which is the core fragment of the standard XPath [3]. We will use Boolean XPath, which is *not* a fragment of the official XPath language, as an auxiliary language only. However, it is worth noticing that all the algorithms and results in this paper easily extend to Boolean XPath.

Both Core and Boolean XPath languages are interpreted over XML trees representing XML documents. Since in the present work we are only interested in the navigational power of XPath, we assume that the XML documents we work with do not contain attributes, namespaces, processing instructions, comments, and parsed character data. An XML tree is a rooted sibling-ordered tree $T = (N, R_{\downarrow}, R_{\rightarrow}, L)$, where:

- N is a set of nodes. We denote by *root* the root node of the tree. A tree node represents an element in the XML document;
- $R_{\downarrow}$ is a binary relation on N such that $(x, y) \in R_{\downarrow}$ iff y is a child of x;
- $R_{\rightarrow}$ is a (functional) binary relation on N such that $(x, y) \in R_{\rightarrow}$ iff y is the right sibling of x;
- L is a function from Σ to the power set of N such that, for $a \in \Sigma \setminus \{*\}$, $L(a)$ is the set of nodes that are labelled with tag a, and $L(*) = N$.

Given an XML tree T, a query q in the Boolean XPath language, and a context set $C \subseteq N$, the semantics of the Boolean XPath language (and hence of the Core XPath language as well) is given by a function $\sigma(T, q, C)$ returning a subset of N. The semantic function σ is inductively defined as follows:

$$
\begin{array}{ll}
\sigma(T, \texttt{axis :: a}, C) & = \{y \in N \mid \exists x \in C.\ (x, y) \in R^T_{\texttt{axis}} \wedge y \in L(\texttt{a})\} \\
\sigma(T, \texttt{axis :: a[path]}, C) & = \{y \in N \mid y \in \sigma(T, \texttt{axis :: a}, C) \wedge \\
 & \quad \sigma(T, \texttt{path}, \{y\}) \neq \emptyset\} \\
 & \\
\sigma(T, \texttt{path}_1/\texttt{path}_2, C) & = \sigma(T, \texttt{path}_2, \sigma(T, \texttt{path}_1, C)) \\
\sigma(T, \texttt{path}_1 \texttt{ and } \texttt{path}_2, C) & = \sigma(T, \texttt{path}_1, C) \cap \sigma(T, \texttt{path}_2, C) \\
\sigma(T, \texttt{path}_1 \texttt{ or } \texttt{path}_2, C) & = \sigma(T, \texttt{path}_1, C) \cup \sigma(T, \texttt{path}_2, C) \\
\sigma(T, \texttt{not(path)}, C) & = N \setminus \sigma(T, \texttt{path}, C) \\
 & \\
\sigma(T, /\texttt{path}, C) & = \sigma(T, \texttt{path}, \{root\}) \\
\sigma(T, \texttt{query}_1 \texttt{ and } \texttt{query}_2, C) & = \sigma(T, \texttt{query}_1, C) \cap \sigma(T, \texttt{query}_2, C) \\
\sigma(T, \texttt{query}_1 \texttt{ or } \texttt{query}_2, C) & = \sigma(T, \texttt{query}_1, C) \cup \sigma(T, \texttt{query}_2, C) \\
\sigma(T, \texttt{not(query)}, C) & = N \setminus \sigma(T, \texttt{query}, C)
\end{array}
$$

The relation $R^T_{\texttt{axis}}$ is a binary relation on N corresponding to the specified axis. For instance, $R^T_{\texttt{child}} = R_{\downarrow}$ and $R^T_{\texttt{descendant}} = (R_{\downarrow})^+$. Finally, the *answer set* of the query q with respect to the tree T is equal to $\sigma(T, q, N)$.

We assume that the reader is familiar with modal logic [12]. We define the *truth set* of a modal formula α with respect to a model M as the set of all worlds x such that $M, x \models \alpha$. The intimate relation between XPath and modal logic is explicated in the definition of Core XPath logic. *Core XPath logic* is an instance of multi-modal logic in which there is one modality for each axis in XPath. It

is defined over a set of labels Σ including the special symbols denoted by $*$ and root. A model for Core XPath logic is a relational structure corresponding to an XML tree. More precisely, given an XML tree $T = (N, R_{\downarrow}, R_{\rightarrow}, L)$, the corresponding model for Core XPath logic is $M_T = (N, \{R^T_{\texttt{axis}}\}_{\texttt{axis}\in\chi}, L)$, where $L(\texttt{root})$ is a singleton containing the root node of T. In Section 3 we will show how to translate Core XPath queries into Core XPath formulas. Moreover, in Section 6 we will map Core XPath formulas to Boolean XPath queries.

3 A bottom-up evaluation strategy

In this section we give an efficient bottom-up algorithm, called BottomXPath, to evaluate a Core XPath query. The algorithm is based on a technique that in the logic context is known as model checking [13]. The *model checking problem* is the following question: given a model M and a formula α, retrieve the truth set of α with respect to M. A *model checker* is an algorithm that solves the model checking problem.

We start by embedding Core XPath queries into Core XPath formulas. A *filter expression* in XPath is defined by the filter clause of the Core XPath grammar given in Section 2. We define a translation ω from XPath filter expressions into Core XPath formulas as follows (if filter is empty in the first two clauses below, then the corresponding conjunct is missing):

$$\begin{aligned}
&\omega(\texttt{axis :: a[filter]}) &&= \langle\texttt{axis}\rangle(\texttt{a} \wedge \omega(\texttt{filter}))\\
&\omega(\texttt{axis :: a[filter]/path}) &&= \langle\texttt{axis}\rangle(\texttt{a} \wedge \omega(\texttt{filter}) \wedge \omega(\texttt{path}))\\
&\omega(\texttt{filter}_1\ \texttt{and}\ \texttt{filter}_2) &&= \omega(\texttt{filter}_1) \wedge \omega(\texttt{filter}_2)\\
&\omega(\texttt{filter}_1\ \texttt{or}\ \texttt{filter}_2) &&= \omega(\texttt{filter}_1) \vee \omega(\texttt{filter}_2)\\
&\omega(\texttt{not(filter)}) &&= \neg\omega(\texttt{filter})
\end{aligned}$$

We now define a translation τ from Core XPath queries into Core XPath formulas as follows (if filter is empty in the clauses below, then the corresponding conjunct is missing):

$$\begin{aligned}
&\tau(\texttt{/axis :: a[filter]}) &&= \texttt{a} \wedge \omega(\texttt{filter}) \wedge \langle\texttt{axis}^{-1}\rangle\ \texttt{root}\\
&\tau(\texttt{path/axis :: a[filter]}) &&= \texttt{a} \wedge \omega(\texttt{filter}) \wedge \langle\texttt{axis}^{-1}\rangle\ \tau(\texttt{path})
\end{aligned}$$

where $\texttt{axis}^{-1}$ is the inverse of axis. Notice that the length of $\tau(q)$ is *linear* in the length of q. We have the following:

Theorem 1. *Let q be a Core XPath query and T be an XML tree. Then, the answer set of q with respect to T is the truth set of $\tau(q)$ with respect to the corresponding model M_T.*

By virtue of Theorem 1, the answer set for a Core XPath query equals to the truth set for the corresponding Core XPath formula. Hence, we can solve the query evaluation problem in terms of the model checking problem by using a model checker as a query processor. BottomXPath is a bottom-up model checker for Core XPath logic. It inputs an XML tree T (and not a multi-modal model)

and a Core XPath formula α and processes sub-formulas of α in increasing length order. The algorithm is similar to a model checker for the temporal logic CTL (see, e.g., [13]); instead of CTL temporal operators, BottomXPath checks XPath axes. The tree T is represented as follows: each node is an object composed of a field *pre* containing the order of the node in a preorder visit of the tree, a field p containing a pointer to the parent of the node, or NIL if the node is the root, a field c containing a pointer to the first child of the node, or NIL if the node is a leaf, a field r containing a pointer to the right sibling of the node, or NIL if the node is the last sibling, a field l containing a pointer to the left sibling of the node, or NIL if the node is the first sibling, and a field *tag* containing the tag of the XML element that the node represents.

BottomXPath uses a subprocedure EvalAxis. The latter inputs a tree T, and axis `axis` and a formula β. For each node $x \in N$, the procedure EvalAxis labels x with $\langle \texttt{axis} \rangle \beta$ if, and only if, there exists a node $y \in N$ reachable from x trough the relation induced by `axis` such that y is labelled with β. EvalAxis takes advantage of a Boolean matrix A, where rows represent formulas and columns represent nodes, in order to label nodes with formulas that are true at them. Finally, EvalAxis uses the auxiliary procedure LabelDescendants in order to label the descendant nodes of a given node with a given formula. For space reasons, we only show the implementation of `descendant` and `ancestor` axes of EvalAxis. See [11] for the entire code.

```
 1: EvalAxis(T, axis, β)
 2: case
 3: • ...
 4: • axis = descendant
 5: for all x ∈ N do
 6:   if A(β, x) = 1 then
 7:     y ← p[x]
 8:     while y ≠ NIL and A(⟨descendant⟩β, y) = 0 do
 9:       A(⟨descendant⟩β, y) ← 1
10:       y ← p[y]
11:     end while
12:   end if
13: end for
14: • axis = ancestor
15: for all x ∈ N do
16:   if A(β, x) = 1 and A(⟨ancestor⟩β, x) = 0 then
17:     y ← c[x]
18:     while y ≠ NIL do
19:       LabelDescendant(⟨ancestor⟩β, y)
20:       y ← r[y]
21:     end while
22:   end if
23: end for
24: end case
```

Let n be the size of the input tree T and k the length of the input formula α. The procedure EvalAxis runs in $O(n)$ and hence the cost of BottomXPath amounts to $O(k \cdot n)$. The whole bottom-up evaluation algorithm for Core XPath is as follows:

1. translate q into $\tau(q)$;
2. run BottomXPath on T and $\tau(q)$;
3. sort, in document order, the result of BottomXPath.

The complexity of the translation step is $O(k)$ and the call to BottomXPath costs $O(k \cdot n)$. Since nodes are integers from 1 to n, we can sort the result with a linear-time sorting algorithm like CountingSort. Hence, the overall worst-case complexity is $O(k \cdot n)$.

4 A top-down evaluation strategy

In this section we give two efficient top-down algorithms, called TopXPath1 and TopXPath2, to evaluate Core XPath queries. Both the algorithms replace the filters present in the input query with query paths and then apply a node retrieval procedure in order to compute the answer set of the original query.

We first show how to get rid of filters. The following *inverting translation* ι inputs a filter expression in the Core XPath language and returns its inverse in the Boolean XPath language.

$$
\begin{array}{ll}
\iota(\texttt{axis :: a}) & = \texttt{self :: a}/\texttt{axis}^{-1}\texttt{ :: *} \\
\iota(\texttt{axis :: a[filter]}) & = \iota(\texttt{filter})/\iota(\texttt{axis :: a}) \\
\iota(\texttt{step/path}) & = \iota(\texttt{path})/\iota(\texttt{step}) \\
\iota(\texttt{filter}_1\texttt{ and filter}_2) & = \iota(\texttt{filter}_1)\texttt{ and }\iota(\texttt{filter}_2) \\
\iota(\texttt{filter}_1\texttt{ or filter}_2) & = \iota(\texttt{filter}_1)\texttt{ or }\iota(\texttt{filter}_2) \\
\iota(\texttt{not(filter)}) & = \texttt{not}(\iota(\texttt{filter}))
\end{array}
$$

Notice that $\iota(q)$ is a Boolean XPath query *without* filters. We now define a translation υ from Core XPath queries into Boolean XPath queries *without* filters:

$$
\begin{array}{ll}
\upsilon(/\texttt{axis :: a}) & = /\texttt{axis :: a} \\
\upsilon(/\texttt{axis :: a[filter]}) & = (/\texttt{axis :: a and }\iota(\texttt{filter})) \\
\upsilon(\texttt{step/path}) & = \upsilon(\texttt{step})/\upsilon(\texttt{path})
\end{array}
$$

Notice that the length of $\upsilon(q)$ is linear in the length of q. We have the following:

Theorem 2. *Let T be an XML tree and q be a Core XPath query. Then, q and $\upsilon(q)$ have the same answer set.*

4.1 A first top-down algorithm

In this section we propose a first top-down strategy, called TopXPath1, to evaluate Core XPath queries. With respect to the tree data structure described in

Section 3, we assume here that an additional field called *count* is added to the object representation of each node of the tree. The new field is used to record whether the node has been visited or not during a step evaluation. TopXPath1 does not care about the order of the nodes in the intermediate context sets and it sorts the final result only. TopXPath1 inputs an XML tree T, a Boolean XPath query without filters q, and a context set C. It uses a sub-procedure ProcessPath1 to elaborate query paths, which in turn calls a procedure ProcessStep1 to evaluate query steps. In particular, the procedure ProcessStep1(T, `axis`, `a`, C) elaborates the step `axis` :: `a` on the tree T with context set C, according to the XPath semantics. In order avoid walking on the same node twice, the procedure checks the *count* field of the node's object. This field is initialized to 0 for each node when TopXPath1 starts. The global variable k is also initialized to 0 and it is incremented by one at each step evaluation performed with ProcessStep1. When a node is visited during a step evaluation, its *count* field is assigned to the value contained in k. Hence, during the k-th step evaluation, all nodes that has been already visited in that step evaluation have their count field set to k, while the count field of the unexplored nodes is less than k. This method avoids the costly resetting of the count field at each step evaluation. Finally, ProcessStep1 uses an auxiliary procedure RetrieveDescendants to retrieve the descendant nodes of a given node that are labelled with a given tag. We show the implementation of TopXPath1, of ProcessPath1, and of `descendant` and `ancestor` axes of ProcessStep1. See [11] for the entire code.

```
1: TopXPath1(T, q, C)
2: k ← 0
3: for x ∈ N do
4:    count[x] ← 0
5: end for
6: case
7: • q = query1 and query2
8: return TopXPath1(T, query1, C) ∩ TopXPath1(T, query2, C)
9: • q = query1 or query2
10: return TopXPath1(T, query1, C) ∪ TopXPath1(T, query2, C)
11: • q = not(query)
12: return N \ TopXPath1(T, query, C)
13: • q = /path
14: return ProcessPath1(T, path, {root(T)})
15: • q = path
16: return ProcessPath1(T, path, N)
17: end case
```

```
1: ProcessPath1(T, p, C)
2: case
3: • p = path1 and path2
4: return ProcessPath1(T, path1, C) ∩ ProcessPath1(T, path2, C)
```

```
• p = path1 or path2
return ProcessPath1(T, path1, C) ∪ ProcessPath1(T, path2, C)
• p = not(path)
return N \ ProcessPath1(T, path, C)
• p = step/path
return ProcessPath1(T, path, ProcessPath1(T, step, C))
• p = axis :: a
return ProcessStep1(T, axis, a, C)
end case
```

```
ProcessStep1(T, axis, a, C)
k ← k + 1
R ← ∅
case
• ...
• axis = descendant
for x ∈ C do
  if count[x] < k then
    count[x] ← k
    y ← c[x]
    while y ≠ NIL do
      R ← R ∪ RetrieveDescendants(y, a)
      y ← r[y]
    end while
  end if
end for
• axis = ancestor
for x ∈ C do
  y ← p[x]
  while y ≠ NIL and count[y] < k do
    count[y] ← k
    if a = ∗ or tag[y] = a then
      R ← R ∪ {y}
    end if
    y ← p[y]
  end while
end for
end case
```

Let n be the size of the input tree T and k the length of the input formula α. The procedure ProcessStep1 costs $O(n)$ and hence the evaluation algorithm TopXPath1 runs in $O(k \cdot n)$. The whole top-down evaluation algorithm for Core XPath is as follows:

1. translate q into $\upsilon(q)$;
2. run TopXPath1 on T and $\upsilon(q)$;

3. sort, in document order, the result of TopXPath1.

The complexity of the translation step is $O(k)$ and the call to TopXPath1 costs $O(k \cdot n)$. The sorting can be performed in linear time. Hence, the overall worst-case complexity is $O(k \cdot n)$.

4.2 A second top-down algorithm

In this section we propose a second top-down strategy, called TopXPath2, to evaluate Core XPath queries. With respect to the tree data structure described in Section 3, we assume here that two additional fields are added to the object representation of each node of the tree: a field called *count* that, as in TopXPath1, is used to record whether the node has been visited or not during a step evaluation, and a field called *post* containing the order of the node in a postorder visit of the tree. TopXPath2 uses a sub-procedure ProcessPath2 which in turns calls ProcessStep2, as done for TopXPath1. ProcessStep2 uses an auxiliary procedure Descendant to retrieve the descendant nodes of a given node that are labelled with a given tag. Moreover, it uses the following auxiliary list procedures, where C and L are double-linked lists and x is a node: NewList(), that initializes a new list, DelFirst(C), that deletes and returns the first element of C, DelLast(C), that deletes and returns the last element of C, AddAfter(C, x), that appends x to C, AddListAfter(C, L), that appends L to C, AddBefore(C, x), that adds x in front of C, AddListBefore(C, L), that adds L in front of C, First(C) that returns the first element of C, Last(C) that returns the last element of C. All these procedures can be implemented in constant time. TopXPath2 differs from TopXPath1 since it maintains document sorted the intermediate context sets. Moreover, it exploits the sorted contexts to speed-up the XPath axis evaluation by pruning the context sets as much as possible before starting each step evaluation. By maintaining both the preorder and the postorder ranks for each node, TopXPath2 implicitly represents an XML tree as a bi-dimensional plane, called the *pre/post plane* in [10]. Each node x is encoded by the point $(pre(x), post(x))$. A nice feature of this encoding is that, for each node x, the top-right (respectively, bottom-left) quadrant of x contains all the following (respectively, preceding) nodes of x, and the bottom-right (respectively, top-left) quadrant of x contains all the descendant (respectively, ancestor) nodes of x. Hence, given two arbitrary nodes x and y, we can check in constant time the relative position of y with respect to x. As an example, consider the cases of `following` and `preceding` axes. By exploiting the pre/post plane properties, the context set can always be reduced to a singleton (see the code below). Finally, TopXpath2 takes advantage, when necessary, of the counting technique described in Section 4.1 to avoid the exploration of the same tree zones twice. For space reasons, we only show the implementation of `following` and `preceding` axes of ProcessStep2. See [11] for the entire code. Notice that, if the input context set is document sorted, then the results of `following` and `preceding` axes are document sorted as well.

1: ProcessStep2(T, `axis`, `a`, C)

```
2: k ← k + 1
3: R ← ∅
4: case
5: • ...
6: • axis = following
7: L ← NewList()
8: if C ≠ ∅ then
9:    x ← DelFirst(C)
10:   while post[First(C)] < post[x] do
11:      x ← DelFirst(C)
12:   end while
13:   while x ≠ NIL do
14:      y ← r[x]
15:      while y ≠ NIL do
16:         AddListAfter(L, Descendants(y, a))
17:         y ← r[y]
18:      end while
19:      x ← p[x]
20:   end while
21: end if
22: return L
23: • axis = preceding
24: L ← NewList()
25: x ← Last(C)
26: while x ≠ NIL do
27:   M ← NewList()
28:   y ← l[x]
29:   while y ≠ NIL do
30:      AddListAfter(M, Descendants(y, a))
31:      y ← l[y]
32:   end while
33:   AddListBefore(L, M)
34:   x ← p[x]
35: end while
36: return L
37: end case
```

Let n be the size of the input tree T and k the length of the input formula α. ProcessStep2 runs in $O(n)$ and TopXPath2 in $O(k \cdot n)$. In this case, the whole top-down evaluation algorithm for Core XPath is as follows:

1. translate q into $\upsilon(q)$;
2. run TopXPath2 on T and $\upsilon(q)$.

The complexity of the translation step is $O(k)$ and the call to TopXPath2 costs $O(k \cdot n)$. Since TopXPath2 maintains sorted the intermediate context sets,

the result of TopXPath2 is already sorted. Hence, the overall worst-case complexity is $O(k \cdot n)$.

5 Experimental analysis

The three proposed algorithms have the same asymptotic worst-case complexity. In order to better understand the computational differences between the proposed strategies, we performed an *experimental analysis*. We implemented the algorithms in standard C language and we used a fragment of the XPath benchmark XPathMark [6] to assess the empirical complexity of the discussed strategies. In this section, we briefly report about this analysis. The source code (released under the GNU General Public License), the executable programs (for Gnu/Linux systems), and additional experimental data and plots (including a comparison with XQuery processor Saxon [14], which is outside of the scope of this paper) are available at the website `http://www.zimuel.it/xpath`.

Our experiments were run on an AMD Sempron 1.7 GHz, with 1 GB RAM, running Debian Gnu/Linux version 2.6.10. All the times are response CPU times in seconds. We ran tests using a variety of XML documents and XPath queries. The XML documents were generated using the XML benchmarking program XMark [15]. We used a document series of 11 XML documents corresponding to the following sizes in MB: 0.116, 0.212, 0.468, 0.909, 1.891, 3.751, 7.303, 15.044, 29.887, 59.489, 116.517. As for the benchmark queries, we took advantage of a fragment of the XPath benchmark XPathMark [6]. The benchmark queries we used are the following (notice that Q7, Q8, Q9, and Q10 are new queries not belonging to XPathMark):

Q1 *The keywords in annotations of closed auctions*

```
/child::site/child::closed_auctions/child::closed_auction
/child::annotation/child::description/child::parlist
/child::listitem/child::text/child::keyword
```

Q2 *All the keywords*

```
/descendant::keyword
```

Q3 *The keywords in a paragraph item*

```
/descendant-or-self::listitem/descendant-or-self::keyword
```

Q4 *The (either North or South) American items*

```
/child::site/child::regions/child::*/child::item
```

```
[parent::namerica or parent::samerica]
```

Q5 *The paragraph items containing a keyword*

```
/descendant::keyword/ancestor::listitem
```

Q6 *The mail containing a keyword*

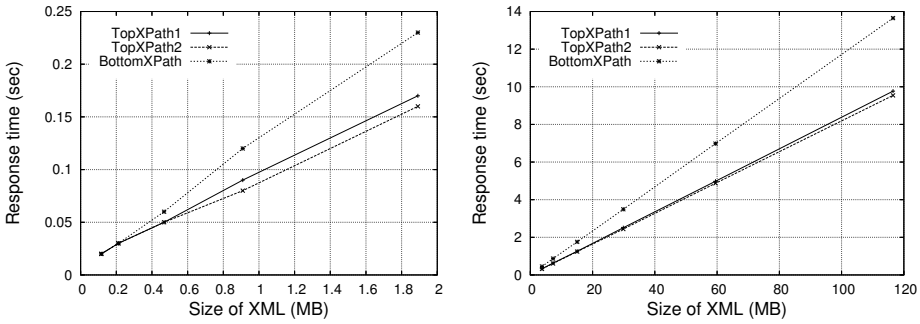


Fig. 1. Query response time (averaged over the benchmark set)

```
/descendant::keyword/ancestor-or-self::mail
```

Q7 *The last bidder of all open auctions*

```
/child::site/child::open_auctions/child::open_auction
/child::bidder[not(following-sibling::bidder)]
```

Q8 *The first bidder of all open auctions*

```
/child::site/child::open_auctions/child::open_auction
/child::bidder[not(preceding-sibling::bidder)]
```

Q9 *The last item of the document*

```
/child::site/child::regions/child::*
/child::item[not(following::item)]
```

Q10 *The first item of the document*

```
/child::site/child::regions/child::*
/child::item[not(preceding::item)]
```

Q11 *People having an address and either a phone or a homepage*

```
/child::site/child::people/child::person
[child::address and (child::phone or child::homepage)]
```

Q12 *People having no homepage*

```
/child::site/child::people/child::person[not(child::homepage)]
```

In order to perform the evaluation, we applied the benchmarking methodology described in [6]. In particular, we used the following measures:

– given a query q and a document d, the *query response time* is the time taken by an algorithm to give the answer for the query q on the document d including all the phases of the elaboration (e.g., parsing of the document, processing of the query, serialization of the results).

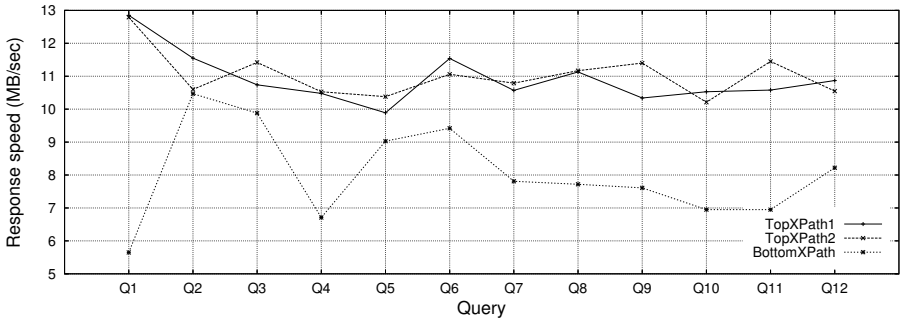


Fig. 2. Query response speed (averaged over the document series)

- Given a query q and a document d, the *query response speed* is defined as the size of the document d divided by the response time for query q and document d. The measure unit is, for instance, MB/sec.
- Given a query q and two documents d_1 and d_2, where the size of d_2 is bigger than the size of d_1, the *data scalability factor* is defined as v_1/v_2, where v_1 is the query response speed of q on d_1 and v_2 is the query response speed of q on d_2.

We say that the data scalability is *sub-linear* whenever the data scalability factor is lower than 1, it is *super-linear* whenever the data scalability factor is higher then 1, and it is *linear* whenever the data scalability factor is equal to 1. Usually, sub-linear and linear scalability indicate that the query processor performs well when the size of the database is increased.

Figure 1 depicts, for each document in the series, the average query response time for the entire benchmark (left plot for small files, right plot for bigger ones), while Figure 2 illustrates, for each query in the benchmark, the average query response speed for the entire document series. Finally, Figure 3 plots, for each document in the series, the average data scalability factors for the entire benchmark.

In the following, we comment the outcomes of our experimental evaluation:

- The response times of the two top-down strategies are very close, with TopXPath2 slightly faster than TopXPath1. This tells us that the approach of maintaining the context sequences document sorted at any time does not pay off in terms of response time.
- The top-down strategy is more efficient than the bottom-up one (about 30% faster, and the difference increases as the size of the data increases). The gap is bigger in the case of queries that do not need to explore big portions of the document tree in order to compute the query answer (like Q1 and Q4), while the response times of the two strategies are similar in the case of queries that need to visit the entire document (like Q2) . This phenomenon can be

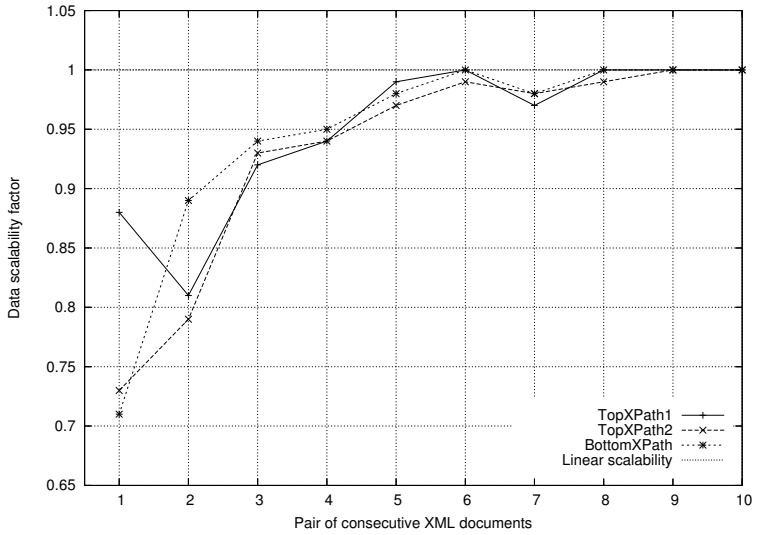


Fig. 3. Data scalability factors (averaged over the benchmark set)

explained as follows: the bottom-up algorithm visits the entire document tree for each sub-query of the main query, while the top-down procedure explores only the tree zones that are relevant for the evaluation of the query.

– All the three XPath processors scale-up linearly with respect to the size of the XML data. This confirms the linear-time complexity of the implemented algorithms.

6 Back to modal logic

What do modal logicians get out of this paper at the end of the day? In Section 3 we gave a translation τ from the Core XPath query language (a fragment of Boolean XPath) to the Core XPath modal language. This mapping can easily be extended to Boolean XPath by adding distributive clauses for the Boolean operators, e.g., $\tau(\texttt{query}_1 \texttt{ and } \texttt{query}_2) = \tau(\texttt{query}_1) \wedge \tau(\texttt{query}_2)$. The natural question is then: is there an inverse translation from Core XPath modal formulas to Boolean XPath queries? Yes, there is. It is as follows (where `a` is an atomic proposition different from `root`):

$$
\begin{array}{ll}
\chi(\texttt{root}) & = \texttt{self} :: *[\texttt{not}(\texttt{parent} :: *)] \\
\chi(\texttt{a}) & = \texttt{self} :: \texttt{a} \\
\chi(\langle \texttt{axis} \rangle \alpha) & = \chi(\alpha)/\langle \texttt{axis}^{-1} \rangle :: * \\
\chi(\alpha \wedge \beta) & = (\chi(\alpha) \texttt{ and } \chi(\beta)) \\
\chi(\alpha \vee \beta) & = (\chi(\alpha) \texttt{ or } \chi(\beta)) \\
\chi(\neg \alpha) & = \texttt{not}(\chi(\alpha))
\end{array}
$$

Notice that the length of $\chi(\alpha)$ is *linear* in the length of α. We have the following:

Theorem 3. *Let α be a Core XPath modal formula and T be an XML tree. Then, the truth set of α with respect to the corresponding model M_T is the answer set of $\chi(\alpha)$ with respect to T.*

The above theorem has some interesting consequences. Putting together Theorems 1 and 3, we have that the model checking problem for Core XPath modal logic and the query evaluation problem for Boolean XPath query language are polynomially inter-reducible. In particular, this gives us the following *top-down* algorithm to solve the model checking problem for Core XPath modal logic: translate a modal formula α into the corresponding query $\chi(\alpha)$ and then use a top-down strategy as described in Section 4 to retrieve the answer set of $\chi(\alpha)$. This paper empirically proves that a top-down evaluation strategy is less expensive than a bottom-up one, because it browses only the relevant parts of the underlying model and not more. However, model checking algorithms for modal and temporal logics typically are bottom-up procedures [13]. The top-down query evaluation strategies described in Section 4 can be turned into an efficient top-down model checking procedure for Core XPath modal logic.

7 Conclusion

We implemented three evaluation strategies for the navigational fragment of XPath and we benchmarked the resulting XPath processors using a fragment of XPathMark, a recently proposed benchmark for XPath. The main outcomes of our investigation are (i) a top-down evaluation approach is faster than a bottom-up one, and (ii) the pre/post plane optimizations for XPath query evaluation are as efficient as tree-search algorithms over the tree modeling the XML document. We also gave a top-down model checker for Core XPath modal logic which empirically performs better than the standard bottom-up one.

It is worth pointing out that a bottom-up strategy outputs much more information than a top-down strategy. In particular, the bottom-up model checking-based procedure computes the answer set for *each* sub-query of the input query, while the top-down routine retrieves only those nodes belonging to the answer of the input query. This feature of the bottom-up approach may in fact become a benefit whenever the answer set for the sub-queries of the input query is relevant. Consider for instance a query processor that is queried many times possibly by different users. It is not unlikely that similar queries are posed at different times. In such a case, a bottom-up strategy may easily reuse the results computed for common sub-queries (in a dynamic programming fashion), while a top-down strategy must re-compute the result for each new query from scratch. As a future work, we would like to compare the performace of the bottom-up and top-down approaches in a multi-query environment.

Acknowledgments

This research was supported by the Netherlands Organization for Scientific Research (NWO) under project number 612.000.207.

References

1. World Wide Web Consortium: Extensible Markup Language (XML). http://www.w3.org/XML (1998)
2. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann Publishers (2000)
3. World Wide Web Consortium: XML Path Language (XPath) Version 1.0. http://www.w3.org/TR/xpath (1999)
4. World Wide Web Consortium: XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery (2005)
5. Gottlob, G., Koch, C., Pichler, R.: Efficient algorithms for processing XPath queries. ACM Transactions on Database Systems (TODS) **30**(2) (2005) 444–491
6. Franceschet, M.: XPathMark: an XPath benchmark for the XMark generated data. In: Proceedings of the International XML Database Symposium (XSym). Volume 3671 of LNCS. (2005) 129–143 http://www.science.uva.nl/~francesc/xpathmark.
7. Afanasiev, L., Franceschet, M., de Rijke, M., Marx, M.: CTL model checking for processing simple XPath queries. In: Proceedings of the International Symposium on Temporal Representation and Reasoning (TIME), IEEE Computer Society Press (2004) 117–124
8. Hartel, P.: A trace semantics for positive Core XPath. In: Proceedings of the International Symposium on Temporal Representation and Reasoning (TIME), IEEE Computer Society Press (2005) 103–112
9. Hidders, J., Michiels, P.: Efficient XPath axis evaluation for DOM data structures. In: Workshop on Programming Language Technologies for XML (PLAN-X). (2004) 54–63
10. Grust, T., van Keulen, M., Teubner, J.: Staircase join: Teach a relational DBMS to watch its (axis) steps. In: Proceedings of International Conference on Very Large Data Bases (VLDB), Morgan Kaufmann Publishers (2003) 524–525
11. Franceschet, M., Zimuel, E.: Modal logic and navigational XPath: an experimental comparison. Technical Report R-2005-001, Dipartimento di Scienze, Università "G. D'Annunzio", Pescara (2005) http://www.sci.unich.it/~francesc/pubs/tr05.pdf.
12. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press (2001)
13. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge, Massachusetts (1999)
14. Kay, M.H.: Saxon. An XSLT and XQuery processor. http://saxon.sourceforge.net (2005)
15. Schmidt, A.R., Waas, F., Kersten, M.L., Carey, M.J., Manolescu, I., Busse, R.: XMark: A Benchmark for XML Data Management. In: Proceedings of the International Conference on Very Large Data Bases (VLDB). (2002) 974–985 http://monetdb.cwi.nl/xml/.

ATEL with Common and Distributed Knowledge is ExpTime-Complete

Dirk Walther
University of Liverpool
dirk@csc.liv.ac.uk

Abstract

Alternating-time Temporal Logic (ATL) of Alur, Henzinger, and Kupferman is increasingly widely applied in formal system specification and verification of distributed systems and game-like multi-agent systems. The extension Alternating-time Temporal Epistemic Logic (ATEL) of van der Hoek and Wooldridge of ATL with knowledge modalities widens the scope of reasoning over multi-agent systems with epistemic notions. In this paper, the complexity of the satisfiability problem for ATEL with epistemic operators for common and distributed knowledge is investigated and settled at ExpTime-complete. Thus ATEL has the same complexity as ATL.

1 Introduction

Alternating-time Temporal Logic (ATL) introduced by Alur, Henzinger, and Kupferman [1] has become a successful tool for formal system specification and verification of distributed systems and game-like multi-agent systems. ATL provides a *path quantifier* $\langle\langle\ \rangle\rangle$ that is parametrized by a coalition A of agents and *temporal operators* $\bigcirc$ ('next'), $\Box$ ('always'), and $\mathcal{U}$ ('until') where, e.g., $\langle\langle A\rangle\rangle\Box\psi$ states that the agents in coalition A can cooperate to ensure that always ψ. ATL generalizes the Computational Tree Logic (CTL) since the path quantifiers A ("on all paths") and E ("on some path") of CTL can be expressed by the ATL path quantifiers $\langle\langle\emptyset\rangle\rangle$ ("the empty coalition can cooperate") and $\langle\langle\Sigma\rangle\rangle$ ("the grand coalition of all agents can cooperate").

ATL satisfiability was investigated in [11, 13]: [11] proves an ExpTime upper bound for ATL satisfiability under the assumption that the set of available agents is fixed in advance; [13] shows containment in ExpTime even when the set of agents occurring in formulas is drawn from an arbitrary (possibly infinite) set. The ExpTime lower bound easily follows from CTL.

Soon after its introduction, ATL was extended with knowledge modalities to Alternating-time Temporal Epistemic Logic (ATEL) by van der Hoek and Wooldridge [14]. For an illustration of ATEL's expressiveness, consider the following formula that states: if agent a knows a secret, then she can keep it to herself, i.e., a can make sure that her secret will never be known by another

agent b:

$$K_a secret \wedge \neg K_b secret \rightarrow \langle\langle a\rangle\rangle \Box \neg K_b secret.$$

The expressivity of ATEL depends on the *epistemic operators* available. In this paper, we focus on the epistemic operators of the very expressive language $\mathrm{S5}^{C,D}$ from [8] which contains epistemic operators K_a ("agent a knows"), E_A ("every agent in A knows"), C_A ("the agents in A have *common knowledge*"), and D_A ("the agents in A have *distributed knowledge*") for each agent a and coalition A of agents. Intuitively, a coalition A of agents has common knowledge of a fact φ if every agent in A knows that every agent in A knows ... that φ is true; A has distributed knowledge of φ if the combined individual knowledge of all agents in A implies φ. In what follows, we denote with ATEL the logic ATL extended with epistemic operators K_a, E_A, C_A, and D_A for each agent a and coalition A of agents.

In [6], Goranko, Jamroga, and van Drimmelen axiomatized ATEL without operators for distributed knowledge and conjectured an axiomatization of ATEL with distributed knowledge operators by extending the axiomatic system for ATL [7]. From this axiomatization, it becomes clear that there is no interaction between the temporal and epistemic modalities in ATEL, i.e., ATEL is an independent fusion in the sense of [5] of the logics ATL and $\mathrm{S5}^{C,D}$. Although both logics, ATL [11, 13] and $\mathrm{S5}^{C,D}$ [8], are ExpTime-complete, it does not follow from general results on fusions of modal logics that ATEL is ExpTime-complete as well. For instance, an increase in computational complexity when taking fusions can be observed with the fusion of S5 with itself: while S5 is in NP [8], the fusion is PSpace-hard [5]. From general results on complexity transfer for fusions [3, 2], it only follows that ATEL is in 2-ExpTime. So far, it was an open problem whether satisfiability for ATEL is also in ExpTime. In fact, because of the different character of ATL and $\mathrm{S5}^{C,D}$, it seemed that their fusion is not in ExpTime. In this paper, however, we show that satisfiability of ATEL (with common and distributed knowledge) is ExpTime-complete using a type elimination construction along the lines of [13] combined with a technique of [8]. This also shows that adding epistemic operators for common knowledge and distributed knowledge to ATL does not yield an increase in computational complexity. The result is presented in a compact way due to space limitations; more details and full proofs can be found in the accompanying technical report [12].

When formulating satisfiability problems for ATEL some care is needed as it was analyzed in [13]. In particular, the range of semantic structures, over which a formula is to be interpreted, needs to be specified. Such a range can be determined by allowing for a certain number of agents to be present in the semantic structures. To see that allowing for different sets of agents to be present can influence satisfiability, look at the following ATL-formula (adapted from [10, p.47]):

$$\neg\langle\langle a\rangle\rangle \bigcirc p \wedge \neg\langle\langle a\rangle\rangle \bigcirc q \wedge \langle\langle a\rangle\rangle \bigcirc (p \vee q).$$

This formula expresses the fact that, in the next state, agent a cannot make p true, and cannot make q true; but it can make either p or q true. Now the question is whether this formula is satisfiable. The answer is that it is only satisfiable in a semantic structure for more than one agent, and not satisfiable with merely one available agent. Thus the number of agents present in a structure is important for determining satisfiability of a formula in this structure. With these concerns in mind, three variants of the satisfiability problem for ATL were suggested in [13] depending on the possibilities for the number of agents to occur in semantic structures. In this paper, however, we concentrate only on one of these problems; the other two satisfiability problems can be reduced to this:

> *Satisfiability over formula-defined sets of agents:* Given a formula φ, is φ satisfiable in a structure for exactly the agents which occur in φ?

This paper is organized as follows. Section 2 introduces ATEL by giving its syntax, defining alternating epistemic transition systems as semantic structures, and, by giving the semantics, stating how formulas are interpreted over these transition systems. In Section 3, we show that ATEL is ExpTime-complete. This is done by defining a decision procedure for ATEL that runs in exponential time using a combination of a type elimination construction introduced in [13] with the technique of [8]. The final section states some concluding remarks and gives ideas for further research.

2 ATEL

In this section, Alternating-time Temporal Epistemic Logic [14] with epistemic operators for common and distributed knowledge is defined. First consider the syntax.

Definition 2.1 [ATEL Syntax] Let $\mathbf{\Pi}$ be a countable infinite set of *atomic propositions* and $\mathbf{\Sigma}$ a countable infinite set of *agents*. A *coalition* is a finite set $A \subseteq \mathbf{\Sigma}$ of agents. The set of ATEL-formulas is the smallest set containing the following:

- p for $p \in \mathbf{\Pi}$;
- $\neg\varphi$, $\varphi \vee \psi$ for ATEL-formulas φ and ψ;
- $\langle\langle A\rangle\rangle\bigcirc\varphi$, $\langle\langle A\rangle\rangle\Box\varphi$, $\langle\langle A\rangle\rangle\varphi\,\mathcal{U}\,\psi$ for a coalition A and ATEL-formulas φ and ψ;
- $K_a\varphi$ for $a \in \mathbf{\Sigma}$ and ATEL-formula φ;
- $E_A\varphi$, $C_A\varphi$, $D_A\varphi$ for a coalition A and ATEL-formula φ.

$\lhd$

Logical truth ($\top$) and the Boolean connectives ($\wedge$, $\rightarrow$, and $\leftrightarrow$) are defined as usual. The operator $\langle\langle A\rangle\rangle\Diamond\varphi$ is defined as $\langle\langle A\rangle\rangle\top\,\mathcal{U}\,\varphi$. Observe that, in ATEL, the operator $\langle\langle A\rangle\rangle\Box$ cannot be expressed in terms of $\langle\langle A\rangle\rangle$ and $\mathcal{U}$ since formulas of the form $\langle\langle A\rangle\rangle\neg(\top\,\mathcal{U}\,\neg\varphi)$, which are equivalent to $\langle\langle A\rangle\rangle\Box\varphi$, are not according to the syntax. ATL is the fragment of ATEL without epistemic operators.

Now consider the semantic structures that are epistemic extensions of alternating transition systems from [1].

Definition 2.2 [Alternating Epistemic Transition System] Given a finite set $\Sigma = \{1, \ldots, n\} \subset \mathbf{\Sigma}$ of agents with $n \geq 1$, an *alternating epistemic transition system (AETS) for* Σ is a tuple $\mathcal{S} = \langle \Pi, \Sigma, Q, \sim_1, \ldots, \sim_n, \pi, \delta\rangle$ where

- $\Pi \subseteq \mathbf{\Pi}$ is a finite, non-empty set of *atomic propositions*,
- Q is a finite, non-empty set of *states*,
- $\sim_a \subseteq Q \times Q$ is an *epistemic accessibility relation*, one for each agent $a \in \Sigma$, which is required to be an equivalence relation,
- $\pi : Q \rightarrow \mathbf{2}^{\Pi}$ is a *valuation function* which assigns to every state a set of atomic propositions that are true there, and
- $\delta : Q \times \Sigma \rightarrow \mathbf{2}^{\mathbf{2}^{Q}}$ is a *transition function* which maps a state $q \in Q$ and an agent $a \in \Sigma$ to a non-empty set of *choices* $\delta(q, a)$ available to a at q such that the following condition is satisfied: for every state $q \in Q$ and every set $Q_1, \ldots, Q_n$ of choices $Q_i \in \delta(q, i)$, $1 \leq i \leq n$, the intersection $Q_1 \cap \cdots \cap Q_n$ is a singleton set.

$\lhd$

For each coalition A, the following abbreviations for epistemic relations corresponding to epistemic operators E_A, C_A, and D_A are used:

$\sim_A^E = \bigcup_{a\in A} \sim_a$ .. the union of the accessibility relations of the agents in A;
$\sim_A^C = (\sim_A^E)^+$ .. the transitive closure of the relation $\sim_A^E$;
$\sim_A^D = \bigcap_{a\in A} \sim_a$ .. the intersection of the accessibility relations of the agents in A.

Below the relations $\sim_A^E$, $\sim_A^C$, and $\sim_A^D$ will be used to give a Kripke-style semantics to the notions "every agent in A knows", "the agents in A have common knowledge", and "the agents in A have distributed knowledge".

Intuitively, $\delta(q, a)$ describes the *a-choices* available in q: when in state q, agent a chooses a set from $\delta(q, a)$ to ensure that the "next state" will be among those in the chosen set. This notion of a-choices is generalized to *A-choices*

for coalitions A of agents as follows: Given an AETS $\mathcal{S} = \langle \Pi, \Sigma, Q, \sim_1, \ldots, \sim_n, \pi, \delta \rangle$, for each state $q \in Q$ and each coalition $A \subseteq \Sigma$, set

$$\delta(q, A) := \begin{cases} \{Q_A \subseteq Q \mid Q_A = \bigcap_{a \in A} Q_a \text{ where } Q_a \in \delta(q, a)\} & \text{if } A \neq \emptyset \\ \{\bigcup \delta(q, \Sigma)\} & \text{if } A = \emptyset \end{cases}.$$

When in state q, the agents in coalition A collectively choose a set from $\delta(q, A)$ to ensure that the "next state" is from this set. Note that $\delta(q, A)$ is non-empty for each state q and coalition A, and $\delta(q, \Sigma)$ is a set of singletons. The states in the singleton sets of $\delta(q, \Sigma)$ are the *successors* of q, i.e., the system is completely determined when all the agents have made their choice. Since the empty coalition cannot influence the behaviour of the system, $\delta(q, \emptyset)$ is set to $\bigcup \delta(q, \Sigma)$, the set of all possible successors of q.

An infinite sequence $\lambda = q_0 q_1 q_2 \cdots \in Q^\omega$ of states is a *computation* if, for all positions $i \geq 0$, there is a choice $\{q_{i+1}\} \in \delta(q_i, \Sigma)$. Denote with $\lambda[i]$ the i-th component q_i in λ and with $\lambda[0, i]$ the initial sequence $q_0 \cdots q_i$ of λ.

A *strategy* for an agent $a \in \Sigma$ is a mapping $f_a : Q^+ \to \mathbf{2}^Q$ such that for all $\lambda \in Q^*$ and all $q \in Q$, $f_a(\lambda \cdot q) \in \delta(q, a)$ where $\lambda \cdot q$ denotes the concatenation of the finite sequence λ with the state q. Note that a strategy f_a maps each finite sequence $\lambda \cdot q$ of states to an choice in $\delta(q, a)$ available to agent a at the state q. A *strategy* for a coalition A is a set of strategies $F_A = \{f_a \mid a \in A\}$, one for each agent in A.

The set $\mathsf{out}(q, F_A)$ of *outcomes* of a strategy F_A starting at a state q is the set of all computations $\lambda = q_0 q_1 q_2 \cdots \in Q^\omega$ such that $q_0 = q$ and $q_{i+1} \in \bigcap_{f_a \in F_A} f_a(\lambda[0, i])$ for all $i \geq 0$.

Formulas of ATEL are interpreted over alternating epistemic transition systems, cf. [14].

Definition 2.3 [ATEL Semantics] Given an AETS $\mathcal{S} = \langle \Pi, \Sigma, Q, \sim_1, \ldots, \sim_n, \pi, \delta \rangle$, the satisfaction relation $\models$ is inductively defined as follows: For all states q of $\mathcal{S}$, coalitions of agents $A \subseteq \Sigma$, agents $a \in \Sigma$, and ATEL-formulas φ, φ_1, and φ_2, it holds that

- $\mathcal{S}, q \models p$ iff $p \in \pi(q)$, for all atomic propositions $p \in \Pi$;
- $\mathcal{S}, q \models \neg\varphi$ iff $\mathcal{S}, q \not\models \varphi$;
- $\mathcal{S}, q \models \varphi_1 \vee \varphi_2$ iff $\mathcal{S}, q \models \varphi_1$ or $\mathcal{S}, q \models \varphi_2$;
- $\mathcal{S}, q \models \langle\langle A \rangle\rangle \bigcirc \varphi$ iff there is a strategy F_A such that for all computations $\lambda \in \mathsf{out}(q, F_A)$, it holds that $\mathcal{S}, \lambda[1] \models \varphi$;
- $\mathcal{S}, q \models \langle\langle A \rangle\rangle \Box \varphi$ iff there is a strategy F_A such that for all computations $\lambda \in \mathsf{out}(q, F_A)$, it holds that $\mathcal{S}, \lambda[i] \models \varphi$ for all positions $i \geq 0$;
- $\mathcal{S}, q \models \langle\langle A \rangle\rangle \varphi_1 \,\mathcal{U}\, \varphi_2$ iff there is an strategy F_A such that for all computations $\lambda \in \mathsf{out}(q, F_A)$, there is a position $i \geq 0$ such that $\mathcal{S}, \lambda[i] \models \varphi_2$ and $\mathcal{S}, \lambda[j] \models \varphi_1$ for all positions j with $0 \leq j < i$;

- $\mathcal{S}, q \models K_a\varphi$ iff $\mathcal{S}, q' \models \varphi$ for all q' with $q \sim_a q'$;
- $\mathcal{S}, q \models E_A\varphi$ iff $\mathcal{S}, q' \models \varphi$ for all q' with $q \sim_A^E q'$;
- $\mathcal{S}, q \models C_A\varphi$ iff $\mathcal{S}, q' \models \varphi$ for all q' with $q \sim_A^C q'$;
- $\mathcal{S}, q \models D_A\varphi$ iff $\mathcal{S}, q' \models \varphi$ for all q' with $q \sim_A^D q'$.

If for some state q of some AETS $\mathcal{S}$ it holds that $\mathcal{S}, q \models \varphi$, then the ATEL-formula φ is *true* at q, and $\mathcal{S}$ is called a *model* of φ. An ATEL-formula is *satisfiable* if it has a model. $\lhd$

3 The Main Result

In this section, ATEL satisfiability is shown to be ExpTime-complete. This is done by extending the ExpTime-completeness result of ATL satisfiability [13] to ATEL which shows that adding epistemic operators for common knowledge and distributed knowledge to ATL does not yield an increase in computational complexity.

The following theorem states the main result of this paper. Note that we consider the satisfiability problem for ATEL as it was presented in the introduction above.

Theorem 3.1 *The satisfiability problem for ATEL is* ExpTime*-complete.*

The lower complexity bound carries over from the ExpTime-hard fragment ATL; see [11, 13]. For the upper bound, it is to show that ATEL satisfiability is in ExpTime. The rest of the paper is devoted to presenting a decision procedure for ATEL that runs in exponential time and thus shows containment of ATEL satisfiability in ExpTime. Correctness of the decision procedure follows from Lemma 3.8 while Lemma 3.9 proves its exponential running time. Due to space limitations, the presentation is rather compact and the proofs are sketched only. More details and full proofs, however, can be found in the accompanying technical report [12] and the paper [13] on which this work is based on.

The decision procedure implements an extended version of the type elimination construction for ATL from [13] that additionally accounts for ATEL's epistemic operators K_a, E_A, C_A, and D_A for each agent a and coalition A of agents where the epistemic operators are dealt with similarly to [8]. Start with the definition of an extended closure that contains all formulas that are relevant for deciding the input formula. This and the following notion of types as specific subsets of the extended closure are adapted from [13] to additionally account for epistemic operators.

Definition 3.2 [Extended Closure] Let φ be an ATEL-formula and n the number of agents occurring in φ. The *extended closure* $\mathsf{ecl}(\varphi)$ *of* φ is the smallest set which is closed under the following conditions:

- $\varphi \in \mathsf{ecl}(\varphi)$;
- $\mathsf{ecl}(\varphi)$ is closed under subformulas;
- $\mathsf{ecl}(\varphi)$ is closed under single negation;
- if $\langle\langle A\rangle\rangle\Box\psi \in \mathsf{ecl}(\varphi)$, then $\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\psi \in \mathsf{ecl}(\varphi)$;
- if $\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta \in \mathsf{ecl}(\varphi)$, then $\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta \in \mathsf{ecl}(\varphi)$;
- if $E_A\psi \in \mathsf{ecl}(\varphi)$ with $A = \{a_1, \ldots, a_\ell\}$, then $\neg(\neg K_{a_1}\psi \vee \cdots \vee \neg K_{a_\ell}\psi) \in \mathsf{ecl}(\varphi)$;
- if $C_A\psi \in \mathsf{ecl}(\varphi)$, then $E_AC_A\psi \in \mathsf{ecl}(\varphi)$;
- if $K_a\psi \in \mathsf{ecl}(\varphi)$ and $D_A\vartheta \in \mathsf{ecl}(\varphi)$ with $a \in A$, then $D_A\psi \in \mathsf{ecl}(\varphi)$;
- if $D_A\psi \in \mathsf{ecl}(\varphi)$ and $D_B\vartheta \in \mathsf{ecl}(\varphi)$ with $A \subseteq B$, then $D_B\psi \in \mathsf{ecl}(\varphi)$.

$\lhd$

Note that the cardinality of $\mathsf{ecl}(\varphi)$ is linear in the length of φ.

Definition 3.3 [Type] Let φ be an ATEL-formula. The set $\Psi \subseteq \mathsf{ecl}(\varphi)$ is a *type for* φ if the following conditions are satisfied:

(T1) $\psi_1 \vee \psi_2 \in \Psi$ iff $\psi_1 \in \Psi$ or $\psi_2 \in \Psi$, for all $\psi_1 \vee \psi_2 \in \mathsf{ecl}(\varphi)$;

(T2) $\psi \in \Psi$ iff $\neg\psi \notin \Psi$, for all $\neg\psi \in \mathsf{ecl}(\varphi)$;

(T3) $\langle\langle A\rangle\rangle\Box\psi \in \Psi$ iff $\{\psi, \langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\psi\} \subseteq \Psi$, for all $\langle\langle A\rangle\rangle\Box\psi \in \mathsf{ecl}(\varphi)$;

(T4) $\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta \in \Psi$ iff $\vartheta \in \Psi$ or $\{\psi, \langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta\} \subseteq \Psi$, for all $\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta \in \mathsf{ecl}(\varphi)$;

(T5) if $K_a\psi \in \Psi$, then $\psi \in \Psi$;

(T6) $E_A\psi \in \Psi$ iff $\neg(\neg K_{a_1}\psi \vee \cdots \vee \neg K_{a_\ell}\psi) \in \Psi$, for all $E_A\psi \in \mathsf{ecl}(\varphi)$ with $A = \{a_1, \ldots, a_\ell\}$;

(T7) $C_A\psi \in \Psi$ iff $\{\psi, E_AC_A\psi\} \subseteq \Psi$, for all $C_A\psi \in \mathsf{ecl}(\varphi)$;

(T8) if $D_A\psi \in \Psi$, then $\psi \in \Psi$;

(T9) if $K_a\psi \in \Psi$, then $D_A\psi \in \Psi$, for all $D_A\psi \in \mathsf{ecl}(\varphi)$ with $a \in A$;

(T10) if $D_A\psi \in \Psi$, then $D_B\psi \in \Psi$, for all $D_B\psi \in \mathsf{ecl}(\varphi)$ with $A \subseteq B$.

The set of all types for φ is designated by Γ_φ. $\lhd$

Note that the cardinality of Γ_φ is exponential in the length of φ.

Intuitively, a type describes a state q of a transition system by means of formulas from $\mathsf{ecl}(\varphi)$ which are true at q. The decision procedure will use all types for an input formula to determine its decidability. The conditions (T1) to (T10) state restrictions on the formulas in a type: (T1) to (T4) deal with Boolean and temporal formulas; the conditions (T5) to (T10) were added to handle epistemic formulas. In particular, the relation between formulas with epistemic operators K and D are taken care of by (T9) and (T10).

Before proceeding with the construction, introduce some auxiliary notions. Denote with Σ_φ the set of all agents that occur in the formula φ. A *next formula* is a formula of the form $\langle\langle A\rangle\rangle\bigcirc\psi$ or $\neg\langle\langle A\rangle\rangle\bigcirc\psi$. Assume that all next formulas in the extended closure $\mathsf{ecl}(\varphi)$ are linearly ordered such that no negative next formula occurs before a positive one. Denote with $\sharp_\psi$ the number of the next formula $\psi \in \mathsf{ecl}(\varphi)$ in this ordering; the numbering starts with 0. Since there are as many positive next formulas in $\mathsf{ecl}(\varphi)$ as negative ones, obtain the following enumeration $\psi_1, \ldots \psi_k$ with $\psi_1, \ldots, \psi_{k/2-1}$ positive next formulas and $\psi_{k/2}, \ldots, \psi_k$ negative next formulas.

Intuitively, the presented type elimination algorithm eliminates those types that are not relevant for building an AETS $\mathcal{S}$ that is a model for an input formula φ. The states of such an $\mathcal{S}$ are thought of consisting of sequences of *n-tuples* where $n = |\Sigma_\varphi|$. Each component of such an n-tuple corresponds to an agent and takes values from $\{0, \ldots, k-1\}$ which in turn refer to one of the k next formulas in $\mathsf{ecl}(\varphi)$. The set of all such n-tuples is denoted with $[k/n]$, and the states of $\mathcal{S}$ will be from $[k/n]^*$. Supposing $q \in [k/n]^*$ is a state of $\mathcal{S}$, then the set $\{q \cdot \vec{t} \mid \vec{t} \in [k/n]\}$ contains its potential successor states, thus the choices in $\delta(q, a)$ are subsets of this set. Every state q of $\mathcal{S}$ corresponds to some type that was not eliminated and the construction of $\mathcal{S}$ is aiming at satisfying every formula of this type at q. In particular, q will have to satisfy a number of next formulas, i.e., the definition of δ that specifies the choices available at q is crucial. The transition function δ is defined as follows: For each agent a (ranging from 0 to $n-1$), set

$$\delta(q, a) := \{\{q \cdot \vec{t} \in [k/n]^* \mid \vec{t} = (t_0, \ldots, t_{n-1}) \text{ and } t_a = p\} \mid p < k/2\}.$$

The idea here is that agent a makes an a-choice by "voting" for a positive next formula that he wants to be satisfied. Such votes are done via a's position in tuples $\vec{t} = (t_0, \ldots, t_{n-1})$. An A-choice is made by a collective vote of the entire coalition A. This is characterized as follows: a set $S \subseteq [k/n]$ is called an *A-voting set* if

$$S = \{\vec{t} = (t_0, \ldots, t_{n-1}) \mid t_a = \tau(a) \text{ for all } a \in A\}$$

for some mapping $\tau : A \to \{0, \ldots, k/2-1\}$. In order for q to satisfy a positive next formula $\langle\langle A\rangle\rangle\bigcirc\psi$, the construction of $\mathcal{S}$ ensures that ψ is satisfied at all states of the A-choice that corresponds to the voting set where coalition A votes solely for this formula.

Satisfying negative next formulas $\neg\langle\langle\Sigma_\varphi\rangle\rangle\bigcirc\psi$ is simple: they merely state that *all* successors of q satisfy $\neg\psi$. For satisfying $\neg\langle\langle A\rangle\rangle\bigcirc\psi$ with $A \subset \Sigma_\varphi$ at q, ψ needs to be refuted at a state in every A-choice. Note that, by picking a state from every A-choice, a state from every A'-choice for all $A' \subseteq A$ is picked as well. But this is fine as $\neg\langle\langle A\rangle\rangle\bigcirc\psi$ implies $\neg\langle\langle A'\rangle\rangle\bigcirc\psi$. However, for any $B \not\subseteq A$, $\neg\langle\langle A\rangle\rangle\bigcirc\psi$ does not imply $\neg\langle\langle B\rangle\rangle\bigcirc\psi$. Thus, in order not to automatically satisfy $\neg\langle\langle B\rangle\rangle\bigcirc\psi$ when picking elements in A-choices, some care is needed such that the picked elements from A-choices are not also elements of B-choices for any such B. Picking the right states that refute ψ is implemented by a "refutation function"; see [13] for details.

The construction of a model is more involved due to the presents of negated box formulas, until formulas and formulas of the form $\neg C_A\psi$. This is because the satisfaction of such formulas needs to be witnessed by some state in a model and such witnesses can be "far away". Negated box formulas and until formulas can be dealt with analogously to [4] for CTL with the help of so-called witness trees. Since, in ATEL, epistemic and temporal operators do not interact, formulas of the form $\neg C_A\psi$ can be treated separately with a notion of witness paths. For defining witness trees, the following auxiliary notions from [13] are needed.

Definition 3.4 [ϑ-vector, φ-tree, $\bigcirc$-matching] Let φ be an ATEL-formula, $n = |\Sigma_\varphi|$, and k the number of next-formulas in $\mathsf{ecl}(\varphi)$. For each next-formula $\vartheta \in \mathsf{ecl}(\varphi)$ and vector $\vec{t} = (t_0, \ldots, t_{n-1}) \in [k/n]$,

- if $\vartheta = \langle\langle A\rangle\rangle\bigcirc\psi$ and $t_a = \sharp_\vartheta$ for each $a \in A$, then $\vec{t}$ is called a *ϑ-vector*;
- if $\vartheta = \neg\langle\langle A\rangle\rangle\bigcirc\psi$ with $A \subset \Sigma_\varphi$, then $\vec{t}$ is called a *ϑ-vector* if the refutation function says so; see [13] for details;
- if $\vartheta = \neg\langle\langle\Sigma_\varphi\rangle\rangle\bigcirc\psi \in \mathsf{ecl}(\varphi)$, then $\vec{t}$ is called a *ϑ-vector*.

For each type $\Psi \in \Gamma_\varphi$ and each $\vec{t} = (t_0, \ldots, t_{n-1}) \in [k/n]$, let $S_\Psi(\vec{t}) \subseteq \mathsf{ecl}(\varphi)$ be the smallest set such that

(M1) if $\langle\langle A\rangle\rangle\bigcirc\psi \in \Psi$ and $\vec{t}$ is a $\langle\langle A\rangle\rangle\bigcirc\psi$-vector, then $\psi \in S_\Psi(\vec{t})$, and

(M2) if $\neg\langle\langle A\rangle\rangle\bigcirc\psi \in \Psi$ and $\vec{t}$ is a $\neg\langle\langle A\rangle\rangle\bigcirc\psi$-vector, then $\neg\psi \in S_\Psi(\vec{t})$.

Given a set M of labels, a *$\langle M, k, n\rangle$-tree* T is a mapping T from a finite prefix-closed subset of $[k/n]^*$ to M. A *$\langle\Gamma_\varphi, k, n\rangle$-tree* is called a *$\varphi$-tree*. A φ-tree T is called *$\bigcirc$-matching* if, for all $\alpha \in \mathsf{dom}(T)$ and all $\vec{t} \in [k/n]$, $\alpha \cdot \vec{t} \in \mathsf{dom}(T)$ implies $S_{T(\alpha)}(\vec{t}) \subseteq T(\alpha \cdot \vec{t})$. ◁

Intuitively, a vector $\vec{t}$ is a $\langle\langle A\rangle\rangle\bigcirc\psi$-vector if, for all states q satisfying $\langle\langle A\rangle\rangle\bigcirc\psi$, the successor $q \cdot \vec{t}$ has to satisfy ψ. The $\neg\langle\langle A\rangle\rangle\bigcirc\psi$-vectors can be understood in an analogous way. This intuition is reflected in (M1) and (M2) and in the notion of $\bigcirc$-matching.

Now, equipped with the notions ϑ-vector, φ-tree, and $\bigcirc$-matching, define witness trees and witness paths. While witness paths testify satisfaction of formulas of the form $\neg C_A\psi$, witness trees are special φ-trees that affirm satisfaction of negated box formulas and until formulas; cf. [13] for witness trees.

Definition 3.5 [Witness Tree, Witness Path] Let φ be an ATEL-formula, Γ a set of types for φ, and $\Psi \in \Gamma$. A φ-tree T is called a *witness-tree rooted at* Ψ *in* Γ *for a formula* $\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta$ if it satisfies the following properties:

1. for all $\alpha \in \mathsf{dom}(T)$, $T(\alpha) \in \Gamma$;
2. T is $\bigcirc$-matching;
3. $T(\varepsilon) = \Psi$;
4. for all $\alpha \in \mathsf{dom}(T)$, $\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta \in T(\alpha)$;
5. for all non-leaf nodes α, $\psi \in T(\alpha)$;
6. for all leaf nodes α, $\vartheta \in T(\alpha)$;
7. if $\alpha \in \mathsf{dom}(T)$, $\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta \in T(\alpha)$, $\vartheta \notin T(\alpha)$, and $\vec{t}$ is a $\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\psi\,\mathcal{U}\,\vartheta$-vector, then $\alpha \cdot \vec{t} \in \mathsf{dom}(T)$.

T is called a *witness-tree rooted at* Ψ *in* Γ *for a formula* $\neg\langle\langle A\rangle\rangle\Box\psi$ if it satisfies the following properties:

1. for all $\alpha \in \mathsf{dom}(T)$, $T(\alpha) \in \Gamma$;
2. T is $\bigcirc$-matching;
3. $T(\varepsilon) = \Psi$;
4. for all $\alpha \in \mathsf{dom}(T)$, $\neg\langle\langle A\rangle\rangle\Box\psi \in T(\alpha)$;
5. for all leaf nodes α, $\neg\psi \in T(\alpha)$
6. if $\alpha \in \mathsf{dom}(T)$, $\neg\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\psi \in T(\alpha)$, $\neg\psi \notin T(\alpha)$, and $\vec{t}$ is a $\neg\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\psi$-vector, then $\alpha \cdot \vec{t} \in \mathsf{dom}(T)$.

A sequence $\Psi_0 \cdots \Psi_m \in \Gamma^*$, $m \geq 1$, is a *witness path rooted at* Ψ *in* Γ *for a formula* $\neg C_A\psi$ if the following conditions are satisfied:

1. $\Psi_0 = \Psi$;
2. for all $i < m$, there is an agent $a \in A$ such that for all $K_a\psi' \in \mathsf{ecl}(\varphi)$, $K_a\psi' \in \Psi_i$ iff $K_a\psi' \in \Psi_{i+1}$;
3. $\neg\psi \in \Psi_m$.

$\lhd$

The presented decision procedure employs a type elimination algorithm which in turn relies essentially on the notion of *realizability*. This notion from [13] was extended to additionally account for epistemic operators. Intuitively, a type Ψ is realizable in a set of types Γ if it is possible to (a) satisfy all next-formulas in Ψ and all formulas of the form $\neg K_a\psi$ or $\neg D_A\psi$ in Ψ using only types from Γ, (b) construct witness trees for all negated box formulas and until formulas in Ψ, and (c) construct witness paths for all formulas of the form $\neg C_A\psi$ in Ψ using only types from Γ.

Definition 3.6 [Realizability] Let φ be an ATEL-formula and Γ a set of types for φ. A type $\Psi \in \Gamma$ is *realizable in* Γ if the following conditions are satisfied:

1. for all $\vec{t} \in [k/n]$, there is a $\Psi' \in \Gamma$ such that $S_\Psi(\vec{t}) \subseteq \Psi'$;
2. for all $\langle\langle A\rangle\rangle\psi \,\mathcal{U}\, \vartheta \in \Psi$, there is a $\langle\langle A\rangle\rangle\psi \,\mathcal{U}\, \vartheta$-witness tree rooted at Ψ in Γ;
3. for all $\neg\langle\langle A\rangle\rangle\Box\psi \in \Psi$, there is a $\neg\langle\langle A\rangle\rangle\Box\psi$-witness tree rooted at Ψ in Γ;
4. for all $\neg K_a\psi \in \Psi$, there is a $\Psi' \in \Gamma$ such that
   (a) $\neg\psi \in \Psi'$, and
   (b) $K_a\psi' \in \Psi$ iff $K_a\psi' \in \Psi'$, for all $K_a\psi' \in \mathsf{ecl}(\varphi)$;
5. for all $\neg C_A\psi \in \Psi$, there is $\neg C_A\psi$-witness path rooted at Ψ in Γ;
6. for all $\neg D_A\psi \in \Psi$, there is a $\Psi' \in \Gamma$ such that
   (a) $\neg\psi \in \Psi'$, and
   (b) $D_A\psi' \in \Psi$ iff $D_A\psi' \in \Psi'$, for all $D_A\psi' \in \mathsf{ecl}(\varphi)$.

$\lhd$

Finally the decision procedure for ATEL using type elimination is described. The algorithm starts with the set of all types for an input formula and then repeatedly eliminates types that are not realizable. Since there are only finitely many types to start with, the algorithm terminates with a set of realizable types. The input formula is satisfiable if it is contained in some of the remaining types. Formally, this works as follows: Let φ be an ATEL-formula for which satisfiability is to be decided. Inductively compute a sequence $\Delta_0, \Delta_1, \ldots$ of sets of types for φ as follows:

$$\begin{array}{lcl} \Delta_0 & := & \Gamma_\varphi, \\ \Delta_{i+1} & := & \{\Psi \in \Delta_i \mid \Psi \text{ is realizable in } \Delta_i\}. \end{array}$$

Stop when $\Delta_{m+1} = \Delta_m$ for some $m \geq 0$. Note that Δ_m is a set of types for φ that are all realizable in Δ_m. The algorithm returns "Yes, φ is satisfiable in an AETS for Σ_φ" if φ is contained in some type in Δ_m; otherwise it returns "No, φ is not satisfiable."

The following lemmas state that the described decision procedure is correct and can be computed within exponential time in the length of the input formula; full proofs can be found in [12].

Lemma 3.7 *Let Γ be a set of types for an ATEL-formula φ. Then the existence of witness trees and paths in Γ can be decided in time exponential in the length of φ.*

Proof *(sketch).* Suppose it is to check for the existence of a witness tree for $\langle\langle A\rangle\rangle\psi\mathcal{U}\vartheta \in \Psi_0$ rooted at Ψ_0 in Γ; the cases for $\neg\langle\langle A\rangle\rangle\Box\psi$ and $\neg C_A\psi$ are similar. Start with identifying leaf nodes of possible witness trees by marking all types in Γ which contain $\langle\langle A\rangle\rangle\psi\mathcal{U}\vartheta$ and ϑ. Then identify inner nodes of possible witness trees as follows: For all unmarked types $\Psi \in \Gamma$ such that $\langle\langle A\rangle\rangle\psi\mathcal{U}\vartheta \in \Psi$, mark Ψ if $\psi \in \Psi$ and for every $\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\psi\mathcal{U}\vartheta$-vector $\vec{t} \in [k/n]$, there is a type $\Psi' \in \Gamma$ such that:

(i) $S_\Psi(\vec{t}) \subseteq \Psi'$, and

(ii) Ψ' is marked.

Repeatedly apply this procedure until no more types in Γ get marked.

It is not hard to see that a witness tree for $\langle\langle A\rangle\rangle\psi\mathcal{U}\vartheta$ exists iff Ψ_0 was marked. Verify that this marking procedure and thus checking for existence of witness trees runs in exponential time. QED

Lemma 3.8 *Let φ be an ATEL-formula. Then the procedure returns "Yes, the input formula φ is satisfiable in an AETS for Σ_φ" iff it is indeed the case.*

Proof *(sketch).* Soundness: Assume that the elimination procedure was started on input φ and returns "Yes, the input formula φ is satisfiable". Let Γ be the computed set of types. Then all types of Γ are realizable in Γ and there is a type $\Psi \in \Gamma$ with $\varphi \in \Psi$. Construct a model $\mathcal{S}$ of φ where states correspond to types in Γ. This is done by constructing witness trees for all until formulas and negated box formulas in $\mathsf{ecl}(\varphi)$ and for each type in Γ, arranging them in a matrix form such that the rows range over the eventualities and the columns over the types. Then we replace all leaf nodes by an edge from the leaf node's predecessor to the root of some other witness tree in the next row such that this root is labeled by the same type as the replaced leaf. This construction is similar to the one for CTL; cf. [4]. Initially, the epistemic operator D_A is treated as an additional modality and the corresponding epistemic accessibility relation $\sim_A^D$ in $\mathcal{S}$ is explicitly defined, i.e., $\sim_A^D = \bigcap_{a\in A} \sim_a$ for all coalitions A does not necessarily hold. However, a proper model $\mathcal{S}'$ can be constructed with the help of $\mathcal{S}$ by introducing epistemic witness trees on each state in $\mathcal{S}$ and rearranging, for each agent a, its epistemic accessibility relation $\sim_a$ such that temporal and epistemic transitions are kept separate.

Completeness: Suppose φ is satisfiable in a model $\mathcal{S}$. Let Q be the set of states in $\mathcal{S}$ and denote with $\mathsf{types}(Q)$ the set of all types associated with

some state in Q. It can be shown that all types in $\mathsf{types}(Q)$ are realizable in $\mathsf{types}(Q)$. QED

Lemma 3.9 *The described type elimination procedure runs in exponential time.*

Proof *(sketch).* Suppose φ is given as input and let $n = |\varphi|$. Recall that the size of the extended closure $\mathsf{ecl}(\varphi)$ is linear in the length of φ, i.e., $|\mathsf{ecl}(\varphi)| = c{\cdot}n$ for some constant $c \geq 1$. The algorithm computes a sequence $\Delta_0, \ldots, \Delta_m$ of sets of types such that $\Delta_0 \supsetneq \Delta_1 \supsetneq \cdots \supsetneq \Delta_m$. Since $\Delta_0 = \Gamma_\varphi \subseteq 2^{\mathsf{ecl}(\varphi)}$, this sequence is finite with $m \leq 2^{c \cdot n}$. For each i with $0 < i \leq m$, it holds that $|\Delta_i| < |\Delta_0| \leq 2^{c \cdot n}$. Thus, to compute the set Δ_{i+1} at most $2^{c \cdot n}$ types in Δ_i need to be checked whether they are realizable in Δ_i. Using Lemma 3.7, it can be shown that for a type $\Psi \in \Delta_i$ at most $2^{\mathcal{O}(n^2)}$ steps are needed to check for Ψ's realizability in Δ_i. Note that checking whether there is a type in Δ_m that contains φ takes not more than $2^{\mathcal{O}(n)}$ steps. Hence the decision procedure runs in time exponential in the size of the input. QED

4 Conclusion

We have investigated the complexity of the satisfiability problem for the epistemic extension ATEL of ATL with common and distributed knowledge which up to now was an open problem. ATEL satisfiability was shown to be ExpTime-complete by extending the type elimination construction for ATL from [13]. This result shows that adding epistemic operators for common and distributed knowledge to ATL does not yield an increase in computational complexity.

ATEL, as it is defined in this paper, does not allow for any interaction between knowledge and time. For instance, it is possible for an agent a that at two for a epistemically indistinguishable states a makes different choices. More discussions on the interaction of temporal and epistemic notions in ATEL can be found in, e.g., [14, 9]. For further research, it is interesting to investigate the complexity of variants of ATEL that capture various desired and reasonable interactions between knowledge and time.

References

[1] Alur, R., T. A. Henzinger and O. Kupferman, *Alternating-time temporal logic*, J. ACM **49** (2002), pp. 672–713.

[2] Baader, F., S. Ghilardi and C. Tinelli, *A new combination procedure for the word problem that generalizes fusion decidability results in modal logics*, in: D. Basin and M. Rusinowitch, editors, *Proceedings of the 2nd International Joint Conference on Automated Reasoning (IJCAR'04)*, Lecture Notes in Artificial Intelligence (2004), to appear.

[3] Baader, F., C. Lutz, H. Sturm and F. Wolter, *Fusions of description logics and abstract description systems*, Journal of Artificial Intelligence Research (JAIR) **16** (2002), pp. 1–58.

[4] Emerson, E. A., *Temporal and modal logic*, in: *Handbook of theoretical computer science (vol. B): formal models and semantics* (1990), pp. 995–1072.

[5] Gabbay, D., A. Kurucz, F. Wolter and M. Zakharyaschev, "Many-Dimensional Modal Logics: Theory and Applications," Studies in Logic **148**, Elsevier, 2003.

[6] Goranko, V., W. Jamroga and G. van Drimmelen, *Axiomatic systems for alternating-time temporal epistemic logics (extended abstract)*, in: *6th Conference on Logic and the Foundations of Game and Decision Theory (LOFT 2004), July 16 - 18, 2004, Leipzig Graduate School of Management, Germany, Proceedings.*

[7] Goranko, V. and G. van Drimmelen, *Decidability and complete axiomatization of the alternating-time temporal logic* (2005), theoretical Computer Science, to appear.

[8] Halpern, J. Y. and Y. Moses, *A guide to completeness and complexity for modal logics of knowledge and belief*, Artificial Intelligence **54** (1992), pp. 319–380.

[9] Jamroga, W. and W. van der Hoek, *Agents that know how to play*, Fundamenta Informaticae **62**, 2004, pp. 1–35.

[10] Pauly, M., "Logic for Social Software," Ph.D. thesis, University of Amsterdam (2001), ILLC Dissertation Series 2001-10.

[11] van Drimmelen, G., *Satisfiability in alternating-time temporal logic*, in: *18th IEEE Symposium on Logic in Computer Science (LICS 2003), 22-25 June 2003, Ottawa, Canada, Proceedings.*

[12] Walther, D., *Satisfiability of ATEL with common and distributed knowledge is ExpTime-complete*, Technical report, University of Liverpool (2005), see http://www.csc.liv.ac.uk/∼dirk/.

[13] Walther, D., C. Lutz, F. Wolter and M. Wooldridge, *Atl satisfiability is indeed exptime-complete*, submitted to Journal of Logic and Computation (2005).

[14] Wooldridge, M. and W. van der Hoek, *Time, knowledge, and cooperation: Alternating-time temporal epistemic logic and its applications*, in: *Proceedings of the 5th International Conference on Coordination Models and Languages* (2002), p. 4.

Automatic Proof Search in Logic of Justified Common Knowledge

Yegor Bryukhov

Graduate Center, City University of New York
365 Fifth Avenue, New York, NY 10016 *
ybryukhov@gc.cuny.edu

Abstract. We consider the logic of justified common knowledge $\mathsf{S4}^{\mathsf{J}}_n$ introduced in [Artemov, 2004]. This system captures the notion of *justified common knowledge*, which is free of some of the deficiencies of the usual common knowledge operator, and yet sufficient for analysis of epistemic problems where common knowledge has been traditionally applied. In particular, $\mathsf{S4}^{\mathsf{J}}_n$ enjoys cut-elimination, which opens a possibility for an automatic proof search in logic of common knowledge. In this paper, we present an implementation that automatically builds cut-free proofs in $\mathsf{S4}^{\mathsf{J}}_n$. Our work is based on the existing matrix-based prover for intuitionistic logic [Schmitt *et al.*, 2001].

1 Introduction

Plato defines knowledge as *justified true belief*. The modal logic approach to knowledge, developed in [Hintikka, 1961; Hintikka, 1962], captures the *true belief* components of Plato's tripartite definition. The *justification* component was introduced into formal epistemology in [Artemov and Nogina, 2005; Artemov, 2004].

1.1 Common Knowledge

Common knowledge is a standard notion in formal epistemology. If K_i $(i = 1, \ldots, n)$ are knowledge operators of individual agents and

$$E\varphi = K_1\varphi \wedge \ldots \wedge K_n\varphi,$$

we can informally define common knowledge as:

$$C\varphi \leftrightarrow \varphi \wedge E\varphi \wedge \ldots \wedge E^m\varphi \wedge \ldots \ .$$

The standard way of defining common knowledge axiomatically is via the *Fixed-Point Axiom*:

$$C\varphi \leftrightarrow E(\varphi \wedge C\varphi)$$

* The work was actually done while I was visiting Prof. Jason Hickey at CalTech

together with the *Induction Rule*:

$$\frac{\varphi \to E(\psi \wedge \varphi)}{\varphi \to C\psi}$$

This formalization does not behave well proof-theoretically; in particular, the above axiomatization does not admit cut-elimination [Alberucci and Jäger, 2005] – an automatic proof search becomes problematic.

An alternative evidence-based approach to common knowledge was introduced in [Artemov, 2004], using the new expressive power of the logics of knowledge with justification. The basic new atoms in the latter are explicit evidence operators of type $t : \varphi$, where φ is a sentence and t is an evidence term that uses a certain manageable system of operations on justification. New evidence operators $t : \varphi$ are completely described by the Logic of Proofs LP and provide an axiomatic and model description of epistemic assertions

$$t \textit{ is a justification for } \varphi \ .$$

We can also consider a forgetful projection of evidence that replaces each evidence assertion $t : \varphi$ by a sentence $J\varphi$, where J is a new modality of *justified knowledge*. [Artemov, 2004] considers, among others, the logic $\mathsf{S4}^{\mathsf{J}}_n$ with n $\mathsf{S4}$-modalities K_i denoting knowledge of n agents and an additional $n+1$-st $\mathsf{S4}$-modality J, together with the axiom scheme $J\varphi \to K_i\varphi$. It has been established in [Artemov, 2004] that $\mathsf{S4}^{\mathsf{J}}_n$ is indeed a forgetful projection of the corresponding logic of knowledge with justification. In particular, [Artemov, 2004] gives a Realization Algorithm that given an $\mathsf{S4}^{\mathsf{J}}_n$-derivation, it recovers explicit evidence terms for each occurrence of the justified knowledge modality J. These results support understanding of the justified knowledge statement $J\varphi$ as

$$\text{there is an access to a justification of } \varphi \ .$$

The logic $\mathsf{S4}^{\mathsf{J}}_n$ is a typical justified knowledge system which is also convenient to compare justified knowledge – represented by $\mathsf{S4}^{\mathsf{J}}_n$ – with the corresponding common knowledge system $\mathsf{S4}^{\mathsf{C}}_n$ based on the Fixed-Point Axiom together with the Induction Rule.

Here are the main properties of the justified knowledge system $\mathsf{S4}^{\mathsf{J}}_n$ established in [Artemov, 2004].

Fixed-Point Axiom The justified knowledge modality J satisfies the Fixed-Point Axiom of common knowledge in this form:

$$J\varphi \leftrightarrow E(\varphi \wedge J\varphi).$$

Connection with Common Knowledge Every evidence-based principle is a common knowledge principle but not vice versa:

$$(\mathsf{S4}^{\mathsf{J}}_n)^* \subset \mathsf{S4}^{\mathsf{C}}_n$$

where $*$ replaces every J-modality with C-modality.

Cut-Elimination $\mathsf{S4}^{\mathsf{J}}_n$ enjoys cut-elimination.

Hence Justified Knowledge is a well-behaved, constructive version of Common Knowledge, which can be used as the latter in solving specific problems (cf. [Artemov, 2004]).

In this paper we describe an implemented automatic proof search procedure for $\mathsf{S4}^{\mathsf{J}}_n$ which produces cut-free proofs. Given a cut-free proof in $\mathsf{S4}^{\mathsf{J}}_n$ one can use the Realization Algorithm to recover evidence terms in this proof.

We successfully formalized, and $\mathsf{S4}^{\mathsf{J}}_n$-prover proved the *Wise Men* and *Muddy Children* puzzles [Fagin *et al.*, 1995]. The Muddy Children Puzzle was formulated in several flavors for four children, and depending on its formulation the proof search and verification time varied from under one second to 90 seconds.

Our $\mathsf{S4}^{\mathsf{J}}_n$-prover is based on existing implementation for intuitionistic logics. The cases of T and $\mathsf{S5}$ as the base knowledge systems instead of $\mathsf{S4}$ are left for future work. The Realization Algorithm is currently implemented for $\mathsf{S4}$, but we leave it outside the scope of this paper, since we still need to extend it to the full $\mathsf{S4}^{\mathsf{J}}_n$ case and connect it to our $\mathsf{S4}^{\mathsf{J}}_n$-prover.

1.2 Existing Code for First-Order Intuitionistic Logic

We based our work on the existing code of the J-prover in the MetaPRL logical framework [Schmitt *et al.*, 2001]. The implementation is based on a uniform algorithm and matrix characterization for the first order intuitionistic logic J, modal logics K, K4, T, S4, S5, and fragments of linear logic were given [Otten and Kreitz, 1996b; Kreitz and Otten, 1999]. It also contains an algorithm for conversion of an existing matrix proof into a sequent proof [Schmitt and Kreitz, 1995; Kreitz and Schmitt, 2000; Schmitt, 1999].

Note that J here stands for first order intuitionistic logic and has nothing to do with "J" in $\mathsf{S4}^{\mathsf{J}}_n$ where it stands for "justified" S4-modality.

The code we started from supported only the first order intuitionistic logic J. Our work is about classical modal logic and does not involve intuitionistic logic. We generalize the matrix characterization of $\mathsf{S4}$ to $\mathsf{S4}^{\mathsf{J}}_n$, and extend that code to support matrix proof search and sequent proof reconstruction for $\mathsf{S4}^{\mathsf{J}}_n$.

We also improve the prefix unification algorithm [Otten and Kreitz, 1996a], which is an important part of the matrix proof search algorithm: on some problems it consumes as much as 99% of the total proof search and construction time. The unification is used iteratively with incremental extension of the equation system. Original algorithm was solving new systems from scratch each time. We modified it to reuse the information obtained from the previous iteration on the next iteration when one more equation was added. For problems with a unique most-general unifier, one has only to retain the unifier which itself represents the solved system. Unfortunately, prefix unification problems do not have unique solutions but rather minimal sets of unifiers. Naturally, not all members of a unifier set U for system S survive as members of a unifier set for system $S \cup p = q$.

The algorithm terminates when any unifier is found, but its full state contains the partially explored decision tree which we have to keep for the next iteration. Passing the unexplored part of decision tree from iteration to iteration reduced the running time by at least an order of magnitude on difficult problems.

The rest of the paper is organized in the following way. In Section 2, we introduce the matrix characterization of logical validity for propositional classical logic [Bibel, 1981; Bibel, 1987; Andrews, 1981]. In Section 3, we extend this notion to S4 [Wallen, 1990]. Section 4 contains our extension of the matrix method to $\mathsf{S4}_n^{\mathsf{J}}$, and Section 5 describes modifications to the prefix unification algorithm, which is a part of matrix method. Section 6 briefly describes the main ideas of matrix proof to sequent proof conversion [Kreitz and Schmitt, 2000; Schmitt, 1999] and our modifications to enable support of $\mathsf{S4}_n^{\mathsf{J}}$. And finally, Section 7 describes the Muddy Children Puzzle and our solution to it.

2 Matrix Characterization of Logical Validity

The connection method was introduced in [Bibel, 1981; Bibel, 1987] for classical logic. In [Wallen, 1990] it was extended to a number of non-classical logics. Kreitz *et al* suggested a uniform algorithm for a variety of modal logics, intuitionistic logic, and fragments of linear logic [Otten and Kreitz, 1996b; Kreitz and Otten, 1999]; described a uniform method of reconstruction of Gentzen sequent proofs from matrix proofs [Schmitt and Kreitz, 1995; Kreitz and Schmitt, 2000; Schmitt, 1999]; and implemented it for intuitionistic logic in the MetaPRL proof assistant [Schmitt *et al.*, 2001].

In this section, we will introduce the basic concepts of matrix method and describe logical validity in these terms. We will limit our attention to propositional case because our target logic $\mathsf{S4}_n^{\mathsf{J}}$ is propositional.

A *formula tree* of a formula F is basically its syntax tree. Each node s corresponds to one particular subformula F_s. We give unique names $a_1, a_2, \ldots$ to each node and call them *positions*. A *label* of a position s is the major (outermost) connective of F_s or is itself F_s if it is atomic. In the latter case, s is called *atomic position* or simply *atom*. The *tree ordering* $<$ on positions is induced by the tree structure where the root is the smallest element with respect to this tree ordering $<$.

Each position is associated with *polarity* in a standard way with the root having polarity 0.

The *principal type* $Ptype(u)$ of a position u is defined by its label and polarity according to the following table:

principal type α	$(A \wedge B)^1$	$(A \vee B)^0$	$(A \to B)^0$	$(\neg A)^1$	$(\neg A)^0$
subformulae polarities	A^1, B^1	A^0, B^0	A^1, B^0	A^0	A^1
principal type β	$(A \wedge B)^0$	$(A \vee B)^1$	$(A \to B)^1$		
subformulae polarities	A^0, B^0	A^1, B^1	A^0, B^1		

Atomic positions have no principal type.

Definition 1. *For a principal type t, we say that two positions a and b of F are* **t-related** *($a \sim_t b$) if the $<$-biggest position c of F that is less than a and b has principal type t ($c = \max\{x \mid x < a \ \wedge \ x < b\}$). a is t-related to a set S iff $a \sim_t s$ for all $s \in S$.*

Definition 2. *For each principal type t, there are* **secondary types** *$t_0, \ldots, t_k$ where k is the arity of connectives that produce type t. For position u of principal type t, its $<$-successors have secondary types t_i according to their relative order as arguments of u. We denote the secondary type of a position v as $Stype(v)$. The root has no secondary type.*

The *matrix(-representation)* of a formula F is a two-dimensional representation of its atomic subformulae without connectives (but with parentheses); it is mainly used for illustrative purposes. In this representation, α-related positions are arranged side by side; β-related positions are arranged one on top of another.

Example 1. The matrix representation of the formula $\neg((\neg A \vee \neg C) \wedge ((A \wedge B) \vee (B \wedge C)))$ is:

$$\begin{pmatrix} A^0 \\ C^0 \end{pmatrix} \quad \begin{pmatrix} (A^1 & B^1) \\ (B^1 & C^1) \end{pmatrix}$$

Definition 3. *A* **path** *through a formula F is a maximal set of α-related positions. It can be visualized as a maximal "horizontal" line through the matrix of F (or more precisely, a maximal line that never runs vertically).*

Definition 4. *A* **connection** *is a pair of positions with the same labels but opposite polarities.*

Theorem 1. *A propositional formula F is (classically) valid iff every path through F has a connection.*

A proof of this theorem (first-order case) can be found in [Bibel, 1987].

3 Matrix characterization for S4

The matrix characterization for S4 was given in [Wallen, 1990]. For this, we need to add two extra principal types ν, π for modal positions; define prefixes and unification over them; define multiplicity and what is an indexed formula; and define complementary connections and admissible substitutions.

We will use only one modality $\Box$ because $\Diamond$ has no explicit counterpart in the logic of knowledge with justification in which we want to realize $\mathsf{S4}^{\mathsf{J}}_n$-theorems.

Two new principal types are defined according to the table below:

principal type ν	$(\Box A)^1$
subformula polarity	A^1
principal type π	$(\Box A)^0$
subformula polarity	A^0

We denote the set of all positions of primary type ν as $\mathcal{V}$, of primary type π as Π, all positions of secondary type ν_0 as $\mathcal{V}_0$, and of secondary type π_0 as Π_0.

Definition 5. *For a position u, the* **modal prefix** $\mathrm{pre}_M(u)$ *is a string $u_1 \dots u_n$ of all positions $u_1 < u_2 < \ldots < u_n \leq u$ such that $u_i \in \mathcal{V}_0 \cup \Pi_0$.*

Definition 6. *The modal substitution $\sigma : \mathcal{V}_0 \to (\mathcal{V}_0 \cup \Pi_0)^*$ induces a relation $\sqsubset_M \subseteq (\mathcal{V}_0 \cup \Pi_0) \times \mathcal{V}_0$: if $\sigma_M(u) = p$, then $v \sqsubset u$ for all $v \in \mathcal{V}_0 \cup \Pi_0$ occurring in p.*

In other words, ν_0-positions are considered to be variables and π_0-positions are considered to be constants. All variables in the result of $\sigma(u)$ are less than u with respect to the ordering induced by σ.

Definition 7. *A modal substitution σ is* **S4-admissible** *iff the induced reduction ordering $\lhd = (< \cup \sqsubset)^+$ is irreflexive.*

In S4-tableaux, ν-formulae can be used several times to populate several prefixes (worlds); in order to reflect it in the matrix characterization, we introduce modal multiplicity $\mu : \mathcal{V}_0 \to \mathbb{N}$ and define *indexed formula* F^μ which is just a pair – a formula and a multiplicity function over it. A position $u \in \mathcal{V}$ has $\mu(u)$ instances of its successor trees with root v, $u < v$. Whenever we refer to index formula, we imply that positions are duplicated according to the multiplicity function.

Definition 8. *A connection $\{u, v\}$ is* **complementary** *under a substitution σ iff $\sigma(\mathrm{pre}(u)) = \sigma(\mathrm{pre}(v))$. A path is complementary iff it contains a complementary connection.*

Definition 9. *A set of connections C* **spans** *the formula A iff any path through A contains some connection from C.*

Theorem 2. *A formula is S4-valid iff there is a multiplicity μ, an S4-admissible substitution σ, and a set of σ-complementary connections C that spans A [Wallen, 1990].*

4 Matrix Characterization of $\mathsf{S4}^{\mathsf{J}}_{n}$

$\mathsf{S4}^{\mathsf{J}}_{n}$ can be described as n modalities K_i with S4 behavior, modality J with S4 behavior and the connection axiom schema

$$Jp \to K_i p, \text{ for all } i. \tag{4.I}$$

To extend matrix characterization from singular S4 to several such modalities, one has to label tree positions with the appropriate modalities and make such positions incompatible for prefix unification purposes.

In terms of tableau rules, the connection axiom 4.I can be expressed as

$$\frac{T\ \sigma \quad Jp}{T\ \sigma.a_i\ p} \text{ for an existing prefix } \sigma.a_i$$

This means that J-modality is compatible with worlds generated by any modality $K_1, \ldots, K_n, J$. For unification purposes, we allow variable positions related to J-modalities to unify with any string.

We will mark position a with a superscript i if it is related to modality K_i. For J we will use superscript 0.

Definition 10. *For a position a^i, we define* $\text{sort}\{a^i\} = i$

Definition 11. *To describe prefix unification for $\mathsf{S4}^{\mathsf{J}}_n$, we define a relation $\preceq$ on positions. For two positions a and b, we say that $a \preceq b$ iff a is ν_0^0 or a is ν_0^i, and b is ν_0^i or π_0^i. Basically, it states when b can be substituted for a.*

Corollary 1. *$\preceq$ is reflexive and transitive.*

Definition 12. *A substitution $\{V_1 \setminus s_1, \ldots, V_n \setminus s_n\}$ is* **$\mathsf{S4}^{\mathsf{J}}_n$-admissible** *iff it is $\mathsf{S4}$-admissible and for all i and all a from s_i, holds $V_i \preceq a$.*

Theorem 3. *A formula is $\mathsf{S4}^{\mathsf{J}}_n$-valid iff there is a multiplicity μ, an $\mathsf{S4}^{\mathsf{J}}_n$-admissible substitution σ, and a set of σ-complementary connections C that spans A.*

Proof. (Sketch) The only difference between $\mathsf{S4}$ and $\mathsf{S4}^{\mathsf{J}}_n$ tableaux are the rules of propagation of $\Box$ from prefix to prefix. Our adjustments in the definition of admissible substitution reflects those differences exactly.

We can modify our definition of a path through a formula A^μ as follows. We say that some sets of α-related positions are paths (and our original paths become atomic paths). Instead of $P \cup \{v\}$, we will write P, v.

(A) Root is a path.
(B) If P, u is a path, and u is an α-position with subformulae u_0 and u_1, then P, u_0 and P, u_1 are paths as well.
(C) If P, u is a path, and u is a β-position with subformulae u_0 and u_1, then P, u_0, u_1 is a path.
(D) If P, u is a path, and u is a π-position with subformula u_0, then P, u_0 is a path.
(E) If P, u is a path, and u is a ν-position with subformula u_0, then P, $u_{0,1}, \ldots$, $u_{0,\mu(u)}$ is a path where $u_{0,i}$ are instances of u_0.

If we augment each position in paths with its prefixes, paths will become $\mathsf{S4}^{\mathsf{J}}_n$-tableau branches. Atomic paths will be closed branches (if there is a spanning set of connections). Our path generation rules become valid tableau rules as long as prefixes are introduced by π-rules before they are used by ν-rules. But we can always apply the ν-rule after the appropriate π-rule – they cannot block each other because their active positions are from different branches of the syntax tree. Hence if there is a matrix proof then there is a tableaux proof as well.

Once we have a tableau proof, we can find the appropriate multiplicity and an $\mathsf{S4}_n^{\mathsf{J}}$-admissible substitution. Each tableau branch represents one or many atomic paths, each closed branch has a complementary connection, and all branches of a closed tableau cover all atomic paths.

Note Our implementation does not use tableaux in any way, we refer to them only for the sake of the proof(-sketch).

[Cerrito and Mayer, 1997] give an upper bound for multiplicity function for a number of propositional modal logics, including S4. We tried to implement it but it proved to be unpractically high.

5 Prefix Unification for $\mathsf{S4}_n^{\mathsf{J}}$

[Otten and Kreitz, 1996a] present a prefix unification algorithm for a number of logics, including S4 (the algorithm is the same but it uses different sets of rules for different logics).

We consider strings over $\mathcal{V} \cup \mathcal{C}$ where $\mathcal{V}$ serves as a set of variables and $\mathcal{C}$ is a set of constants. Symbols are actually positions, so for the $\mathsf{S4}_n^{\mathsf{J}}$ case, we assume that each symbol has a sort : $\mathcal{V} \cup \mathcal{C} \rightarrow \mathbb{N}$ which we denote as a superscript, and we have a sort compatibility relation $\prec$ over symbols.

Definition 13 (T-string property). *Two strings a and b over an alphabet $\mathcal{A}$ have* **T-string property** *iff*

(A) $\forall_{1 \leq i,j \leq |s|}.i \neq j \rightarrow s_i \neq s_j$ *and* $\forall_{1 \leq i,j \leq |t|}.i \neq j \rightarrow t_i \neq t_j$
(B) $\exists_{0 \leq k \leq \min(|s|,|t|)}.(\forall_{1 \leq i \leq k}.s_i = t_i \& \forall_{k < i \leq |s|} \forall_{k < j \leq |t|}.s_i \neq t_j)$

In other words, these strings form a tree.

Definition 14. *A set of strings S has* **T-string property** *iff all pairs of strings $s, t \in S$ have T-string property.*

Definition 15. *A system of equations $\{s_1 = t_1, \ldots, s_n = t_n\}$ has T-string property iff the set $\{s_1, \ldots, s_n, t_1, \ldots, t_n\}$ has T-string property.*

Definition 16. *A substitution σ is a* **T-unifier** *of a system of equations $\{s_i = t_i | 1 \leq i \leq n\}$ iff $\sigma s_i = \sigma t_i$ for all $1 \leq i \leq n$.*

Definition 17. *A substitution σ is an* **instance** *of τ if there is δ such that $\sigma = \delta \circ \tau$ (δ applied after τ).*

Definition 18. *A set of substitutions Σ is a* **(minimal) set of most general unifiers** *for a system of equations Γ iff:*

Correctness *Every $\sigma \in \Sigma$ is a T-unifier for Γ.*
Completeness *Every T-unifier τ for Γ is an instance of some $\sigma \in \sigma$.*
Minimality *No $\sigma \in \Sigma$ is an instance of another $\sigma' \in \Sigma$.*

A nondeterministic version of the unification algorithm is the following. It assumes that each equation in Γ has the form $s = r|t$ where $r = \varepsilon$ at the beginning:

while $\Gamma \neq \emptyset$ **do**
select the left most system $s = t \in \Gamma$
set Γ to $\Gamma \setminus \{s = t\}$
select a transformation rule $\Pi \to \Delta, \Theta$ from $\mathcal{T}$ which is applicable to $s = t$;
if no rule is applicable stop with failure
apply rule $\Pi \to \Delta, \Theta$ to $s = t$, let Δ', Θ' be the results
$\Gamma := \Theta'(\Gamma)$, $\sigma := \Theta(\sigma)$
$\Gamma := \Delta' \cup \Gamma$, $\sigma = \sigma \cup \Theta'$
stop with success and return σ**.**

The set of most general unifiers is a result of all successful runs of the algorithm.

Definition 19. *For two variable positions a^i and b^j, we say that $a^i \simeq b^j$ iff $i = j$ or one of them is 0.*

Corollary 2. *$\simeq$ is an equivalence relation.*

If $a^i \simeq b^j$, then the least constraining sort of variables that can be substituted for *both* a^i and b^j is $\max(i, j)$. Indeed, if $\max(i, j)$ is positive, then it is the only possible sort; if it is 0, then it can be replaced with any other sort, but if we choose any positive sort, we will then be bound to it.

The original list of rules for T-unification for S4 described in [Otten and Kreitz, 1996a] changes to the one described in Table 1. There we added the Rules 11 trough 14 and sort compatibility conditions to the Rules 7 trough 10.

Corollary 3. (T-String Preserving) *Let Γ be system of equations with T-string property and σ be a most general unifier for one of its equations, then a system of equations $\sigma\Gamma$ still has T-string property.*

Definition 20. *Let* $\mathrm{mgu}(\Gamma)$ *be the set of all substitutions produced by the given algorithm with the given set of rules.*

Corollary 4. (Correctness) *Let $\Gamma = \{s = \varepsilon|t\}$ be an equation of T-strings. Then all $\sigma \in \mathrm{mgu}(\Gamma)$ are T-unifiers for Γ.*

Proof is the same as in [Otten and Kreitz, 1996a].

Theorem 4. (Termination) *Let $\Gamma = \{s = \varepsilon|t\}$ be an equation of T-strings. Then the given algorithm with the given set of rules always terminates.*

Proof. [Otten and Kreitz, 1996a] prove it by defining an ordering on equations and showing that we need at most two rule applications to obtain a smaller equation with respect to that ordering. This proof works for our set of rules as well.

R1	$\{\varepsilon = \varepsilon \mid \varepsilon\}$	$\rightarrow \{\}, \{\}$
R2	$\{\varepsilon = \varepsilon \mid t^+\}$	$\rightarrow \{t^+ = \varepsilon \mid \varepsilon\}, \{\}$
R3	$\{Xs = \varepsilon \mid Xt\}$	$\rightarrow \{s = \varepsilon \mid t\}, \{\}$
R4	$\{Cs = \varepsilon \mid Vt\}$	$\rightarrow \{Vt = \varepsilon \mid Cs\}, \{\}$
R5	$\{Vs = z \mid \varepsilon\}$	$\rightarrow \{s = \varepsilon \mid \varepsilon\}, \{V \setminus z\}$
R6	$\{Vs = \varepsilon \mid C_1 t\}$	$\rightarrow \{s = \varepsilon \mid C_1 t\}, \{V \setminus \varepsilon\}$
R7	$\{Vs = z \mid C_1 C_2 t\}$	$\rightarrow \{s = \varepsilon \mid C_2 t\}, \{V \setminus zC_1\}$, where $V \preceq C_1$
R8	$\{Vs^+ = \varepsilon \mid V_1 t\}$	$\rightarrow \{V_1 t = V \mid s^+\}, \{\}$, where $V_1 \preceq V$
R9	$\{Vs^+ = z^+ \mid V_1 t\}$	$\rightarrow \{V_1 t = V' \mid s^+\}, \{V \setminus z^+ V'\}$, where $V \simeq V_1$, and $\mathrm{sort}\{V'\} = \max(\mathrm{sort}\{V\}, \mathrm{sort}\{V_1\})$
R10	$\{Vs = z \mid Xt\}$	$\rightarrow \{Vs = zX \mid t\}, \{\}$, where $V \preceq X$, and $V \neq X$, and $s = \varepsilon$ or $t \neq \varepsilon$ or $X \in \mathcal{C}$
R11	$\{Vs = z \mid V_1 t\}$	$\rightarrow \{s = \varepsilon \mid V_1 t\}, \{V \setminus z\}$, where ($z \neq \varepsilon$ and $V \not\preceq V_1$) or ($z = \varepsilon$ and not $V \simeq V_1$)
R12	$\{V = \varepsilon \mid V_1 t\}$	$\rightarrow \{V = \varepsilon \mid t\}, \{V_1 \setminus \varepsilon\}$, where $V_1 \preceq V$ but $V \not\preceq V_1$
R13	$\{V = \varepsilon \mid V_1 t\}$	$\rightarrow \{V_1 t = V' \mid \varepsilon\}, \{V \setminus V'\}$, where $V_1 \preceq V$ but $V \not\preceq V_1$, and $\mathrm{sort}\{V'\} = \mathrm{sort}\{V\}$
R14	$\{Vs = zV_1 \mid Ct\}$	$\rightarrow \{s = \varepsilon \mid Ct\}, \{V \setminus zV_1\}$, where $V \not\preceq C$

Table 1. s, t and z denote (arbitrary) strings, and s^+, t^+, z^+ non-empty strings. X, V, V_1, C, C_1, and C_2 denote single characters with $X \in \mathcal{V} \cup C \cup \mathcal{V}'$, $V, V_1 \in \mathcal{V} \cup \mathcal{V}'$ (with $V \neq V_1$), and $C, C_1, C_2 \in \mathcal{C}$. $V' \in \mathcal{V}'$ is a new variable which does not occur in the substitution σ computed so far.

[Otten and Kreitz, 1996a] claim minimality of the resulting unifier sets for their S4-set of rules. We do not claim it for our rules. See our practical considerations at the end of this section.

Theorem 5. (Completeness) *Let $\Gamma = \{s = \varepsilon \mid t\}$ be an equation of T-strings.* $\mathrm{mgu}(\Gamma)$ *is complete, i.e. any unifier of Γ is an instance of some $\sigma \in \mathrm{mgu}(\Gamma)$.*

Proof. [Otten and Kreitz, 1996a] claim completeness of the original S4-set of rules. Completeness of our extended set can be obtained by lengthy analysis of all cases when S4-rules are not applicable due to modal sort compatibility requirements.

5.1 Practical considerations

Informally, T-unification problems are not exactly isomorphic to desired solutions of prefix equations. For example, fresh variables introduced in the Rules 9 and 13 are set to ε after a unifier is found. Substitution $\{V_1 \setminus \varepsilon, V_2 \setminus \varepsilon\}$ is an instance of $\{V_1 \setminus V_2\}$; for proof search purposes they are equally good, but if we still have some equations to solve, the former substitution being applied to them produces a smaller problem. These considerations suggest that minimality and completeness of the resulting unifier sets do not guarantee optimality.

We must also pay close attention to the order in which rules are attempted – swapping Rules 12 and 13 changes running time on certain problems by more

than an order of magnitude, and changes (practical) feasibility of some problems. We plan to measure "popularity" of the rules on a large base of problems and reorder them by decreasing order of popularity.

6 From Matrix Proofs to Sequent Proofs

The algorithm for the conversion of matrix proofs to sequent proofs was given by Kreitz and Schmitt [Kreitz and Schmitt, 2000; Schmitt, 1999] for a number of modal logics (K,K4,T,S4,D,D4), first order intuitionistic logic, and fragments of linear logic.

The full description of the algorithm is lengthy and very technical, so we will highlight only the main idea and details that change with our move from S4 to $\mathsf{S4}^{\mathsf{J}}_n$.

The algorithm goes over the formula tree starting from its root and converts each position into an inference rule (or several inference rules); the result is a derivation tree. Suppose, the original formula that we had to prove was $A \wedge B$ (or $\Rightarrow A \wedge B$ as a sequent). The primary type of the root is α and the main connective is $\wedge$, the algorithm chooses this rule:

$$\frac{\Gamma \Rightarrow A, \Delta \qquad \Gamma \Rightarrow B, \Delta}{\Gamma \Rightarrow A \wedge B, \Delta}$$

Positions for A and B become *open* and the algorithm picks next position among open ones.

In general, inference rules are not permutable because certain rules can destroy formulae of a sequent that have not yet been used, hence the order in which the algorithm has to explore the formula tree is important. The primary factor is the reduction ordering induced by unifying substitution.

Definition 21. *We say that the processed positions are* **solved***, and that P_a is the set of solved atomic positions.*

Definition 22. *We call* **open** *a not-solved position whose immediate predecessor in the tree ordering is solved. $P_o = \{x \mid x \text{ not solved } \& \ y \prec x \text{ solved}\}$ is the set of open positions.*

Basically, open positions represent current formulas in the sequent.

Definition 23. *If $x \sqsubset y$ (substitution induced relation), we say that x is blocking y. And $W_y = \{x \mid x \sqsubset y\}$ is the set of positions blocking y.*

Open positions that are blocked will not be processed until there is no blocking. There are more requirements for first-order S4 in [Kreitz and Schmitt, 2000; Schmitt, 1999], unfortunately, they do not fully agree.

Note Because generated proofs are always checked by MetaPRL engine, it is not critical if the algorithm has bugs and/or incomplete. There are two worst case scenarios: algorithm runs into a deadlock because of these blocking conditions, or it might produce an incorrect proof, which will be rejected by MetaPRL.

Consider the $\mathsf{S4}$-rule for $\Box$-introduction to the right:

$$\frac{\Gamma^* \Rightarrow \Delta, A}{\Gamma \Rightarrow \Delta, \Box A} \qquad (\Rightarrow \Box)$$

Where $\Gamma^* = \{\Box X | \Box X \in \Gamma\}$, it is more convenient to formulate it in this form with * operation when we do a bottom-up proof.

If we apply this rule too early, the * operation might delete some formulae that we will need later in our proof. The conversion algorithm has a complicated condition describing when it is not too early to use this rule.

For $\mathsf{S4}^{\mathsf{J}}_n$, the similar rule is:

$$\frac{\Gamma^* \Rightarrow \Delta, A}{\Gamma \Rightarrow \Delta, \Box A} \qquad (\Rightarrow \Box)$$

If $\Box$ is J then $\Gamma^* = \{JA | JA \in \Gamma\}$.

If $\Box$ is K_i then $\Gamma^* = \{JA | JA \in \Gamma\} \cup \{K_i A | K_i A \in \Gamma\}$.

So in the case of $\mathsf{S4}^{\mathsf{J}}_n$, not all boxes survive after this rule but only boxes of the same sort or stronger (i.e. J). In order to reflect this in the proof conversion algorithm for $\mathsf{S4}^{\mathsf{J}}_n$, we impose one more condition when a π-position cannot be processed – *if there is an open ν-position of a different non zero sort*. Otherwise, if we process that π-position, we will apply the $\Rightarrow \Box$-rule to it and it will delete the incompatible ν-positions.

This modification is the only one needed to add $\mathsf{S4}^{\mathsf{J}}_n$ support to the existing proof conversion algorithm for $\mathsf{S4}$.

7 Muddy Children Puzzle

The paradigmatic example of $\mathsf{S4}^{\mathsf{J}}_n$ application is the renowned:

Muddy Children Puzzle There are n children playing in the yard; as they play m of them become muddy. Their father enters the yard and says: "at least one of you is muddy, do you know if you are muddy?" Nobody answers, so he asks again: "Do you know if you are muddy?" He asks this question until all the children are able to answer the question.
After how many questions the father will receive the answers?
We assume that all the children are smart and answer simultaneously.

Let's formalize this puzzle in $\mathsf{S4}^{\mathsf{J}}_n$ for four children, two of them muddy. Propositional letter c_i will stand for "i^{th} child is muddy". We will use the formalization, that we heard from Melvin Fitting.

Definition 24. *The ith child knows whether A is true:*

$$KW_i A = K_i A \vee K_i \neg A$$

Definition 25. *It is common knowledge that every child knows about other children if they are muddy:*

$$KAO = J(\ KW_1c_2 \wedge KW_1c_3 \wedge KW_1c_4 \wedge KW_2c_1 \wedge KW_2c_3 \wedge KW_2c_4 \wedge KW_3c_1 \wedge KW_3c_2 \wedge KW_3c_4 \wedge KW_4c_1 \wedge KW_4c_2 \wedge KW_4c_3)$$

Definition 26. *The initial state before the father said anything was*

$$s_0 = c_1 \wedge c_2 \wedge \neg c_3 \wedge \neg c_4 \wedge KAO$$

Definition 27. *The father's statement "At least one of you is muddy" is formalized as*

$$s_1 = J(c_1 \vee c_2 \vee c_3 \vee c_4)$$

Definition 28. *Some child knows whether (s)he is muddy or not:*

$$SK = KW_1c_1 \vee KW_2c_2 \vee KW_3c_3 \vee KW_4c_4$$

Theorem 6. *After the father asked his question first time, nobody answered. Immediately after that, muddy children (c_1 and c_2) realized that they were muddy:*

$$s_0, J(\neg J(s_1 \rightarrow SK)) \Rightarrow K_1c_1$$
and
$$s_0, J(\neg J(s_1 \rightarrow SK)) \Rightarrow K_2c_2$$

- $s_1 \rightarrow SK$ *says that after first question some child knew if (s)he is muddy.*
- $J(s_1 \rightarrow SK)$ *says that it became common knowledge, i.e. that child publicly said that (s)he knows.*
- *And finally,* $J(\neg J(s_1 \rightarrow SK))$ *says that it is common knowledge that no child actually answered after the first question.*

Proof. The proof was found automatically by our matrix-based prover and verified by MetaPRL, it took approximately 92 seconds in total (but separately for c_1 and c_2).

Unfortunately, Four Children Puzzle represents a real challenge to our implementation. It was not able to find a proof for the Theorem 6 above with $K_1c_1 \wedge K_2c_2$ in conclusion.

8 Final Remarks

We presented a matrix-based prover for $\mathsf{S4}^{\mathsf{J}}_n$. It builds cut-free proofs that can be fed to a Realization Algorithm, that would recover evidence terms (justifications) for justified common knowledge modality J, and build an $\mathsf{S4}^{\mathsf{LP}}_n$-proof for a realization of the original formula [Artemov, 2004]. It is our immediate goal to finish this Realization Algorithm and connect it to our $\mathsf{S4}^{\mathsf{J}}_n$-prover.

We are not satisfied with the performance of our implementation. One way to address it is to implement non-constant multiplicity function, i.e. to increase

multiplicity of generative positions individually, on demand. Although, it is not obvious how to do it. May be, we should move to tableaux-based algorithms, though we need sequent proofs as the output for the Realization Algorithm.

We also want to extend our work to other modal logics ($\mathsf{T}^{\mathsf{J}}_n$ and $\mathsf{S5}^{\mathsf{J}}_n$) [Artemov, 2004] and investigate possible connections between logic of proofs LP (and $\mathsf{T}^{\mathsf{LP}}_n$, $\mathsf{S4}^{\mathsf{LP}}_n$, $\mathsf{S5}^{\mathsf{LP}}_n$) and description logics.

We thank Karen Kletter for editing the linguistic aspect of the paper and all the anonymous reviewers for their helpful comments.

References

Alberucci and Jäger, 2005. Luca Alberucci and Gerhard Jäger. About cut elimination for logics of common knowledge. *Annals of Pure and Applied Logic*, 133(1-3):73–99, 2005.

Andrews, 1981. Peter B. Andrews. Theorem proving via general matings. *JACM*, 28(2):193–214, 1981.

Artemov and Nogina, 2005. Sergei Artemov and Elena Nogina. On epistemic logic with justification. In R. van der Meyden, editor, *Theoretical Aspects of Rationality and Knowledge. Proceedings of the Tenth Conference (TARK 2005), June 10-12, 2005, Singapore.*, pages 279–294. National University of Singapore, 2005.

Artemov, 2004. Sergei Artemov. Evidence-based common knowledge. Technical Report TR-2004018, CUNY Ph.D. Program in Computer Science Technical Reports, November 2004.

Bibel, 1981. Wolfgang Bibel. On matrices with connections. *JACM*, 28(4):633–645, 1981.

Bibel, 1987. Wolfgang Bibel. *Automated Theorem Proving*. Vieweg Verlag, Braunschweig, 2nd edition, 1987.

Cerrito and Mayer, 1997. Serenella Cerrito and Marta Cialdea Mayer. Hintikka multiplicities in matrix decision methods for some propositional modal logics. In *Analytic Tableaux and Related Methods (TABLEAUX 97)*, pages 138–152, 1997.

Fagin *et al.*, 1995. Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. Massachusetts Institute of Technology, 1995.

Hintikka, 1961. Jaakko Hintikka. Modalities and quantification. *Theoria*, 27:119–128, 1961.

Hintikka, 1962. Jaakko Hintikka. *Knowledge and Belief*. Cornell University Press, Ithaca, NY, 1962.

Kreitz and Otten, 1999. Christoph Kreitz and Jens Otten. Connection-based theorem proving in classical and non-classical logics. *Journal for Universal Computer Science, Special Issue on Integration of Deductive Systems*, 5(3):88–112, 1999.

Kreitz and Schmitt, 2000. Christoph Kreitz and Stephan Schmitt. A uniform procedure for converting matrix proofs into sequent-style systems. *Journal of Information and Computation*, 162(1–2):226–254, 2000.

Otten and Kreitz, 1996a. Jens Otten and Christoph Kreitz. T-string-unification: Unifying prefixes in non-classical proof methods. In U. Moscato, editor, *5th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, volume 1071 of *Lecture Notes in Artificial Intelligence*, pages 244–260. Springer Verlag, 1996.

Otten and Kreitz, 1996b. Jens Otten and Christoph Kreitz. A uniform proof procedure for classical and non-classical logics. In G. Görz and S. Hölldobler, editors,

KI-96: Advances in Artificial Intelligence, volume 1137 of *Lecture Notes in Artificial Intelligence*, pages 307–319. Springer Verlag, 1996.

Schmitt and Kreitz, 1995. Stephan Schmitt and Christoph Kreitz. On transforming intuitionistic matrix proofs into standard-sequent proofs. In P. Baumgartner, R. Hähnle, and J. Posegga, editors, *Theorem Proving with Analytic Tableaux and Related Methods: Proc. of the 4th International Workshop TABLEAUX'95*, pages 106–121. Springer, Berlin, Heidelberg, 1995.

Schmitt *et al.*, 2001. Stephan Schmitt, Lori Lorigo, Christoph Kreitz, and Aleksey Nogin. JProver: Integrating connection-based theorem proving into interactive proof assistants. In *International Joint Conference on Automated Reasoning*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 421–426. Springer-Verlag, 2001.

Schmitt, 1999. Stephan Schmitt. *Proof Reconstruction in Classical and Non-classical Logics*. PhD thesis, Technical University of Darmstadt, 1999.

Wallen, 1990. Lincoln A. Wallen. *Automated deduction in nonclassical logics*. MIT Press, Cambridge, MA, USA, 1990.

A Completeness Result for BAN Logic

Mika Cohen Mads Dam

KTH Royal Institute of Technology, Stockholm, Sweden
{*mikac,mfd*}*@imit.kth.se*

Abstract

BAN logic is an epistemic logic for verifying cryptographic protocols. While BAN has been quite successful from a practical point of view, the semantics of the epistemic modality is controversial. Several Kripke semantics have been proposed, but they do not attempt at anything beyond a soundness result. Completeness is prevented by the so called logical omniscience problem: Agents in BAN can draw only feasibly computable consequences of their knowledge, whereas agents in Kripke semantics can draw all logical consequences of their knowledge. To avoid logical omniscience, we index the epistemic possibility relation of Kripke semantics with a message renaming, relating how cipher texts at the current state correspond to cipher texts at the epistemically possible state. An agent is said to know a statement if corresponding statements hold at epistemically possible states. We obtain completeness with respect to message passing systems and decidability by transferring canonical model and filtration constructions from Kripke semantics.

Keywords: BAN Logic, Completeness, Decidability, Logical Omniscience

1 Introduction

BAN logic [4] is an epistemic logic proof system for reasoning about cryptographic protocols. Since BAN was introduced in the late eighties, a substantial amount of work has been done applying, varying and clarifying BAN (cf. [1,2,5,6,8,10,11,12,13,15]). However, BAN's central language construct, the epistemic modality, has no agreed upon semantics. The confusion around the semantics hampers the application of semantically based methods, and makes it difficult to evaluate variations to the proof system.

Any interpretation of cryptographic knowledge faces the so-called logical omniscience problem [7]: According to the intended, informal meaning of knowledge, agents can only perform computationally feasible cryptographic

⋆ Work supported by the Swedish Research Council grants 621-2003-2597 and 622-2003-6108

calculations, whereas in the standard semantics for knowledge, Kripke semantics, agents draw arbitrary logical inferences, including computationally unjustified cryptographic calculations. To illustrate, under BAN's idealized cryptography we get the validity $\mathit{fresh}\, M \models \mathit{fresh}\, \{M\}_K$, which, in Kripke semantics, yields the entailment $a\ \mathit{knows}\ \mathit{fresh}\, M \models a\ \mathit{knows}\ \mathit{fresh}\, \{M\}_K$. However, the latter entailment goes against the intended meaning in BAN, since in BAN agent a can know that M is inside $\{M\}_K$ only when a knows K. From the point of view of modal logic, the example shows the failure of the rule of normality that allows inference of an entailment $a, \mathit{knows}\, F \models a\, \mathit{knows}\, F'$ from the entailment $F \models F'$. We refer to [5] for additional counterexamples to the rule of normality.

In Kripke semantics, the knowledge of an agent a at a state s is determined by the data $s|a$ available to a at s. An agent knows a statement if her available data entail the statement. Thus, logical omniscience (rule of normality) is inescapable, no matter how $s|a$ is defined. As it happens, all semantics for BAN-like logics, except for [5], are based on Kripke semantics [1,6,10,11,12,13,15]. Therefore, as the above example illustrates, every extension to BAN is incomplete with respect to these semantics, assuming the extension is faithful to the intended meaning in BAN. Indeed, existing work have so far been limited to soundness results.

Recently, we proposed a generalized Kripke semantics that avoids logical omniscience [5]. There, we established some of the basic properties of the semantics, including soundness. Here we extend this work to show that a faithful version of BAN is complete with respect to message passing systems [7], and that it is decidable. We emphasize that completeness for original BAN [4] cannot be expected, since original BAN, as its authors make clear, leaves out rules that are validated by any reasonable semantics.

The key intuition is that two different cipher texts can be indistinguishable to an agent, due to her limited decryption power. Thus, a cipher text $\{M\}_K$ which the agent cannot decrypt might as well, for all the agent knows, be some other cipher text $\{M'\}_{K'}$. In this sense, $\{M'\}_{K'}$ is an epistemically possible interpretation of $\{M\}_K$. More generally, we can think of a 1-1 mapping r between messages as a joint interpretation of all messages. For each state and agent we identify, based on the keys available to the agent, a set of epistemically possible joint interpretations. We say that agent a knows a statement $F(\{M\}_K)$ at state s, if for every possible joint interpretation r, the reinterpreted data $r(s|a)$ entails the reinterpreted statement $F(r(\{M\}_K))$. This semantics departs from Kripke semantics, in that we check the renamed cipher text $r(\{M\}_K)$ instead of the predicated cipher text $\{M\}_K$. As a result, logical omniscience is avoided.

We present an axiomatization of validity with respect to our semantics. The axiomatization uses standard modal axioms *K*, *T*, *S4* and *S5*, but excludes the rule of necessitation. The latter is weakened so that agents can only infer "feasibly computable" theorems. The axiomatization employs, in addition,

some axioms specific to message passing systems, including an epistemic axiom stating that an agent knows if she sent or received a message. We obtain completeness and decidability by transferring canonical model and filtration constructions from Kripke semantics.

Our semantics is a variation on counterpart semantics, a semantics for first order modal logic due to [9]. Epistemically possible mappings between messages are related to the message congruences of [1], and to the states of knowledge and belief of [3,14].

2 Message Passing Systems

In a message passing system agents take turns to send messages, receive messages or perform unspecified internal actions [7]. For the sake of brevity, we omit the customary initialization action, i.e., a special action that establishes initial shared or private secrets. We also use a simple, perhaps simplistic, notion of session along the lines of [1].

Assume a non-empty set of keys $K, K', ...$, a non-empty set of plain-texts $T, T', ...$, including a finite set A of agent names $a, b, ...$ Messages are generated by:

$$M ::= \ K \mid T \mid M \cdot M \mid \{M\}_K$$

where $\cdot$ represents pairing and $\{\ \}$ represents symmetric encryption. The sub-message relation $\geq$ is the smallest binary relation on messages such that $M \geq M$, $\{M\}_K \geq M$, $\{M\}_K \geq K$, $M \cdot M' \geq M$ and $M \cdot M' \geq M'$. A message space is a non-empty set of messages closed under $\geq$, i.e., if $M \geq M'$ then the space contains M' if it contains M. We fix a finite message space: A message M is, from now on, a message in this fixed space.

Actions are:

$$\sigma ::= \ a\,\texttt{sends}\,M \mid a\,\texttt{receives}\,M \mid a\,\mathit{int}\,M \mid \texttt{begin}\,\texttt{session}$$

where agent a is in A or is the special agent `env`, the environment, and *int* ranges over a finite set of internal action types. Write $\sigma(M)$ for an action in the message term M. A history, or action trace, is a finite sequence h of actions. Write $\mathit{Actions}(h)$ for the set of all actions in h: $\mathit{Actions}(\epsilon) = \emptyset$ and $\mathit{Actions}(h \cdot \sigma) = \mathit{Actions}(h) \cup \{\sigma\}$. A message passing system, or *system* for short, is a non-empty set H of histories.

The local history of agent a in history h, in symbols $h|a$, is the sequence of actions performed by a in h: $\epsilon|a = \epsilon$, if $\sigma \in \{a\,\texttt{sends}\,M,\ a\,\texttt{receives}\,M,\ a\,\mathit{int}\,M\}$ then $(h \cdot \sigma)|a = h|a \cdot \sigma$ else $(h \cdot \sigma)|a = h|a$.

A key assignment κ on H assigns a set $\kappa(a, h)$ of keys, the keys used by a at h for encryption and decryption, to each agent $a \in A$ and history $h \in H$. A model on H is a pair $\mathcal{M} = \langle H, \kappa \rangle$, where κ is a key assignment on H. We leave κ undefined until Section 5.

3 Language of Full Propositional BAN

An atomic statement is a predicate p applied to a message M:

$$p(M) ::= \ a \textit{ received } M \mid a \textit{ rec } M \mid a \textit{ sent } M \mid a \textit{ sen } M \mid \textit{old } M$$

The meaning of atomic statements is straightforward: $a \textit{ received } M$ holds if agent a received message M from the network, $a \textit{ rec } M$ holds if M is a sub-message of some message a received. [1] The intended meaning of $a \textit{ sent } M$ and $a \textit{ sen } M$ are analogous. The statement $\textit{old } M$ holds if M is a sub-message of some message sent in an old session, i.e., prior to a `begin session` event. We note that, unlike BAN like languages in the literature, atomic statements do not involve any notion of "feasible cryptographic computation". In particular, atomic statements do not depend on the keys used by agents. Statements are generated by:

$$F ::= \ p(M) \mid \Box_a F \mid F \wedge F \mid \neg F$$

where $\Box_a$ is the epistemic modality for a, read "agent a knows that". Define disjunction ($\vee$) and implication ($\rightarrow$) in the usual way. Let $G \subseteq A$. We introduce BAN predicates as abbreviations, similar to [10]:

- $a \textit{ sees } M = \Box_a a \textit{ rec } M$
- $a \textit{ said } M = \Box_a a \textit{ sen } M$
- $M \textit{ secret of } G = \bigwedge_{a \notin G} \neg\, a \textit{ sees } M$
- $\textit{fresh } M = \neg \textit{ old } M$

The language defined above differs somewhat from the original BAN [4]. Firstly, original BAN has some constructs for asymmetric cryptography. Secondly, there is no negation operator in original BAN. Thirdly, original BAN includes messages that contain statements, so called idealized messages. Fourth, we have dropped predicates *good* and *controls*; *good* is dropped because it is analogous to *secret*, and *controls* is dropped since it becomes superfluous when the epistemic modality is interpreted as knowledge rather than belief [10].

4 Semantics

In Kripke semantics, the epistemic modality $\Box_a$ is interpreted through an epistemic possibility relation $\longrightarrow_a$ between states, in our case histories. Intuitively, $h \longrightarrow_a h'$ means that at history h agent a could, for all she knows, be at h'. The epistemic possibility relation has a default definition in computer science, due to [7]: $h \longrightarrow_a h'$, if and only if, $h|a = h'|a$. However, because of the the limited decryption power of agents, there may be more than one way for the agent to interpret the cipher texts she has sent, received

[1] If we include initialization actions in histories, $a \textit{ rec } M$ also holds if M is a sub-message of some "initial secret" of a.

or otherwise acted upon. Consider, for example, a model $\mathcal{M}_0 = \langle H_0, \kappa_0 \rangle$ where H_0 has three execution histories, $h_0 = b\,\mathsf{sends}\,\{M\}_K \cdot a\,\mathsf{receives}\,\{M\}_K$, $h_1 = b\,\mathsf{sends}\,\{M'\}_{K'} \cdot a\,\mathsf{receives}\,\{M'\}_{K'}$ and $h_2 = b\,\mathsf{sends}\,\{M\}_K$, and agent a does not use any keys, i.e., $\kappa_0(a, h_0) = \kappa_0(a, h_1) = \kappa_0(a, h_2) = \emptyset$. Even though $h_0|a \neq h_1|a$, it may still be reasonable to say that at h_0, agent a could, for all she knows, be at h_1. Based on such intuitions, the AT semantics [1] and its descendants [13,15] drop the requirement of local history identity, in effect hiding cryptographically inaccessible parts of the local history from the agent herself.

We depart from AT semantics, and from Kripke semantics in general, by extending the epistemic uncertainty to predicated cipher texts. To illustrate, the cipher text $\{M\}_K$ at h_0 in the above model $\mathcal{M}_0$, could, for all agent a knows, be the cipher text $\{M'\}_{K'}$ at h_1. Everything a observes of $\{M\}_K$ at h_0, a also observes of $\{M'\}_{K'}$ at h_1. In this sense, $\{M'\}_{K'}$ at h_1 corresponds for a to $\{M\}_K$ at h_0. In general, a message sequence $M'_0, M'_1, \cdots$ at h_1 may correspond for a to another message sequence $M_0, M_1, \cdots$ at h_0. In order to keep track of message correspondences, we therefore relativize the epistemic possibility relation to a message renaming, a $1-1$ function r on the set of messages. Informally, $h \longrightarrow_a^r h'$ if any message sequence $M_0, M_1, \cdots$ at h could, for all a knows, be the sequence $r(M_0), r(M_1), \cdots$ at h'. We extend a renaming to statements by renaming the message terms inside a statement: $r(p(M)) = p(r(M))$, $r(\Box_a F) = \Box_a r(F)$, etc. Then renamings are extended to sets of statements by renaming each statement in a set: $r(\Delta) = \{r(F) \mid F \in \Delta\}$ for any set Δ of statements, and, finally, to sets of messages by renaming each message in a set: $r(\Pi) = \{r(M) \mid M \in \Pi\}$ for any set Π of messages.

Assuming the the relativized epistemic possibility relation, we say that an agent knows a statement if corresponding statements hold at epistemically possible histories:

$$h \models_{\mathcal{M}} \Box_a F \Leftrightarrow \forall r : \forall h' \in H : h \longrightarrow_a^r h' \Rightarrow h' \models_{\mathcal{M}} r(F)$$

The break with Kripke semantics should be clear. We check a renamed statement $r(F)$ at h', and not the original statement F.

For $h \longrightarrow_a^r h'$ to hold, we require that r respects the observations (actions) of a in h as well as the message structure accessible through a:s keys at h. The former requirement means that

- $r(h|a) = h'|a$

where r is extended to histories by point-wise renaming each message acted upon in the history: $r(\epsilon) = \epsilon$ and $r(h \cdot \sigma(M)) = r(h) \cdot \sigma(r(M)))$. For the latter requirement, we simply assume a *transparency relation* $\lhd$, which determines if a renaming r respects structure accessible with a set Π of keys: $r \lhd \Pi$, if Π cannot distinguish a message, a history or a statement from its renaming under r. We leave the definition of $\lhd$ open, merely insisting on four requirements:

- $r \lhd \Pi,\ \Pi \supseteq \Pi' \Rightarrow r \lhd \Pi'$ (Monotonicity)

- $\iota \lhd \Pi$, where ι is identity of messages (Reflexivity)
- $r \lhd \Pi,\ r' \lhd r(\Pi) \Rightarrow (r' \circ r) \lhd \Pi$ (Transitivity)
- $r \lhd \Pi \Rightarrow r^{-1} \lhd r(\Pi)$ (Symmetry)

For example, we may stipulate that $r \lhd \Pi$, if and only if, r respects encryption with keys in Π:

(i) $k \in \Pi \Rightarrow r(\{M\}_k) = \{r(M)\}_{r(k)}$ (Encryption)

and r respects clear text constructions:

(ii) $r(M \cdot M') = r(M) \cdot r(M')$ (Pairing)

(iii) $r(T) = T$, plain text T (Plain text)

So defined, $\lhd$ is monotone, reflexive, transitive and symmetric [5]. We say that $\lhd$ respects encryption, if, for all r and Π, $r \lhd \Pi$ implies that r respects encryption with keys in Π. Similarly, we say that $\lhd$ respects pairing/plain text, if, for all r and Π, $r \lhd \Pi$ implies that r respects paring/ plain text. Note that any $\lhd$ is finite. (Since the message space is finite, there are finitely many sets Π of keys and finitely many renamings r.) Putting the two requirements on $\longrightarrow_a^r$ together, we stipulate:

$$h \longrightarrow_a^r h' \text{ in } \mathcal{M} \Leftrightarrow r(h|a) = h'|a \ \&\ r \lhd \kappa(a, h)$$

Truth conditions for Boolean operators and atomic statements are as expected: $h \models_{\mathcal{M}} \neg F$, if and only if, $h \not\models_{\mathcal{M}} F$; $h \models_{\mathcal{M}} F \wedge F'$, if and only if, $h \models_{\mathcal{M}} F$ and $h \models_{\mathcal{M}} F'$; $h \models_{\mathcal{M}} a \mathit{\ received\ } M$, if and only if, $a\,\texttt{receives}\,M \in \mathit{Actions}(h)$; $h \models_{\mathcal{M}} a \mathit{\ sent\ } M$, if and only if, $a\,\texttt{sends}\,M \in \mathit{Actions}(h)$; $h \models_{\mathcal{M}} a \mathit{\ rec\ } M$, if and only if, M is a sub-message of some M' such that $a\,\texttt{receives}\,M' \in \mathit{Actions}(h)$; $h \models_{\mathcal{M}} a \mathit{\ sen\ } M$, if and only if, M is a sub-message of some M' such that $a\,\texttt{sends}\,M' \in \mathit{Actions}(h)$; $h \models_{\mathcal{M}} \mathit{old\ } M$, if and only if, for some h' and h'', $h = h' \cdot \texttt{begin session}{\cdot}h''$ and $h' \models_{\mathcal{M}} a \mathit{\ rec\ } M \vee a \mathit{\ sen\ } M$ for some $a \in A \cup \{\texttt{env}\}$. Finally, statement F is valid in model $\mathcal{M}$ if $h \models_{\mathcal{M}} F$, for all $h \in H$.

Returning to the example $\mathcal{M}_0$ above, $h_0 \models_{\mathcal{M}_0} \Box_a a \mathit{\ received\ } \{M\}_K$, since if $h_0 \longrightarrow_a^r h_0$ then $r(\{M\}_K) = \{M\}_K$ and if $h_0 \longrightarrow_a^r h_1$ then $r(\{M\}_K) = \{M'\}_{K'}$. On the other hand, $h_0 \not\models_{\mathcal{M}_0} \Box_a a \mathit{\ rec\ } M$, since there is some r such that $h_0 \longrightarrow_a^r h_1$ and $r(M) = M$, assuming that $\lhd$ is non-degenerate. As the implication $a \mathit{\ received\ } \{M\}_K \rightarrow a \mathit{\ rec\ } M$ is valid, the example illustrates that agents need not be logically omniscient.

The relativized possibility relation implicitly contains an AT-like possibility relation: $h \longrightarrow_a h'$, if and only if, there exists a renaming r such that $h \longrightarrow_a^r h'$. Thus, $h \longrightarrow_a h'$ if $h'|a$ is a possible interpretation of $h|a$. With the existential quantification over renamings r we lose the information how cipher texts at h may correspond for a to cipher texts at h'.

5 Inductive Key Assignment

We left the key assignment κ open. We now stipulate that the keys used are the keys seen. This requires a recursive definition, since *a sees* is defined in terms of $\Box_a$, which in turn is interpreted through κ itself. An inductive, rather than a coinductive definition is appropriate, since κ should assign the set of keys that the agent has gathered some positive information about. We call a key assignment κ inductive on system H, if κ is a minimal (with respect to point-wise subset inclusion) key assignment such that

$$K \in \kappa(a,h) \Leftrightarrow h \models_{\langle H,\kappa\rangle} \Box_a a \, rec \, K$$

for all $a \in A$, $h \in H$ and keys K. A model $\langle H, \kappa\rangle$ is inductive if κ is inductive on H.

Theorem 5.1 (Existence of Inductive Key Assignment) *There exists a unique inductive key assignment on every message passing system.*

Proof. An inductive key assignment on H is, by definition, a least fixed point of the following function f assigning a key assignment $f(\kappa)$ to every key assignment κ: $f(\kappa)(a,h) = \{K \mid h \models_{\langle H,\kappa\rangle} \Box_a \, a \, rec \, K\}$. Function f is monotone, as $\lhd$ is monotone and the extension of *a rec* is independent of the key assignment. [2] Therefore, f has a unique least fixed point. □

In inductive models, the relativized epistemic possibility relation generalizes an equivalence relation on histories.

Lemma 5.2 *In inductive models:*

(i) $h \longrightarrow_a^{\iota} h$

(ii) $h \longrightarrow_a^{r} h'$, $h' \longrightarrow_a^{r'} h'' \Rightarrow h \longrightarrow_a^{r' \circ r} h''$

(iii) $h \longrightarrow_a^{r} h' \Rightarrow h' \longrightarrow_a^{r^{-1}} h$

Proof. (1): From reflexivity of $\lhd$. (2): From transitivity of $\lhd$ and fixed point induction. (3): From symmetry of $\lhd$ and fixed point induction. For more detail, we refer to [5]. □

As Lemma 5.2.iii shows, the apparent asymmetry in the definition of the epistemic possibility relation disappears in inductive models. From now on, if no key assignment is given, we assume the inductive key assignment: $h \models_H F$, if and only if, $h \models_{\mathcal{M}} F$ for the inductive model $\mathcal{M}$ based on H; F is valid in H if $h \models_H F$, for all $h \in H$.

Many semantics proposed for BAN logics give a more straightforward, operational definition of seen messages and used keys. Roughly, a message is seen if it was received, or if it is the first or second pairing component of a

[2] This independence is preserved if we include initialization actions in histories, and make *a rec* apply also to sub-messages of "initial secrets" of a.

seen message, or if it is the body of a seen cipher text locked with a seen key. A key is used if it is seen. To illustrate, at a history $h_0 = b \,\mathsf{sends}\, K \cdot \{K' \cdot \{K''\}_{K''}\}_K \cdot a \,\mathsf{receives}\, K \cdot \{K' \cdot \{K''\}_{K''}\}_K$ agent a uses the keys K and K'. In fact, the inductive key assignment is at least as inclusive as this operational key assignment, assuming that $\lhd$ respects pairing and encryption. For instance, key K is in the second iteration of the fixed point definition of $\kappa(a, h_0)$, and K' is in the third. A forthcoming full version of [5] shows the inclusion and under what conditions the inductive and the operational key assignments coincide. We refer to [5] for examples of when the operational key assignment might be inappropriately weak.

6 BAN Theories

A BAN theory is a set L of statements containing all axioms and closed under all rules in Table 1, where axiom *Taut* consists of all tautologies from classical propositional logic, and a *sees* $\Pi = \bigwedge_{K \in \Pi} a$ *sees* K, with the empty conjunction abbreviating some tautology. When $F \in L$, we write $\vdash_L F$ and say that F is a theorem of L. The axioms and rules in Table 1 isolate "feasible cryptographic computation" to one point, namely renaming necessitation (*RNec*). According to *RNec*, an agent knows all theorems that are preserved under renamings transparent to seen keys. Since there are finitely many renamings, *RNec* is finitary, i.e., involves a finite set of premisses. The introspection axiom I says that an agent knows if she sent or received a message. Axioms K, T, *4* and *5* are standard. The remaining axioms and rule are non-epistemic. Axiom *Mono* says that *a rec*, *a sen* and *old* are monotone with respect to $\geq$. The disjunctions in axioms *R2* and *S2* are finite, since there are only finitely many messages.

Let Δ be a set of statements, possibly infinite. Write $\Delta \vdash_L F$, if there is a finite subset $\{F_1, ..., F_n\} \subseteq \Delta$ such that $\vdash_L F_1 \to (F_2 \to (\cdots \to (F_n \to F) \cdots))$. Write $\Box_a \Delta$ for $\{\Box_a F \mid F \in \Delta\}$. We have the following weakening of normality.

Lemma 6.1 (Renaming Normality) *If* $r(\Delta) \vdash_L r(F)$ *for all* $r \lhd \Pi$, *then* a *sees* Π, $\Box_a \Delta \vdash_L \Box_a F$.

Proof. Assume $r(\Delta) \vdash_L r(F)$, $\forall r \lhd \Pi$. Since the message space is finite, there are only finitely many renamings. Let $r_1, ..., r_n$ be all renamings r such that $r \lhd \Pi$. For each $i \in \{1, ..., n\}$ there is a finite $\Delta_i \subseteq \Delta$ such that $r_i(\Delta_i) \vdash_L r_i(F)$. Thus for each $i \in \{1, ..., n\}$: $r_i(\Delta_1, ..., \Delta_n) \vdash_L r_i(F)$. Since $\Delta_1, ..., \Delta_n$ is finite, by rule *RNec* and axiom K: a *sees* Π, $\Box_a \Delta_1, ..., \Box_a \Delta_1 \vdash_L \Box_a F$. Since $\Delta_i \subseteq \Delta$: a *sees* Π, $\Box_a \Delta \vdash_L \Box_a F$. □

Since the renaming normality rule only closes knowledge under "feasibly computable" logical implications, BAN theories can avoid the absurdities of

K	$\Box_a(F \to F') \to \Box_a F \to \Box_a F'$
T	$\Box_a F \to F$
4	$\Box_a F \to \Box_a \Box_a F$
5	$\neg\Box_a F \to \Box_a \neg\Box_a F$
RNec	$\dfrac{r(F),\ \ \forall r \lhd \Pi}{a\ sees\ \Pi \to \Box_a F}$
I	$\pi_a \to \Box_a \pi_a,\ \ \pi_a \in \{a\ received\ M, a\ sent\ M\}$
Mono	$p(M) \to p(M'),\ M \geq M',\ p \in \{a\ rec, a\ sen, old\}$
R	$a\ received\ M \to a\ rec\ M$
S	$a\ sent\ M \to a\ sen\ M$
R2	$a\ rec\ M \to \bigvee_{M' \geq M} a\ received\ M'$
S2	$a\ sen\ M \to \bigvee_{M' \geq M} a\ sent\ M'$
Taut	F, F tautology from propositional logic
MP	$\dfrac{F \quad F \to F'}{F'}$

Table 1
Axioms and Rules

logical omniscience. For instance, a BAN theory need *not* contain:

$$a\ rec\ M \to \Box_a a\ rec\ M \tag{1}$$

$$\Box_a fresh\ M \to \Box_a fresh\ \{M\}_K \tag{2}$$

This follows from Soundness Theorem 7.1. In contrast, if a BAN theory L satisfies the rule of normality, i.e., if $\Delta \vdash_L F$ implies that $\Box_a \Delta \vdash_L \Box_a F$, then L contains (1) and (2); Normality and axioms I, R, $R2$ and *Mono* give (1), while normality and axiom *Mono* yield (2). As the following fact illustrates, BAN theories contain a significant part of original BAN [4].

Corollary 6.2 *Assume $\lhd$ respects pairing and encryption. Then every BAN theory contains the following, where $s_a \in \{a\ sees, a\ said\}$*

(i) $\Box_a fresh\ M \to \Box_a fresh\ (M \cdot M')$

(ii) $\Box_a fresh\ M \to (a\ sees\ K \to \Box_a fresh\ \{M\}_K)$

(iii) $\Box_b s_a\ (M \cdot M') \to \Box_b s_a\ M$

(iv) $\Box_b s_a\ (M \cdot M') \to \Box_b s_a\ M'$

(v) $s_a\,\{M\}_K \rightarrow (a\ sees\, K \rightarrow s_a\, M)$

(vi) $s_a\, M \rightarrow \Box_a s_a\, M$

(vii) $\neg s_a\, M \rightarrow \Box_a \neg s_a\, M$

(viii) $a\ received\, M \rightarrow a\ sees\, M$

Proof. (1), (3) and (4): Axiom *Mono*, renaming normality (Lemma 6.1) and $\lhd$ respects pairing. (2) and (5): Axiom *Mono*, renaming normality and $\lhd$ respects encryption. (6): Axiom *4*. (7): Axiom *5*. (8): Axioms I and R and renaming normality. □

The well-known message meaning rule (in our setting: axiom) of BAN is conspicuously absent in Corollary 6.2. To obtain this axiom for a group G of agents , we need to assume an origination axiom for G:

$$K\ secret\ of\ G \rightarrow (b\ rec\ \{M\}_K \rightarrow \bigvee_{a \in G} (a\ said\ \{M\}_K \wedge a\ sees\ K) \qquad (3)$$

as well as an honesty axiom for G, where *from* is some special plain text atom:

$$\neg a\ said\ from \cdot b \cdot M, \text{ whenever } a \neq b, a \in G \qquad (4)$$

Corollary 6.3 (Message Meaning Axiom) *Assume $\lhd$ respects pairing, encryption and plain texts. Every BAN theory containing the origination axiom (3) and honesty axiom (4) for agent group G also contains:*

$$a\ sees \{from \cdot b \cdot M\}_K,\ a\ sees\, K,\ \Box_a K\, secret\ of\ G \rightarrow \Box_a\, b\, said\, M,\ b \in G$$

Proof. Immediate from renaming normality (Lemma 6.1) and that $\lhd$ respects plain texts, pairing and encryption. □

The message meaning axiom in Corollary 6.3 weakens the original in [4] by adding to the antecedent that agent a uses (sees) the key K, as in, for instance, [1,6,8]. We briefly illustrate BAN theories with protocol specific axioms.

Example 6.4 Consider the Needham-Schröder Shared Key Protocol between principals a and b and with key server s. If the server sends the cipher text $\{N \cdot b \cdot K \cdot M\}_{K_a}$, and K_a is a:s server key, then the server generated K for a and b:

$$s\ sen \{N \cdot b \cdot K \cdot M\}_{K_a},\ K_a\ secret\ of\ a \cdot s,\ fresh\, N \rightarrow K\ secret\ of\ a \cdot b \cdot s$$

If a BAN theory contains this protocol specific axiom, for all keys N, K and K_a and all messages M, contains the origination axiom 3 for agent group $\{a, s\}$, and assuming $\lhd$ respects pairing, encryption and plain texts, the BAN theory also contains the authentication specification:

$$a \textit{ sees } \{N \cdot b \cdot K \cdot \{K \cdot a\}_{K_b}\}_{K_a},\ \neg a \textit{ said } \{N \cdot b \cdot K \cdot \{K \cdot a\}_{K_b}\}_{K_a},$$
$$a \textit{ sees } K_a,\ \Box_a K_a \textit{secret of } a \cdot s,\ \Box_a \textit{fresh } N$$
$$\rightarrow \Box_a K \textit{ secret of } a \cdot b \cdot s$$

stating that if a sees the message from the server, did not send the same message herself, knows the key to this message, and knows that the nonce inside is fresh, then a knows that the key provided inside is secret between a, b and s. The derivation is by way of renaming normality (Lemma 6.1) and Corollary 6.2.

Corollary 6.3 and Example 6.4 suggest that we might be interested in BAN theories generated by adding various theory bases, *finite* sets $\boldsymbol{A}$ of statements. Note that the origination and honesty schemata in Corollary 6.3, as well as the protocol specific axiom schemata in Example 6.4, are indeed finite, since there are only finitely many messages. We define the BAN theory induced by $\boldsymbol{A}$, in symbols $L\boldsymbol{A}$, as the smallest BAN theory containing the finite set $\boldsymbol{A}$.

7 Main Results

Write $\|\Delta\|$ for the set of all message passing systems validating all statements in Δ. BAN theory L is sound with respect to a class $\mathcal{C}$ of message passing systems, if $\mathcal{C} \subseteq \|L\|$. BAN theory L is complete with respect to $\mathcal{C}$, if L contains all statements valid in all systems in $\mathcal{C}$.

Theorem 7.1 (Soundness) *$L\boldsymbol{A}$ is sound with respect to $\|\boldsymbol{A}\|$.*

Proof. Rule *RNec*: Since $\triangleleft$ is monotone. Axioms T, 4 and 5: From Lemma 5.2. Axiom I: Assume $h \models a \textit{ received } M$, i.e., $a\,\texttt{receives}\,M \in \textit{Actions}(h|a)$. Pick any $h' \in H$ and renaming r such that $h \longrightarrow_a^r h'$. Then $r(h|a) = h'|a$. Therefore, $a\,\texttt{receives}\,r(M) \in \textit{Actions}(h'|a)$, i.e., $h' \models a \textit{ received } r(M)$. Since h' and r were chosen at random, $h \models \Box_a a \textit{ received } M$. Analogously for $a \textit{ sent } M$. Remaining axioms and rule MP are immediate. □

Theorem 7.2 (Completeness) *$L\boldsymbol{A}$ is complete with respect to $\|\boldsymbol{A}\|$.*

Proof. Section 8. □

Thus, the protocol base semantically guarantees a specification only if the specification is a theorem. Contrast this with the usual verification practice in BAN, based on an open ended proof system: If your specification is unprovable, you conclude that either the base logic or your protocol assumptions are too weak [4].

Theorem 7.3 (Decidability) *$L\boldsymbol{A}$ is decidable.*

Proof. Section 8. □

8 Proof of Completeness and Decidability

We show completeness and decidability by transferring canonical model and filtration techniques from Kripke semantics. As a first step, we lift the semantics from message passing models to a more general class of structures, counterpart models. Next, we build a canonical counterpart model $\mathcal{C}_L$ that validates precisely the theorems of a given BAN theory L. For any finite set Γ of statements, $\mathcal{C}_L$ is transformed, while preserving truth values in Γ, into a finite message passing system $H_{L,\Gamma}$.

8.1 Canonical Counterpart Model

A counterpart model is a triple $\mathcal{C} = \langle W, \longrightarrow, I\rangle$, where W is a non-empty set of worlds (states), $\longrightarrow_a^r \subseteq W \times W$ for each agent $a \in A$ and renaming r, and $I(p, w)$ is a set of messages, the messages satisfying predicate p at w. Intuitively, $w \longrightarrow_a^r w'$ says that any M at w, could, for all a knows, be $r(M)$ at w'. The semantics of Section 4 is generalized in the obvious way: $w \models_{\mathcal{C}} p(M) \Leftrightarrow M \in I(p, w)$ and $w \models_{\mathcal{C}} \Box_a F \Leftrightarrow \forall r \forall w \in W : w \longrightarrow_a^r w' \Rightarrow w' \models_{\mathcal{C}} r(F)$. Truth conditions for boolean operators are unchanged.

Counterpart models are used in counterpart semantics for first order modal logic due to [9]. The truth condition above for knowledge is unorthodox, treating message terms the way counterpart semantics treats free variables.

Next, we build a canonical counterpart model that validates precisely the theorems of a given BAN theory. Assume a BAN theory L. A set Δ of statements is consistent if there is no statement $\neg F$ such that $\Delta \vdash \neg F$ and $\Delta \vdash F$. Δ is maximal consistent if there is no consistent set Δ' such that $\Delta' \supset \Delta$. Using the standard Lindenbaum construction we obtain:

Lemma 8.1 (Extension Lemma) *If $\Delta \nvdash F$, there is a maximal consistent set $\Delta' \supseteq \Delta$ such that $F \notin \Delta'$.*

Write $Keys(a, \Delta)$ for the set $\{K \mid a \text{ sees } K \in \Delta\}$. The canonical counterpart model for BAN theory L is $\mathcal{C}_L = \langle W_L, \underset{L}{\longrightarrow}, I_L\rangle$, where

- W_L is the set of all maximal L-consistent sets
- $w \underset{L}{\longrightarrow}_a^r w' \Leftrightarrow r \triangleleft Keys(a, w) \wedge \forall F : \Box_a F \in w \Rightarrow r(F) \in w'$
- $I_L(w, p) = \{M \mid p(M) \in w\}$

Lemma 8.2 (Truth lemma) $w \models_{\mathcal{C}_L} F \Leftrightarrow F \in w$.

Proof. By induction in (the number of statement operators in) F, using renaming normality (Lemma 6.1). The base case, for atomic F, is immediate. The induction step, for boolean operators: uses standard properties of maximal consistent sets. For the epistemic modality let $w|a$ be the set $\{F \mid \Box_a F \in w\}$. For the only-if direction first:

$\Box_a F \notin w$

$\Rightarrow r(w|a) \not\vdash r(F)$ & $r \triangleleft Keys(a, w)$, $\exists r$ (By renaming normality) (5)

$\Rightarrow r(w|a) \subseteq w'$ & $r(F) \notin w'$, $\exists w' \in W_L$ (By lemma 8.1) (6)

$\Rightarrow w' \not\models_{\mathcal{C}_L} r(F)$ (By the ind. hyp.) (7)

$\Rightarrow \forall F : \Box_a F \in w \Rightarrow r(F) \in w'$ (By (6)) (8)

$\Rightarrow w \underset{L}{\longrightarrow}{}^{r}_{a} w'$ (By (5) and (8)) (9)

$\Rightarrow w \not\models_{\mathcal{C}_L} \Box_a F$ (By (7) and (9))

For the if-direction:

$\Box_a F \in w$ & $w \underset{L}{\longrightarrow}{}^{r}_{a} w'$ & $w' \in W_L$

$\Rightarrow r(F) \in w'$

$\Rightarrow w' \models_{\mathcal{C}_L} r(F)$ (By the ind. ass.)

$\Rightarrow w \models_{\mathcal{C}_L} \Box_a F$ (By the assumptions)

□

8.2 Canonical Message Passing System

Given a finite set Γ of statements, we transform the canonical model into a finite inductive message passing model, while preserving truth values in Γ. Each $w \in W_L$ is transformed, in three steps, into a collection of histories that satisfy $w \cap \Gamma$. In the first step, w is collapsed into the finite set $w \cap \Gamma$. Secondly, this finite set is sequenced in different ways, yielding a collection of statement sequences. Thirdly, statements (in the sequences) are replaced by actions that "ground" them.

The first transformation step is trivial. For the second step, we consider various enumerations of statements. An enumeration of all statements in the language is *admissible* if it makes all atomic statements of the form *old M* appear prior to other kind of statements. In what follows, whenever we talk about *an enumeration e* we mean an admissible enumeration *e*. Any *e* induces a sequence $e(\Delta)$ from a set Δ of statements, obtained by removing all non-members of Δ from the enumeration *e*. For the third transformation step, we define an internal action to be of the form $a\,\texttt{int}\,F$, where F is any statement and $a \in A \cup \{\texttt{env}\}$.[3] We then take sequences s of statements to histories under a mapping *hist* as follows:

(i) $hist(\epsilon) = \epsilon$

(ii) $hist(s \cdot old\ M) = hist(s) \cdot \texttt{env sends}\ M \cdot \texttt{begin session}$

(iii) $hist(s \cdot a\ received\ M) = hist(s) \cdot a\ \texttt{receives}\ M$

(iv) $hist(s \cdot a\ sent\ M) = hist(s) \cdot a\ \texttt{sends}\ M$

[3] This assumes a slightly more general definition of internal action than that of Section 2. Alternatively, we could introduce $a\,\texttt{int}\,F$ as an abbreviation for an internal action of the form $a\ int\ M$.

(v) $hist(s \cdot \Box_a F) = hist(s) \cdot a \texttt{ int } F$

(vi) $hist(s \cdot F) = hist(s)$, otherwise

Condition (2) assures that every message which s claims is old, is old at $hist(s)$ according to the semantics of predicate *old*. Condition (3) sees to it that every message that s claims a received, a received at $hist(s)$ according to the semantics of predicate *receive*. Analogously for condition (4). Condition (5) places a token, or evidence, in $hist(s)|a$, for every statement s claims a knows.

We denote the set $w \cap \Gamma$ by $[w]_\Gamma$, or simply $[w]$, when Γ is clear from the context. Write $hist_e([w])$ for $hist(e([w]))$. The canonical message passing model for finite Γ and BAN theory L is $\mathcal{M}_{L,\Gamma} = \langle H_{L,\Gamma}, \kappa_{L,\Gamma} \rangle$, where

(i) $H_{L,\Gamma} = \{hist_e([w]) \mid w \in W_L,\ \exists e\}$.

(ii) $\kappa_{L,\Gamma}(a, hist_e([w])) = Keys(a, [w])$

To ensure that condition (2) is well-defined we need some conditions on Γ. We say Γ is *adequate* if Γ is finite, Γ is closed under sub-statements (if $F \in \Gamma$ and F' is a sub-statement of F then $F' \in \Gamma$), Γ is closed under renamings (if $F \in \Gamma$ and r is any renaming then $r(F) \in \Gamma$), Γ contains all atomic statements and contains $\Box_a a$ *received* M, $\Box_a a$ *sent* M and $\Box_a a$ *rec* M, for all $a \in A$ and messages M.

Lemma 8.3 *Assume Γ is adequate and $w \in W_L$.*

(i) $old\, M \in [w] \Leftrightarrow \exists M' \geq M : old\, M' \in [w]$.

(ii) $a\ rec\, M \in [w] \Leftrightarrow \exists M' \geq M : a\ received\, M' \in [w]$.

(iii) $a\ sen\, M \in [w] \Leftrightarrow \exists M' \geq M : a\ sent\, M' \in [w]$.

(iv) $a\ received\, M \in [w] \Leftrightarrow \Box_a a\ received\, M \in [w]$.

(v) $a\ sent\, M \in [w] \Leftrightarrow \Box_a a\ sent\, M \in [w]$.

Proof. From axioms *R*, *S*, *R2*, *S2*, *Mono*, *I* and *T*. We show case (2), the remaining cases are analogous. $a\ rec\, M \in [w]$, if and only if (Γ is adequate), $a\ rec\, M \in w$, if and only if (*R*, *R2*, *Mono*, w is maximal consistent set), $\exists M' \geq M : a\ received\, M' \in w$, if and only if ($\Gamma$ is adequate), $\exists M' \geq M : a\ received\, M' \in [w]$. □

Lemma 8.4 (Injectivity) *Assume Γ is adequate. $hist_e([w]) = hist_{e'}([w']) \Rightarrow [w] = [w']$, for $w, w' \in W_L$.*

Proof. From Lemma 8.3. □

Lemma 8.4 assures us that the key assignment in $\mathcal{M}_{L,\Gamma}$ is well-defined for adequate Γ. We proceed to show that the transformation $w \curvearrowright hist_e([w])$ preserves truth values from $\mathcal{C}_L$ to $\mathcal{M}_{L,\Gamma}$ for statements in Γ.

Lemma 8.5 (Reflection) *Assume Γ is adequate and $w, w' \in W_L$. The following statements are equivalent:*

(i) $\exists e' : hist_e([w]) \longrightarrow_a^r hist_{e'}([w'])$ *in* $\mathcal{M}_{L,\Gamma}$

(ii) $r \triangleleft Keys(a,[w]) \wedge \forall F : \Box_a F \in [w] \Leftrightarrow \Box_a r(F) \in [w']$.

Proof. From Lemma 8.3. We show the implication from (2) to (1). So assume that $r \triangleleft Keys(a,[w]) \wedge \forall F : \Box_a F \in [w] \Leftrightarrow \Box_a r(F) \in [w']$. We show that a receives $M \in Actions(hist_e([w])|a)$ iff a receives $r(M) \in Actions(hist_e([w'])|a)$, and similarly for internal and send actions. The proofs in the latter cases are similar and left to the reader. For receive actions:

a receives $M \in Actions(hist_e([w])|a)$
$\Leftrightarrow a$ receives $M \in Actions(hist_e([w]))$
$\Leftrightarrow a$ *received* $M \in [w]$
$\Leftrightarrow \Box_a a$ *received* $M \in [w]$ (By lemma 8.3)
$\Leftrightarrow \Box_a a$ *received* $r(M) \in [w']$ (By the assumption)
$\Leftrightarrow a$ *received* $r(M) \in [w']$ (By lemma 8.3)
$\Leftrightarrow a$ receives $r(M) \in Actions(hist_e([w']))$
$\Leftrightarrow a$ receives $r(M) \in Actions(hist_e([w'])|a)$

We thus conclude that $Actions(r(hist_e([w])|a)) = Actions(hist_e([w'])|a)$. But then there is an enumeration e' such that $r(hist_e([w])|a) = hist_{e'}([w'])|a$, and it follows that $hist_e([w]) \longrightarrow_a^r hist_{e'}([w'])$ as desired. The implication from (1) to (2) is immediate from requirement (5) in the definition of *hist*. □

Lemma 8.6 (Filtration Lemma) *Assume* Γ *is closed and* $w, w' \in W_L$.

(i) $w \underset{L}{\longrightarrow_a^r} w' \Rightarrow \exists e' : hist_e([w]) \longrightarrow_a^r hist_{e'}([w'])$

(ii) $hist_e([w]) \longrightarrow_a^r hist_{e'}([w']) \Rightarrow \forall F : \Box_a F \in [w] \Rightarrow r(F) \in [w']$

(iii) $p(M) \in [w] \Leftrightarrow hist_e([w]) \models_{M_{L,\Gamma}} p(M)$, *any predicate* p.

Proof. (3): Immediate from Lemma 8.3 and, in the case of *old*, the fact that e is admissible. (1) and (2) depend on axioms T, 4 and 5 and Reflection Lemma 8.5. For (1), assume $w \underset{L}{\longrightarrow_a^r} w'$. First observe that $r \triangleleft Keys(a,[w])$. By Lemma 8.5 it suffices to show $\Box_a F \in [w]$ iff $\Box_a r(F) \in [w']$. For the only-if direction, if $\Box_a F \in [w]$ then $\Box_a F \in \Gamma \cap w$. By ax. 4, $\Box_a \Box_a F \in w$, since w is maximal consistent. Hence $\Box_a r(F) \in w'$, and so $\Box_a r(F) \in [w']$ as well, since Γ is closed under renamings. The if-direction uses ax. 5 in a similar way, and condition (2) uses T. We omit the details. □

Lemma 8.7 (Truth Lemma for $\mathcal{M}_{L,\Gamma}$) *Assume* Γ *is adequate and* $w \in W_L$. *For all* $F \in \Gamma$*:* $hist_e([w]) \models_{\mathcal{M}_{L,\Gamma}} F \Leftrightarrow F \in [w]$.

Proof. By induction in (the number of statement operators in) F, using Truth Lemma 8.2 and Filtration Lemma 8.6. Base case, atomic statements: From Filtration Lemma 8.6.3. Induction step, negation and conjunction: Since Γ is closed under sub-statements. Induction step, epistemic modality: From Truth Lemma 8.2 and Filtration Lemma 8.6, since Γ is closed under sub-statements

and renamings. □

We continue to show that the transformation $w \curvearrowright hist_e([w])$ preserves truth values from $\mathcal{C}_L$ to $H_{L,\Gamma}$ for statement in Γ.

Lemma 8.8 (Induction) *Assume Γ is adequate. $\mathcal{M}_{L,\Gamma}$ is inductive.*

Proof. By Truth Lemma for $\mathcal{M}_{L,\Gamma}$, as Γ contains *sees*, $K \in \kappa_{L,\Gamma}(a, hist_e([w]))$ iff $hist_e([w]) \models_{\mathcal{M}_{L,\Gamma}} a\ sees\ K$ for all K, w, a. Assume that κ is strictly smaller than $\kappa_{L,\Gamma}$. Then we find some K, w, a such that $K \notin \kappa(a, hist_e([w]))$ and $K \in \kappa_{L,\Gamma}(a, hist_e([w]))$. By the definition of $\kappa_{L,\Gamma}$, $\Box_a a\ rec\ K \in [w]$. It follows that $(a\ \texttt{int}\ a\ rec\ K) \in Actions(hist_e([w])|a)$. We want to show that $hist_e([w]) \models_{\langle H_{L,\Gamma},\kappa\rangle} \Box_a a\ rec\ K$, so that the inductive property fails for κ. To this end assume that $hist_e([w]) \longrightarrow_a^r hist_{e'}([w'])$. Then $r(hist_e([w])|a) = hist_{e'}([w'])|a$, and so $(a\ \texttt{int}\ a\ rec\ r(K)) \in Actions(hist_{e'}([w'])|a)$. It follows that $\Box_a a\ rec\ r(K) \in [w']$, so also $a\ rec\ r(K) \in [w']$, by T, since Γ is closed under sub-statements and w' is maximal. Then $hist_{e'}([w']) \models_{\mathcal{M}_{L,\Gamma}} a\ rec\, r(K)$, by the Truth Lemma 8.7. Since truth of an atomic statement in a message passing model is independent of the key assignment, it follows that $hist_{e'}([w']) \models_{\langle H_{L,\Gamma},\kappa\rangle} a\ rec\, r(K)$ as well. This is sufficient to establish the result, as w' and r were chosen arbitrary. □

Lemma 8.9 (Truth Lemma for $H_{L,\Gamma}$) *Assume Γ is adequate and $w \in W_L$. For all $F \in \Gamma$: $hist_e([w]) \models_{H_{L,\Gamma}} F \Leftrightarrow F \in [w]$.*

Proof. Immediate from Truth Lemma 8.7 for $\mathcal{M}_{L,\Gamma}$ and Induction Lemma 8.8. □

Corollary 8.10 *Assume Γ is adequate. For all $F \in \Gamma$: $\models_{H_{L,\Gamma}} F \Leftrightarrow F \in L$.*

Proof. Assume $F \in \Gamma$. Then $F \in L$, if and only if, $\forall w \in W_L : F \in w$, if and only if (since $F \in \Gamma$), $\forall w \in W_L : F \in [w]$, if and only if (by Truth Lemma 8.9 for $H_{L,\Gamma}$), $\models_{H_{L,\Gamma}} F$. □

Theorem 8.11 (Finite Model Property) *If $\nvdash_{L\boldsymbol{A}} F$, then there is a finite message passing system $H \in \|A\|$ such that $\not\models_H F$.*

Proof. Assume $\nvdash_{L\boldsymbol{A}} F$. Let Γ be the smallest set closed under renamings and sub-statements and containing F and $\boldsymbol{A}$, containing all atomic statements, containing $a\ sees\ M$, $\Box_a a\ received\ M$ and $\Box_a a\ sent\ M$, for all $a \in A$ and messages M. Then Γ is finite, i.e., Γ is adequate. From Corollary 8.10, $\not\models_{H_{L\boldsymbol{A},\Gamma}} F$ and $\models_{H_{L\boldsymbol{A},\Gamma}} \boldsymbol{A}$. By construction, $H_{L\boldsymbol{A},\Gamma}$ is finite, since Γ is finite. □

From Finite Model Property 8.11, we immediately get Completeness Theorem 7.2. By soundness and the proof of completeness it is not difficult to find a bound n such that $F \in L\boldsymbol{A}$, if and only if, F is valid in all systems in $\|A\|$ with at most n histories, each of size less than n. This is sufficient to establish Decidability Theorem 7.3.

9 Conclusion

Several Kripke semantics for BAN have been proposed in the literature. However, no logic faithful to BAN is complete with respect to Kripke semantics, due to the logical omniscience problem. In fact, there have been no completeness results so far for BAN and related logics.

Adopting a recently proposed generalization of Kripke semantics that avoids logical omniscience, we have shown that a logic close to BAN, with full boolean operators, is decidable, and that it is sound and complete with respect to message passing systems. Completeness and decidability generalize to logics induced by an arbitrary protocol specific base. The protocol base may express how participants in a given protocol are expected to behave, or state general assumptions about the network, such as honesty and origination assumptions.

The results assume a finite message space, thus excluding some systems, such as systems in which execution proceeds without an end, with agents constantly generating fresh messages. Also, the language in this paper covers only the symmetric key fragment of BAN.

In the future, we intend to look for effective decision procedures and to extend the completeness result in various directions: To an infinite message space, to asymmetric cryptography, to a description logic extension and to a reformulation of renaming necessitation using message variables, analogously to crypto normality in [5], rather than the quantification over renamings.

References

[1] Abadi, M. and M. Tuttle, *A semantics for a logic of authentication*, in: *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing* (1991), pp. 201–216.

[2] Agray, N., W. van der Hoek and E. P. de Vink, *On ban logics for industrial security protocols.*, in: *CEEMAS*, 2001, pp. 29–36.

[3] Bieber, P., *A logic of communication in hostile environments.*, in: *Proceedings of the Computer Security Foundation Workshop III* (1990), pp. 14–22.

[4] Burrows, M., M. Abadi and R. M. Needham, *A logic of authentication.*, ACM Trans. Comput. Syst. **8** (1990), pp. 18–36.

[5] Cohen, M. and M. Dam, *Logical omniscience in the semantics of BAN logic*, in: *Proceedings of the Foundations of Computer Security Workshop*, 2005, pp. 121–132.

[6] Dekker, A. H., *C3po: A tool for automatic sound cryptographic protocol analysis.*, in: *PCSFW: Proceedings of The 13th Computer Security Foundations Workshop* (2000), pp. 77–87.

[7] Fagin, R., J. Y. Halpern, Y. Moses and M. Y. Vardi, "Reasoning About Knowledge," MIT Press, 1995.

[8] Gong, L., R. M. Needham and R. Yahalom, *Reasoning about belief in cryptographic protocols.*, in: *IEEE Symposium on Security and Privacy* (1990), pp. 234–248.

[9] Lewis, D., *Counterpart theory and quantified modal logic*, Journal of Philosophy **65** (1968), pp. 113–126.

[10] Pucella, R., "Reasoning about Resource-Bounded Knowledge: Theory and Application to Security Protocol Analysis," Ph.D. Thesis, Cornell University, 2004.

[11] Syverson, P. F., *Towards a strand semantics for authentication logics*, in: *Electronic Notes in Theoretical Computer Science*, 20,2000.

[12] Syverson, P. F. and P. C. van Oorschot, *On unifying some cryptographic protocol logics*, in: *Proc. IEEE Symposium on Research in Security and Privacy* (1994), pp. 14–28.

[13] Syverson, P. F. and P. C. van Oorschot, *A unified cryptographic protocol logic*, NRL Publication 5540-227, Naval Research Lab (1996).

[14] Toussaint, M.-J. and P. Wolper, *Reasoning about cryptographic protocols*, in: J. Feigenbaum and M. Merritt, editors, *Distributed Computing and Cryptography*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science **2**, American Mathematical Society, 1989 pp. 245–262.

[15] Wedel, G. and V. Kessler, *Formal semantics for authentication logics*, in: E. Bertino, H. Kurth and G. Martella, editors, *ESORICS'96*, LNCS 1146 (1996), pp. 219–241.

Extensions of Tableau calculi for preference-based conditional logics

Laura Giordano[1], Valentina Gliozzi[2], Nicola Olivetti[3], Camilla Schwind[4]

[1]Diparimento di Informatica, Università del Piemonte Orientale, Alessandria, Italy
[2] Diparimento di Informatica,Università di Torino, Torino, Italy
[3]LSIS-UMR CNRS 6168, Université Paul Cezanne, (Aix-Marseille 3)
[4]MAP, CNRS, Marseille, France

Abstract. In this paper we extend the labelled tableau calculus for conditional logic **CE** proposed in [16]. We consider some extensions of **CE** defined by well-known semantic conditions/ conditional axioms. The proposed methods give a decision procedure for the respective logics and can be used to prove complexity bounds.

1 Introduction

In this paper we focus on extensions of conditional logic **CE** which can be considered as the basic logic of preferential structures [23, 24, 20, 10, 16]. We extend the calculus for **CE** proposed in [16] in order to deal with its most significant extensions. Our aim is to develop *modular* and uniform tableaux systems, where each semantic condition is captured exactly by one tableau rule (or a small group thereof). This work aims to be a step further into the development of proof systems for conditional logics, which are up to now rather scarce, [22, 19, 4, 2, 12, 8, 16, 26, 25] if compared to the development of proof systems for other families of logics, such as modal logics.

Conditional logics have been studied for a long time: they were first introduced by R.Stalnaker and D. Lewis [28, 23, 24] to formalize counterfactual, hypothetical reasoning (if A were the case then B) that cannot be captured by material implication in classical logic. Conditional logics have been used in knowledge representation to reason about prototypical properties [13], to model database update [18], belief revision [3, 14], causal inference in action planning [27, 15] and diagnosis [13]. Moreover, conditional logics are strictly related to non-monotonic reasoning [20], as it turns out that all forms of inferences studied in the framework of non-monotonic logics are particular cases of conditional axioms [6]. The conditional logic **CE** closely corresponds to preferential logic **P** as defined in [20]: the latter coincides with the first-degree fragment of **CE** whereas rational non-monotonic logic corresponds to **CE**+CV, and so on.

Similarly to modal logics, the semantics of conditional logics can be defined in terms of possible world structures. The intuition is that a conditional $A \Rightarrow B$ is true in a world x just in case B is true in the A-worlds that are *most similar/most preferred/closest* to x. This can be expressed by associating to each possible world x a preference relation $<_x$ (an irreflexive and transitive relation) [1]. In preferential struc-

[1] In [16], we took as primitive a pre-order relation $\leq_x$, whose strict version is $<_x$, clearly there is no significant difference.

tures, a conditional $A \Rightarrow B$, is true in x just in case B is true in all worlds satisfying A that are minimal with respect to the relation $<_x$. We assume that *minimal worlds always exist*, thereby accepting the so-called Limit Assumption (or Smoothness Condition as in KLM framework). The logic corresponding to this semantics is exactly the logic **CE** [16]. Stronger logics are obtained by imposing further conditions on the relations $<_x$. The preference semantics is related to the most popular semantics for conditional logics, namely the *sphere semantics* [23] and the *selection function semantics* [24]. The relation is investigated for some systems by Grahne [18].

In this work we concentrate on the extensions of **CE** that are obtained by adding any combination of conditions from the set of well known axioms {**MP**, **CV**, **CS**, **CEM**}. Our tableau systems give a feasible decision procedure for the respective logics and can be used to prove complexity bounds on them. In [16], we have given a terminating sound and complete tableau procedure for **CE** and we proposed some extensions of it. However these extensions were not entirely uniform and no termination analysis was carried out. The present work extends thus our previous results by giving terminating and uniform tableau calculi for the mentioned extensions of **CE** and we provide a non deterministic exponential upper bound for the time complexity of the tableau based decision procedure.

2 The system CE

Let $\mathcal{L}$ be a language containing a set of propositional variables ATM, and the set of all formulas of $\mathcal{L}$ built from ATM by means of the connectives $\neg, \wedge, \bot, \Rightarrow$, where $\Rightarrow$ is the conditional operator.

Definition 1 (Semantics of CE). *A* **CE** *model M has the form $(W, \{<_x\}_{x \in W}, I)$, where W is a non-empty set (of worlds), I is a function $W \rightarrow Pow(ATM)$, and $\{<_x\}_{x \in W}$ is a family of relations on W for each element $x \in W$. For $S \subseteq W$ we define the set of $<_x$-minimal elements of S:*

$$Min_x(S) = \{a \in S \mid \neg \exists b \in S : b <_x a\}.$$

For every $x \in W$: (1) $<_x$ is an irreflexive and transitive relation on W, (2) for every non-empty $S \subseteq W$, $Min_x(S) \neq \emptyset$. We define the truth conditions of formulas wrt. worlds in a model M by the relation $M, x \models \phi$, as follows:

1. $M, x \models p$, *for p atomic, if* $p \in I(x)$,
2. $M, x \not\models \bot$,
3. $M, x \models \neg\phi$ *if* $M, x \not\models \phi$,
4. $M, x \models \phi \wedge \psi$ *if* $M, x \models \phi$ *and* $M, x \models \psi$.
5. $M, x \models \phi \Rightarrow \psi$ *if for all* $y \in Min_x(\phi)$, $M, y \models \psi$, *where* $Min_x(\phi)$ *stands for* $Min_x(\{y \in W \mid M, y \models \phi\})$.

We say that ϕ is valid in M if $M, x \models \phi$ for every $x \in W$.
We say that ϕ is **CE***-valid if it is valid in every* **CE***-model.*

The axiomatization corresponding to the previous semantics is the following axiom system **CE**.

Definition 2 (Axiom system CE). *The system* **CE** *is defined by*

(TAUT) *All classical tautologies and the Modus Ponens rule.*
(ID) $\phi \Rightarrow \phi$
(CA) $(\phi \Rightarrow \chi) \wedge (\psi \Rightarrow \chi) \rightarrow (\phi \vee \psi \Rightarrow \chi)$
(CSO) $(\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi) \rightarrow (\phi \Rightarrow \chi) \leftrightarrow (\psi \Rightarrow \chi)$
(A0) $(\neg\phi \Rightarrow \bot) \rightarrow \phi$
(A1) $(\neg\phi \Rightarrow \bot) \rightarrow \neg(\neg\phi \Rightarrow \bot) \Rightarrow \bot$
(A2) $\neg(\phi \Rightarrow \bot) \rightarrow (\phi \Rightarrow \bot) \Rightarrow \bot$
(RCEA) *if* $\vdash \phi \leftrightarrow \psi$ *then* $\vdash (\phi \Rightarrow \chi) \leftrightarrow (\psi \Rightarrow \chi)$
(RCK) *if* $\vdash (\phi_1 \wedge \ldots \wedge \phi_n) \rightarrow \chi$ *then* $\vdash (\psi \Rightarrow \phi_1 \wedge \ldots \wedge \psi \Rightarrow \phi_n) \rightarrow (\psi \Rightarrow \chi)$

The following theorems can be derived from the axioms of **CE**:

(AC) $(\phi \Rightarrow \chi) \wedge (\phi \Rightarrow \psi) \rightarrow (\phi \wedge \chi \Rightarrow \psi)$
(RT) $(\phi \Rightarrow \chi) \wedge (\phi \wedge \chi \Rightarrow \psi) \rightarrow (\phi \Rightarrow \psi)$

All **CE** axioms and theorems (except (A0),(A1) and (A2)) correspond to well-known properties of nonmonotonic systems: (AC) is called *cumulativity*, (RT) is called *non-monotonic cut* [20]. More precisely, the first-degree fragment of this logic corresponds to preferential logic **P**. Axioms (A0), (A1) and (A2) encode the assumption that each relation $<_x$ ranges on the same set of worlds. This condition is called *Uniformity* in [10].

We can state the following theorems.

Theorem 1 (Soundness). *If a formula ϕ is derivable in* **CE***, then it is* **CE***-valid.*

Theorem 2 (Completeness). *If a formula ϕ is* **CE***-valid, then it is derivable in* **CE***.*

The proof of this last theorem is based on a generalization of Kraus Lehmann and Magidor's completeness proof for the system **P** provided in [20]. Full details of the proof will be provided in a long version of this paper.

2.1 Tableau calculus for CE

In [16], a tableau calculus for **CE** is proposed, which is based on the following ideas:

1 for all w, the fact that $w \in Min_x(\phi)$ can be represented by extending the language $\mathcal{L}$ with a kind of *hybrid* formulas. Given a model $M = (W, \{<_x\}_{x \in W}, I)$, we introduce pseudo-formulas of the form $\Box_x\phi$, for every formula ϕ and world $x \in W$;
2 the semantics of hybrid formulas is the following: $M, y \models \Box_x\phi$ iff $\forall z \in W$ if $z <_x y$, then $M, z \models \phi$ (this can also be expressed by defining for each world x the accessibility relation R_x as the inverse of $<_x$: yR_xz iff $z <_x y$, and then adopting the standard semantics for $\Box_x$: $M, y \models \Box_x\phi$ iff $\forall z \in W$ if yR_xz, then $M, z \models \phi$);
3 from definition of Min_x in 1 above, it follows that that for any formula ϕ, $y \in Min_x(\phi)$ iff $M, y \models \phi \wedge \Box_x\neg\phi$.
4 We use these modal formulas in our tableau calculus for **CE** as illustrated in Figure 1. For instance: if $M, x \models \phi \Rightarrow \psi$, then for all $y \in Min_x(\phi)$, $M, y \models \psi$ (see definition 1 above). By point [3], this can be captured by saying that for all y, if $M, y \models \phi \wedge \Box_x\neg\phi$ then $M, y \models \psi$. This is equivalent to saying that for all y either $M, y \not\models \phi$ or $M, y \not\models \Box_x\neg\phi$ or $M, y \models \psi$. This is exactly what is expressed by rule $(T \Rightarrow)$ in Figure 1.

The tableau rules for **CE** are illustrated in Figure 1. The tableau formulas are of the following kinds: (a) $x : \phi$, where ϕ is a formula or a pseudo-formula; (b) $x <_y z$. A branch is a set of tableau formulas. Given a branch B, we denote by W_B the set of labels occurring in B. A branch is closed if it contains both $x : \phi$ and $x : \neg\phi$, or it contains $x : \bot$, otherwise it is open. A tableau is *closed* if every branch is closed. A formula ϕ is T*-provable* if there exists a closed tableau for $x : \neg\phi$.

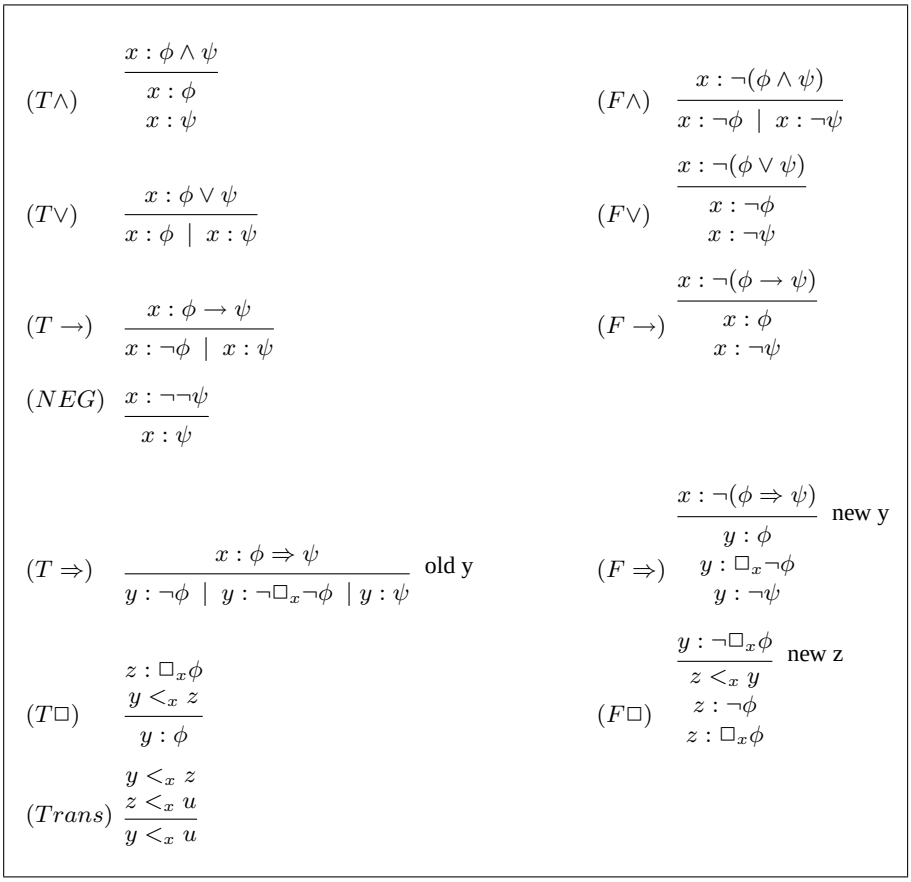


Fig. 1. Tableau rules for **CE**

In [16], the following theorems are proved.

Theorem 3 (Soundness). *If ϕ is T-provable then it is* **CE***-valid.*

Theorem 4 (Completeness). *If a formula ϕ is* **CE***-valid then ϕ is T-provable.*

The tableau calculus for **CE** as given in Figure 1 does not grant termination, since repeated applications of the rules $(F \Rightarrow)$ and $(F\Box)$ may generate infinite labels. In [16]

suitable restrictions on the applications of the rules $(F \Rightarrow)$ and $(F\Box)$ are introduced, which are shown to preserve the completeness of the calculus. Under such restrictions, the calculus is proved to be terminating.

3 Extensions of CE

The tableau method for **CE** can be extended uniformly to deal with some wellknown extensions of **CE** defined by combinations of the following axioms and corresponding semantic properties. We provide a set of extra rules for each extra axiom.

In order to obtain a calculus for conditional logics containing more than one axiom, it will be enough to consider the calculus containing all the rules introduced for the concerned axioms. Thus, for instance, in order to obtain a calculus for the logic obtained by adding to **CE** axioms (CV) and (MP) below, we will have to consider all the rules introduced for **CE** + (CV) and **CE** + (MP). In order to deal with Stalnaker's logic C2, containing **CE** plus axioms (CV), (MP) and (CEM) we will further have to consider the rules introduced to deal with (CEM), and so on.

We consider here the extensions of **CE** by axioms **(MP)**,**(CS)**, **(CEM)**, and **(CV)**:

(MP) $((\phi \Rightarrow \psi) \wedge \phi) \rightarrow \psi$,
its corresponding semantic property is (S-MP):
$\forall x, y \quad \neg(y <_x x)$.

(CS) $\phi \wedge \psi \rightarrow (\phi \Rightarrow \psi)$,
its corresponding semantic property is (S-CS):
$\forall x, y \quad x <_x y \vee x = y$.

(CV) $(\phi \Rightarrow \psi) \wedge \neg(\phi \Rightarrow \neg\chi) \rightarrow (\phi \wedge \chi \Rightarrow \psi)$,
its corresponding semantic property is (S-CV):
$\forall x, y, z, u \quad y <_x z \rightarrow (y <_x u \vee u <_x z)$.

(CEM) $(\phi \Rightarrow \psi) \vee (\phi \Rightarrow \neg\psi)$,
its corresponding semantic property is (S-CEM):
$\forall x, y, z \quad y = z \vee y <_x z \vee z <_x y$.

These axioms/semantic conditions have been variously considered in the literature on conditional logics. We give some intuitive justification. According to (S-MP) no world is preferred to x, with respect to $<_x$.

According to (S-CS), which is stronger than (S-MP), x is preferred to all other possible worlds with respect to its associated relation $<_x$. This condition is controversial. It is for instance the condition that differentiates Lewis's systems VC and VW [23], namely (S-CS) belongs to the first system whereas it does not belong to the second one.

The condition (S-CV) makes $<_x$ a *modular* (or ranked) relation. This condition is required by most of the conditional logics based on preference semantics (or the related sphere based semantics), with the exception of **CE** . It is contained, for instance in Lewis's logics VC and VW [23].

The property (S-CEM) requires that given two different worlds y and z, either y is strictly preferred to z or z is strictly preferred to y w.r.t. $<_x$. In turn, this entails that for each formula ϕ, $Min_<(\phi)$ contains at most one single possible world. This property

is the well-known Uniqueness Assumption that characterizes Stalnaker's logic $C2$ [28, 24].

Let $\Sigma \subseteq \{$**(S-MP)**, **(S-CS)**, **(S-CV)**, **(S-CEM)**$\}$, to ease the reading, we sometime say that a model M is a **(S-Σ)**-*model* if it satisfies the semantic conditions in Σ.

In order to provide the calculi for the extensions of **CE**, we have to extend the language of the tableau calculus by allowing equalities of the form $x = y$ in branches, where x, y are labels. This extension is needed for systems containing (CS) or (CEM). Consequently, we enrich the notion of closed branch as follows.

Definition 3 (Closed Branch). *We say that a branch is closed if it contains: (i) $x : \phi$ and $x : \neg\phi$, or $x :\bot$; (ii) $x = y$, $x : \phi$, $y : \neg\phi$; (iii) $y <_x y$. A tableau is closed if every branch is closed.*

In order to deal with equality we need the following rules:

$$(\mathbf{E-R})\ \frac{}{x = x}\ (*) \qquad (\mathbf{E-S})\ \frac{x = y}{y = x}$$

$$(\mathbf{E-T})\ \frac{x = y, y = z}{x = z} \qquad (\mathbf{E}{-}{<})\ \frac{x = x', y = y', z = z', y <_x z}{y' <_{x'} z'}$$

Fig. 2. Rules for Equality. (*) x is a label in the branch.

The rules corresponding to the mentioned extensions of **CE** are the following ones:

$$(\mathbf{R-MP})\ \frac{y <_x x}{x :\bot} \qquad (\mathbf{R-CS})\ \frac{}{x <_x y \mid x = y}\ old\ x, y$$

$$(\mathbf{R-CV})\ \frac{y <_x z}{y <_x u \mid u <_x z}\ old\ u \qquad (\mathbf{R-CEM})\ \frac{}{y = z \mid y <_x z \mid z <_x y}\ old\ x, y, z$$

Fig. 3. Rules for the extensions of **CE**.

Definition 4. *Let $\Sigma \subseteq \{$**(R-MP)**, **(R-CS)**, **(R-CV)**, **(R-CEM)**$\}$ a Σ-tableau is a tableau built according to: (i) the rules for* **CE** *of Figure 1, (ii) the rules in Σ of Figure 3, (iii) the rules for Equality in case* **(R-CS)** *or* **(R-CEM)** *belongs to Σ.*
A formula ϕ is **T + (**Σ**)**-*provable if there exists a closed Σ-tableau tableau for $x : \neg\phi$.*

The soundness of the rules can be proved in a standard way (see Theorem 2 in [16]). To this purpose, we need to give a formal meaning to the notion of satisfiability of a

branch. Let $M = (W, \{\leq_x\}_{x \in W}, I)$ be a model. Given a branch B, a mapping is a function f from W_B to W such that (i) if $y <_x z \in B$ then $f(y) <_{f(x)} f(z)$, (ii) if $x = y \in B$ then $f(x) = f(y)$.

Definition 5. *Let $\Sigma \subseteq \{$**(R-MP)**, **(R-CS)**, **(R-CV)**, **(R-CEM)**$\}$, given a branch B of a Σ-tableau, a* **(S-Σ)***-model M, and a mapping f from W_B to W, we say that B is satisfiable under f in M if the following holds:*

1. *if $x : \phi \in B$ then $M, f(x) \models \phi$,*
2. *if $x : \neg\phi \in B$ then $M, f(x) \not\models \phi$.*

where ϕ is a conditional formula or a pseudo formula.

Moreover, B is **(S-Σ)**-satisfiable *if it is satisfiable in some* **(S-Σ)***-model M under some mapping f. A tableaux is* **(S-Σ)***-satisfiable if one of its branches is* **(S-Σ)***-satisfiable.*

One can prove the following lemma[2].

Lemma 1. *Let $\Sigma \subseteq \{$**(R-MP)**, **(R-CS)**, **(R-CV)**, **(R-CEM)**$\}$, let T be a* **(S-Σ)***-satisfiable tableau and let T' be obtained from T by applying one of the rules of Σ-tableaux. Then T' is also* **(S-Σ)***-satisfiable.*

Theorem 5 (Soundness of $T - \Sigma$). *If ϕ is* **T + (Σ)***-provable, then it is valid in all* **(S-Σ)***-models.*

Proof. Suppose that ϕ is not valid, then $\neg\phi$ is satisfiable in a **(S-Σ)**-model M, thus the tableau beginning with $x : \neg\phi$ is **(S-Σ)**-satisfiable. By the previous lemma any expansion of the tableaux contains a **(S-Σ)**-satisfiable branch B; B cannot be closed, for otherwise we would have $z : \psi, z : \neg\psi \in B$, whence $M, f(z) \models \psi$ and $M, f(z) \models \neg\psi$, a contradiction. □

3.1 Completeness of the calculi

First of all, notice that since none of the rules above introduce in the tableau new labels or new formulas that create new labels when decomposed, from Lemma 3, Section 4.2 of [16], we have the following lemma.

Lemma 2. *Let B be a branch of a tableau. B does not contain any infinite descending chain of labels s.t. $y_1 <_x y_0, y_2 <_x y_1, \ldots, y_{i+1} <_x y_i, \ldots$.*

We can now show the completeness of the calculi. To this purpose, we have to enrich the notion of saturated set used in the proof of the completeness of the calculus in Definition 5 in Section 4, of [16].

Definition 6 (Saturated Branch). *We say that a branch B of a tableau is:*

– saturated *w.r.t. T if:*
  1. *If $x : \phi \wedge \psi \in B$ then $x : \phi \in B$ and $x : \psi \in B$.*
  2. *If $x : \neg(\phi \wedge \psi) \in B$ then either $x : \neg\phi \in B$ or $x : \neg\psi \in B$.*

[2] The proof will be included in a full paper.

3. *If* $x : \neg\neg\phi \in B$ *then* $x : \phi \in B$.
4. *If* $x : (\phi \Rightarrow \psi) \in B$ *then for any label* $y \in W_B$, *either* $y : \neg\phi \in B$ *or* $y : \neg\Box_x\neg\phi \in B$ *or* $y : \psi \in B$.
5. *If* $x : \neg(\phi \Rightarrow \psi) \in B$ *then there is a label* y *such that* $y : \phi \in B$ *and* $y : \neg\psi \in B$ *and* $y : \Box_x\neg\phi \in B$.
6. *If* $y : \Box_x\phi \in B$ *and* $z <_x y \in B$ *then* $z : \phi \in B$.
7. *If* $y : \neg\Box_x\phi \in B$ *then there is a label* z *such that* $z <_x y \in B$ *and* $z : \neg\phi \in B$ *and* $z : \Box_x\phi \in B$.
8. *If* $y <_x z \in B$ *and* $z <_x u \in B$ *then* $y <_x u \in B$.

– saturated *w.r.t.* $=$ *if:*
1. *for all* $x \in B$, $x = x \in B$;
2. *if* $x = y \in B$, *also* $y = x \in B$;
3. *if* $x = y \in B$ *and* $y = z \in B$, *also* $x = z \in B$;
4. *if* $x = x' \in B$, $y = y' \in B$, $z = z' \in B$, *and* $y <_x z \in B$, *then also* $y' <_{x'} z' \in B$;

– saturated *with respect to* **CE + (MP)** *if it is saturated w.r.t. to* T, *and for no* x, y $y <_x x \in B$;
– saturated *with respect to* **CE + (CS)** *if it is saturated w.r.t. to* T, *to* $=$, *and for all* $x, y \in B$, *either* $x <_x y \in B$ *or* $x = y \in B$;
– saturated *with respect to* **CE + (CV)** *if it is saturated w.r.t. to* T, *and for all* $x, y, z, u \in B$, *if* $y <_x z \in B$, *then also* $y <_x u \in B$ *or* $u <_x z \in B$;
– saturated *with respect to* **CE + (CEM)** *if it is saturated w.r.t. to* T, *to* $=$, *and for all* $x, y, z \in B$, *either* $y = z \in B$ *or* $y <_x z \in B$ *or* $z <_x y \in B$.

Given a set of axioms $\mathbf{\Sigma} \subseteq \{$**(MP) (CS)(CV)(CEM)**$\}$, we can now show the following lemma and completeness theorem.

Lemma 3. *If* ϕ *is not* **T + (**Σ**)-** *provable, then there exists a tableau which contains an open branch starting with* $x : \neg\phi$, *and which is saturated with respect to the rules corresponding to the axioms contained in* Σ.

Proof.(Sketch) Since ϕ is not provable, all tableaux starting with $\neg\phi$ will contain an open branch. Thus also the tableau built according to the following systematic procedure contains an open branch: **(step a:)** apply all the static rules as far as possible; in case of branching, make the choice which preserves non-closure (there is always one choice, by hypothesis); **(step b:)** apply the dynamic rules to the new formulas generated in the previous step.

By this strategy, every formula in the branch is eventually considered. It can be easily shown that the branch generated by this systematic procedure is saturated with respect to the rules corresponding to the axioms contained in Σ. □

Theorem 6 (Completeness of $T - \Sigma$**).** *If* ϕ *is valid in all* **(S-**Σ**)-***models, then it is* **T + (**Σ**)-***provable.*

Proof.(Sketch) We show the contrapositive: that if ϕ is not **T + (**Σ**)-**provable, then $\neg\phi$ is satisfiable by a **CE** model satisfying **(S-** Σ**)**. If ϕ is not provable, then the tableau starting with $x : \neg\phi$ contains an open branch B, saturated with respect to the rules corresponding to the axioms contained in Σ. Starting from this open, saturated branch,

we build a canonical model. We have two alternative constructions, one for the case in which neither axiom **(CS)** nor axiom **(CEM)** belong to Σ, and another corresponding to the case in which (at least) one of the two axioms belongs to Σ (and therefore the symbol $=$ appears in B).
Case 1: (CS) $\notin \mathbf{\Sigma}$ and **(CEM)** $\notin \mathbf{\Sigma}$. We can build the canonical model M_C satisfying $\neg\phi$, as follows. $M_C = \langle W, \{\sqsubset_x\}_{x\in W}, I\rangle$, where: 1) $W = W_B$; 2) For each $x \in W$, $y \sqsubset_x z$ iff $y <_x z \in B$; 3) For each $x \in W$, $I(x) = \{p \mid p \in ATM$ and $w : p \in B\}$.
We prove that:

1 $\sqsubset_x$ is an irreflexive and transitive relation on W.This follows from definitions 3 and 6 above;
2 For all $S \subseteq W$, $S \neq \emptyset$, $Min_x(S) \neq \emptyset$. This follows from Lemma 2 above.
3 By induction over the complexity of the formulas, we can show that M_C satisfies all formulas and pseudo-formulas in B. As an example, we consider $x : \phi \Rightarrow \psi \in B$. Let $y \in W_B$. Then either (i) $y : \neg\phi \in B$ or (ii) $y : \neg\Box_x\neg\phi \in B$, or (iii) $y : \psi \in B$. If (i) by the induction hypothesis, we have $M_C, y \models \neg\phi$, thus $y \notin Min_x(\phi)$. If (ii) there is a label z such that $z <_x y \in B$ and $z : \phi \in B$. By construction of M_C, we have $z \sqsubset_x y$ and by the induction hypothesis $M_C, z \models \phi$, thus $y \notin Min_x(\phi)$. If (iii) by the induction hypothesis we get $M_C, y \models \psi$. Thus, if $y \in Min_x(\phi)$ then $M_C, y \models \psi$.
4 If **(MP)** $\in \mathbf{\Sigma}$, then M_C satisfies **(S-MP)**. For a contradiction, suppose $y \sqsubset_x x$. By definition of $\sqsubset_x$, $y <_x x \in B$. But this contradicts definition 6 above.
5 If **(CV)** $\in \mathbf{\Sigma}$, then M_C satisfies **(S-CV)**. Let $y \sqsubset_x z$. By definition of M_C, $y <_x z \in B$, and by definition 6, for all $u \in W_B$, either $y <_x u$ or $u <_x z$. In the first case, $y \sqsubset_x u$; in the second case $u \sqsubset_x z$.

Case 2: (CS) $\in \mathbf{\Sigma}$ or **(CEM)** $\in \mathbf{\Sigma}$. We build the canonical model as follows: for all $x, y \in W_B$, we let $x =_M y$ iff $x = y \in B$. Notice that $=_M$ is an equivalence relation. Indeed: it is symmetric (all the rules that introduce $x = y$ also introduce $y = x$); it is transitive, by definition 6; it is reflexive by symmetry + transitivity.
We let $[x] = \{y : x =_M y\}$ and $W = \{[w]/ =_M\}$, the set of all the equivalence classes of W_B with respect to $=_M$.
We define $I([x]) = \bigcup_{y\in[x]}\{p : p \in ATM$ and $y : p \in B\}$.
Furthermore, we let $[y] \sqsubset_{[x]} [z]$ iff $y' <_{x'} z' \in B$ for $x' = x$, $y' = y$, and $z' = z$. We show that M_C is a **CE** model of B satisfying **(S-CS)**.

1 $\sqsubset_{[x]}$ is irreflexive and transitive. Irreflexive, by 3 and by definition of $\sqsubset_{[x]}$, and transitive by 6.
2 if $S \subseteq W$ and $S \neq \emptyset$, then $Min_x(S) \neq \emptyset$. This follows from Lemma 2.
3 Reasoning by induction on the complexity of the formulas, we show that for all formula or pseudo-formula F (a) if $x : F \in B$, then $M_C, [x] \models F$, and (b) if $x : \neg F \in B$, then $M_C, [x] \not\models F$:
   - for $F \in ATM$: (a) follows from definition of I, and (b) follows from definition of I together with the fact that there is no x' s.t. $x' = x \in B$, and $x : F \in B$ (otherwise B would be closed);
   - if F is a boolean combination of formulas, both (a) and (b) can be easily shown;

- if F is a pseudo-formula $\Box_y\phi$. (a): for all $[z] \sqsubset_{[y]} [x]$, by definition of M_C and by definition 6 above, $z <_y x \in B$. Hence, $z : \phi \in B$, and by inductive hypothesis, $M_C, [z] \models \phi$. (b): for some z, $z <_y x \in B$, $z : \neg\phi$. By definition of M_C, $[z] \sqsubset_{[y]} [x]$, and by inductive hypothesis, $M_C, [z] \not\models \phi$;
- if F is $\phi \Rightarrow \psi$. (a): for all $y \in W_B$, either (i)$y : \neg\phi \in B$ or (ii)$y : \neg\Box_x\neg\phi \in B$ or (iii)$y : \psi \in B$. In all cases, the result follows by inductive hypothesis. (b): there is a z s.t. $z : \phi \in B$, $z : \Box_x\neg\phi \in B$, and $z : \neg\psi \in B$. By inductive hypothesis, $M_C, [z] \models \phi$, $M_C, [z] \models \Box_x\neg\phi$, $M_C, [z] \not\models \psi$, and the result follows.

4 If **(CS)** $\in \mathbf{\Sigma}$, then M_C satisfies **(S-CS)**. Indeed, by definition 6, either $x = y \in B$ (and $[x] = [y]$) or $x <_x y \in B$, and $[x] <_{[x]} [y]$.

5 If **(CEM)** $\in \mathbf{\Sigma}$, then M_C satisfies **(S-CEM)**. Indeed, by definition 6, for all $[x], [y], [z]$, either $y = z \in B$, and by construction of M_C $[y] = [z]$, or $y <_x z \in B$, and by construction of M_C, $[y] <_{[x]} [z]$ or $z <_x y \in B$, and by construction of M_C, $[z] <_{[x]} [y]$.

6 If **(MP)** $\in \mathbf{\Sigma}$, then M_C satisfies **(S-MP)**. We reason analogously to Case 1.

7 If **(CV)** $\in \mathbf{\Sigma}$, then M_C satisfies **(S-CV)**. We reason as for Case 1. □

3.2 Termination and Complexity of the Calculi

As in the case of logic **CE**, the calculi presented above for **CE** extensions can lead to non-terminating computations due to the interplay among the rules $(F \Rightarrow)$ and $(F\Box)$, generating new labels, and rule $(T \Rightarrow)$, whose applications to the new labels may generate new formulas to which $(F \Rightarrow)$ or $(F\Box)$ can again be applied. In this section we show that the calculi for **CE** extensions can be made terminating. The solution we propose is similar to the one we have adopted to prove the termination of **CE** in [16]. More precisely, in order to avoid the generation of infinite branches, we introduce a systematic procedure to build a tableau and we put suitable restrictions on the applications of rules $(F \Rightarrow)$ and $(F\Box)$. We show that the systematic procedure terminates, whence it does not lead to generate infinitely many labels. Moreover, we show that the completeness of the calculi is not lost if we adopt the systematic procedure with the mentioned restrictions.

First of all let us assume that the tableau construction does not allow any redundant application of the rules ([16]). In particular, we assume that the $(T \Rightarrow)$ applies exactly once to each formula and label on the branch. Also, we assume that a branch does not contain repetitions of labelled formulas: labelled formulas which are already on the branch are not added again when applying new rules.

Following the terminology of [17], we refer to the rules generating new worlds as *dynamic rules*, and to all other rules as *static rules*. Observe that all the rules we have introduced are static except for $(F \Rightarrow)$ and $(F\Box)$ which are dynamic.

We define a *systematic procedure* for constructing the tableau for a given formula $x : \alpha$. This is the same procedure described in the proof of Lemma 3. The procedure executes repeatedly two steps: **(step a)** applies all the static rules as far as possible; **(step b)** applies the rules $(F\Box)$ and $(F \Rightarrow)$ to the new formulas generated in the previous step.

We observe that (step a) terminates after a finite number of rule applications. Indeed, none of the static rules adds new labels and the number of pseudo-formulas built from a

finite set of formulas (subformula property) and from a finite number of labels is finite. As we require that the tableau does not contain redundant applications of rules, each application of a rule in step (a) must at least add a new pseudo-formula to the branch, and thus the number of rule applications must be finite. After (step a) is terminated the branch is downward saturated except for the formulas of the form $w : \neg(\phi \Rightarrow \psi)$ and $w : \neg\Box_x\phi$. In (step b) we apply the rule $(F \Rightarrow)$ to all the formulas $y : \neg(\gamma \Rightarrow \delta)$ on the branch and rule $(F\Box)$ to all the formulas $y : \neg\Box_x\gamma$ on the branch, where y must be a label generated in the previous (step a).

To avoid that (step a) and (step b) in the systematic procedure repeat forever, by continuing to generate new worlds, we put the following restrictions on the application of the rules $(F \Rightarrow)$ and $(F\Box)$, which are essentially loop checking conditions.

- **(Restriction 1)** the rule $(F \Rightarrow)$ can be applied to the formula $y : \neg(\gamma \Rightarrow \delta)$ on a branch only if there is no label z on the branch such that:
  - (1a) the branch contains the formula $z : \neg(\gamma \Rightarrow \delta)$ and the rule $(F \Rightarrow)$ has already been applied to that formula;
  - (1b) the positive conditional formulas labelled by y are a subset of the positive conditional formulas labelled by z.
- **(Restriction 2)** the rule $(F\Box)$ can be applied to the formula $y : \neg\Box_x\gamma$ on a branch only if we cannot find two labels z and w on the branch such that:
  - (2a) the branch contains a formula $z : \neg\Box_w\gamma$ and the rule $(F\Box)$ has already been applied to that formula;
  - (2b) the positive conditional formulas labelled by x on the branch are a subset of the positive conditional formulas labelled by w;
  - (2c) for each (positive) modal pseudo-formula $y_1 : \Box_x\phi$ occurring on the branch, such that either y_1 coincides with label y or one of the two relations $y <_x y_1$ and $y_1 = y$ occurs on the branch, there is a (positive) modal pseudo-formula $z_1 : \Box_w\phi$ on the branch, such that either z_1 coincides with label z or one of the two relations $z <_w z_1$ and $z_1 = z$ occurs on the branch.

The idea behind (Restriction 1) is that applying the rule $(F \Rightarrow)$ to $y : \neg(\gamma \Rightarrow \delta)$ cannot add anything more on the branch than what has been obtained by applying it to $z : \neg(\gamma \Rightarrow \delta)$, if all the positive conditional formulas that hold at y also hold at z. In such a case we can avoid to generate a new label y' from y. The idea behind (Restriction 2) is that applying the rule $(F\Box)$ to $y : \neg\Box_x\gamma$ cannot add anything more on the branch than what has been obtained by applying it to $z : \neg\Box_w\gamma$, under the assumptions that: all the positive conditional formulas that hold at x also hold at w; and all the positive modal pseudo-formulas on the branch that would be applicable to the world y' generated from y, can also be applied to the world z' generated from z.

The systematic procedure with the two restrictions preserves completeness and guarantees termination.

Lemma 4. *If a branch B closes, it still closes under (Restriction 1) and (Restriction 2).*

Theorem 7. *The systematic procedure with (Restriction 1) and (Restriction 2) terminates.*

The proofs of Lemma 4 and of Theorem 7 are similar to those reported in [16] for logic **CE**. The intuition is the following one. If a systematic attempt at proving a formula goes forever, there must be an infinite branch S containing infinitely many different labels because it does not contain repetitions and only a finite number of formulas can appear in the tableau (subformula property). Therefore, the branch must either contain infinitely many applications of rule $(F \Rightarrow)$ or infinitely many applications of rule $(F\Box)$ to some formula (or both). By (Restrictions 1) and (Restriction 2) above, this is not possible. For instance, each time the rule $(F \Rightarrow)$ is applied to $y : \neg(\phi \Rightarrow \psi)$, by (Restriction 1) for all possible $z : \neg(\phi \Rightarrow \psi)$ on the branch to which $(F \Rightarrow)$ has already been applied there must be some positive conditional formula occurring on the branch labelled by y and not by z. This can only occur a finite number of times, since (by the subformula property) there is only a finite number of positive conditionals.

Observe that the number of positive conditionals is finite, and at most linear in the size n of the input formula. Hence, the number of distinct sets of positive conditionals is $O(2^n)$. Therefore, the number of applications of rule $(F \Rightarrow)$ to the same formula (but different labels) on a branch is $O(2^n)$. As there are $O(n)$ distinct positive conditionals, the number of applications of rule $(F \Rightarrow)$ on a branch is $O(2^n)$. By similar considerations, we can conclude that the number of applications of rule $(F\Box)$ on a branch is $O(2^n)$. We can thus say that the number of different labels on the branch is $O(2^n)$.

In order to establish an upper bound on the complexity of the systematic procedure above, we need to estimate the overall number of rule applications, including those in step (a). To this concern, we can observe that the number of pseudo-formulas (including equalities and disequalities) which can be built from $O(n)$ formulas and $O(2^n)$ labels is $O(2^n)$. As we do not admit redundant applications of rules, on each branch the overall number of tableau rule applications determined by the systematic procedure is $O(2^n)$. Thus, our systematic procedure allows us to define a non-deterministic algorithm for testing satisfiability in **CE**: at each step the algorithm first saturates with respect to the static rules (step a), then guesses a branch and saturates it w.r.t. the dynamic rules (step b). By what said just above, the algorithm eventually terminates, and returns "satisfiable" if it finds an open branch and "unsatisfiable" if it does not. Since the number of rules applied is $O(2^n)$, we can conclude that:

Theorem 8. ***Theorem 9.*** *For all extensions* **CE**+Σ *of* **CE**, *the systematic procedures decides the satisfiability of a formula in non-deterministic exponential time with respect to the size of the input.*

The above result seems to be in partial accordance with [10], where the authors show that a formula is satisfiable iff it is satisfiable in an exponentially sized structure. Moreover, the authors argue that deciding satisfiability can be done in EXPTIME (that we understand as Deterministic Exponential Time), even if the algorithm they suggest is not described in full details (Theorem 0.13). In future investigation we will study how we can obtain a *deterministic* EXPTIME decision procedure based on our tableau calculus.

4 Conclusions

In this work we have presented tableaux systems for the most important extensions of **CE**, the logic of preferential structures. The proof methods are uniform in the sense

that each specific semantic condition is captured by a tableau rule. Moreover, the proof method is based on the introduction of pseudo-formulas, that are modalities in a hybrid language indexed on worlds. We have been able to obtain terminating procedures by a loop-checking mechanism. These methods can also be used to establish complexity bounds of the logics, in a constructive way.

In future work, we intend to carry on a finer grained complexity analysis of **CE** extensions by means of tableaux methods. Moreover, we intend to investigate several improvements on the tableau systems, including loop-checking mechanisms, a better treatment of equality via label-substitution and specific proof-search strategies.

We briefly remark on some related works. De Swart [8] and Gent [12] give sequent/tableaux calculi for conditional logics VC (= **CE**+CV+MP+CS) and VCS. The kind of systems they propose are based on the entrenchment connective $\leq$, from which the conditional operator can be defined. Their systems are analytic and comprise an infinite set of rules $\leq F(n, m)$, with a uniform pattern, to decompose each sequent with m negative and n positive entrenchment formulas.

Crocco and Farinas [4] present sequent calculi for some conditional logics including minmal CK, CEM, CO (= **CE** without CA) and others. Their calculi comprise two levels of sequents: principal sequents with $\vdash_P$ corresponds to the basic deduction relation, whereas auxiliary sequents with $\vdash_a$ corresponds to the conditional operator: thus the constituents of $\Gamma \vdash_P \Delta$ are sequents of the form $X \vdash_a Y$, where X, Y are sets of formulas.

Artosi, Governatori, and Rotolo [2] develop labelled tableau for the *first-degree* fragment (i.e. without nested conditionals) of conditional logic CO (they call it CU and it corresponds to cumulative non-monotonic logics). Formulas are labelled by path of worlds containing also variable worlds. Since they adopt a selection function semantics, they have to cope with the problem of equivalent antecedents: i.e. if $[A]^M = [A']^M$ then $f(A, w) = f(A', w)$. They use an efficient unification procedure to propagate positive conditionals, and the unification procedure takes care of checking the equivalence of antecedents. Their tableau system is not analytical, as it contains a cut-rule, called PB, which is not eliminable. Moreover it is not clear how to extend it to **CE** and stronger systems on the one hand, and to nested conditionals, on the other.

Broda, Gabbay, Lamb, and Russo develop an elegant natural deduction system for Boutillier's conditional logic of normality and some variants of it [3], [1]. Their proof system uses labels following the methodology of *Labelled Deductive Systems* [1], where the objects involved in the proofs are structured configurations of formulas, worlds, and relations thereof. In this respect their approach is rather similar to ours. However, as we already observed, Boutilier's conditional logic has a simpler semantics defined in terms of standard modal logic without world-indexed relations or modalities (and thus it cannot handle nested and iterated conditionals). Moreover, it is not evident if one can extract a decision procedure for Boutilier's logic from their natural deduction system.

Lamarre [22] presents tableaux systems for conditional logics V(= **CE**+CV), VN, VC and VW(= **CE**+CV+MP). Lamarre's method is a consistency-checking procedure which tries to build a system of sphere falsifying the input formulas. The method makes use of a subroutine to compute the *core*, that is defined as the set of formulas characterizing the minimal sphere. The computation of the core needs in turn the consistency

checking procedure. Thus there is a mutual recursive definition between the procedure for checking consistency and the one to compute the core.

Gronebner and Delgrande [19] have developed a tableau method for conditional logic VN which is based on the translation of this logic into modal logic S4.3.

In [26], it is presented a labelled sequent calculus minimal normal conditional logic **CK** and some extensions of it. The calculi are based on the selection function semantics for these logics. In case of **CK**, the calculus is used to provide a polynomial-space complexity bound for this logic.

References

1. K. Broda, D. Gabbay, L.Lamb and A. Russo. Labelled Natural Deduction for Conditional Logic of Normality. *Logic Journal of the IGPL*, Vol 10(2), 2002, 123-163, Oxford University Press.
2. A. Artosi, G. Governatori, and A. Rotolo. Labelled tableaux for non-monotonic reasoning: Cumulative consequence relations. *Journal of Logic and Computation*, 12, 2002.
3. C. Boutilier, Conditional logics of normality: a modal approach. *Artificial Intelligence*, 68:87–154, 1994.
4. G. Crocco and L. Fariñas del Cerro, *Structure, Consequence relation and Logic*, volume 4, pages 239–259. Oxford University Press, 1992.
5. G. Crocco, L. Farias del Cerro, and A. Herzig, *Conditionals: From philosophy to computer science*, Oxford University Press, Studies in Logic and Computation, 1995.
6. G. Crocco and P. Lamarre, *On the Connection between Non-Monotonic Inference Systems and Conditional Logics*, In B. Nebel and E. Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 3rd International Conference*, pages 565-571, 1992.
7. J.P. Delgrande, A first-order conditional logic for prototypical properties. *Artificial Intelligence*, (33):105–130, 1987.
8. H.C.M de Swart, A Gentzen-or Beth-type system, a practical decision procedure and a constructive completeness proof for the counterfactual logics VC and VCS, *Journal of Symbolic Logic*, 48:1-20, 1983.
9. M. Fitting, *Proof methods for Modal and Intuitionistic Logic*, vol 169 of Synthese library, D. Reidel, Dordrecht, 1983.
10. N. Friedman and J. Halpern, On the complexity of conditional logics. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 4th International Conference, KR'94*, pages 202–213.
11. D. M. Gabbay, *Labelled Deductive Systems* (vol I), Oxford Logic Guides, Oxford University Press, 1996.
12. I. P. Gent, A sequent or tableaux-style system for Lewis's counterfactual logic VC. *Notre Dame j. of Formal Logic*, 33(3): 369-382, 1992.
13. M. L. Ginsberg. Counterfactuals. *Artificial Intelligence*, 30(2):35–79, 1986.
14. L. Giordano, V. Gliozzi, and N. Olivetti. Iterated Belief Revision and Conditional Logic. *Studia Logica*, special issue on Belief Revision, Vol.70, No. 1, pp.23-47, 2002.
15. L. Giordano and Camilla Schwind. Conditional logic of actions and causation. *Artif. Intell.*, Vol.157, No. 1-2, pp.239-279, 2004.
16. L. Giordano and V. Gliozzi and N. Olivetti and C. Schwind, Tableau Calculi for Preference-Based Conditional Logics, In Proc. of TABLEAUX 2003, vulme 2796 of LNAI, Springer, pp.81-101, 2003.

17. R. Goré.Tableau Methods for Modal and Temporal Logics Rajeev Goré in *Handbook of Tableau Methods*, M D'Agostino, D Gabbay, R Haehnle, J Posegga (Eds.) Kluwer Academic Publishers, pages 297-396, 1999.
18. G. Grahne, Updates and Counterfactuals, *in* Journal of Logic and Computation, Vol 8 No.1:87-117, 1998.
19. C. Groeneboer and James Delgrande, A general approach for determining the validity of commonsense assertions using conditional logics. *International Journal of Intelligent Systems*, (5):505–520, 1990.
20. S. Kraus, D. Lehmann, and M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence*, 44:167–202, 1990.
21. P. Lamarre. Etude des raisonnements non-monotones: apports des logiques des conditionnels et des logiques modales. PhD thesis, universit Paul Sabatier, Toulouse 1992.
22. P. Lamarre. A tableaux prover for conditional logics. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 4th International Conference, KR'94*, pages 572–580.
23. D. Lewis, *Counterfactuals*. Basil Blackwell Ltd, 1973.
24. D. Nute, *Topics in Conditional Logic*, Reidel, Dordrecht, 1980.
25. N. Olivetti and G. Pozzato. CondLean: A Theorem Prover for Conditional Logics. In Proc. of TABLEAUX 2003 (Automated Reasoning with Analytic Tableaux and Related Methods), volume 2796 of LNAI, pages 264-270. Springer, 2003.
26. N. Olivetti and C. Schwind, A sequent calculus and a complexity bound for minimal conditional logic. In *Proc. ICTCS01 - Italian Conference on Theoretical Computer Science*, vol LNCS 2202, pages 384-404, 2001.
27. C. B. Schwind, Causality in Action Theories. Electronic Articles in Computer and Information Science, Vol. 3 (1999), section A, pp. 27-50.
28. R. Stalnaker, A Theory of Conditional, *in* N. Rescher (ed.), Studies in Logical Theory, American Philosophical Quarterly, Monograph Series no.2, Blackwell, Oxford: 98-112, 1968.
29. L. Viganò, *Labelled Non-classical Logics*. Kluwer Academic Publishers, Dordrecht, 2000.

KLMLean 1.0: a Theorem Prover for Logics of Default Reasoning

Nicola Olivetti* and Gian Luca Pozzato**

Abstract. In this paper we present KLMLean 1.0, a theorem prover for some logics of default reasoning, namely Preferential logic **P** and Loop-Cumulative logic **CL** introduced by Kraus, Lehmann, and Magidor. KLMLean 1.0 implements some tableaux calculi for these logics recently introduced. It is implemented in SICStus Prolog and also comprises a graphical user interface written in Java. KLMLean 1.0 is available for free download at `http://www.di.unito.it/~pozzato/klmlean1/`

1 Introduction

In the early 90' [6, 7] Kraus, Lehmann and Magidor (from now on KLM) proposed a formalization of non-monotonic reasoning that was early recognized as a landmark. Their work stemmed from two sources: the theory of nonmonotonic consequence relations initiated by Gabbay [4] and the preferential semantics proposed by Shoham [11] as a generalization of Circumscription. Their works lead to a classification of nonmonotonic consequence relations, determining a hierarchy of stronger and stronger systems.

According to the KLM framework, defeasible knowledge is represented by a (finite) set of nonmonotonic conditionals or assertions of the form $A \mathrel{|\!\sim} B$ whose reading is *normally (or typically) the A's are B's*. The operator "$\mathrel{|\!\sim}$" is nonmonotonic, in the sense that $A \mathrel{|\!\sim} B$ does not imply $A \wedge C \mathrel{|\!\sim} B$. For instance, a knowledge base K may contain the following set of conditionals: $adult \mathrel{|\!\sim} work, retired \mathrel{|\!\sim} adult, retired \mathrel{|\!\sim} \neg work$, whose meaning is that adults typically work, retired people are typically adults but they typically do not work. Observe that if $\mathrel{|\!\sim}$ were interpreted as classical (or intuitionistic) implication, we simply would get $retired \mathrel{|\!\sim} \bot$, i.e. typically there are not retired people, thereby obtaining a trivial knowledge base. One can derive new conditional assertions from the knowledge base by means of a set of inference rules.

In KLM framework, the set of adopted inference rules defines some fundamental types of inference systems, namely, from the weakest to the strongest: Cumulative (**C**) , Loop-Cumulative (**CL**), Preferential (**P**) and Rational logic (**R**). All these systems allow one to infer new assertions from K without incurring in the trivialising conclusions of classical logic: regarding our example, in

* LSIS - UMR CNRS 6168 Université Paul Cézanne (Aix-Marseille 3) Avenue Escadrille Normandie-Niemen 13397 Marseille Cedex 20 - France - E-mail: `nicola.olivetti@univ.u-3mrs.fr`

** Dipartimento di Informatica - Universitá degli Studi di Torino - corso Svizzera 185 - 10149 Torino - Italia - e-mail: `pozzato@di.unito.it`

none of them, one can infer $retired \mathrel{|\!\sim} work$. In cumulative logics (both **C** and **CL**) one can infer $adult \wedge retired \mathrel{|\!\sim} \neg work$ (giving preference to more specific information), in Preferential logic **P** one can also infer that $adult \mathrel{|\!\sim} \neg retired$ (i.e. typical adults are not retired). In the rational case **R**, if one further knows that $adult \not\mathrel{|\!\sim} \neg married$ (i.e. it is not the case the adults are typically unmarried), one can also infer that $adult \wedge married \mathrel{|\!\sim} work$.

From a semantic point of view, to the each logic (**C**, **CL**, **P**, **R**) corresponds one kind of models, namely, possible-world structures equipped with a preference relation among worlds or states. More precisely, for **P** we have models with a preference relation (an irreflexive and transitive relation) on worlds. For the stronger **R** the preference relation is further assumed to be *modular*. For the weaker logic **CL**, the preference relation is defined on *states*, where a state can be identified, intuitively, with a set of worlds. In the weakest case of **C**, the preference relation is on states, as for **CL**, but it is no longer assumed to be transitive. In all cases, the meaning of a conditional assertion $A \mathrel{|\!\sim} B$ is that B holds in the *most preferred* worlds/states where A holds.

Even if KLM was born as an inferential approach to nonmonotonic reasoning, very few proof systems have been proposed for these logics. In [5] analytic tableaux $\mathcal{T}S^{\mathbf{T}}$ for logics **P** and **CL** have been introduced, where S stands for $\{\mathbf{P}, \mathbf{CL}\}$. The calculi $\mathcal{T}S^{\mathbf{T}}$ are based on a novel interpretation of **P** into modal logic G. The idea is simply to interpret the preference relation as an accessibility relation: a conditional $A \mathrel{|\!\sim} B$ holds in a model if B is true in all A-worlds w, i.e. worlds w where A holds, that are minimal. An A-world is minimal if all smaller worlds are not A-worlds. The relation with modal logic G is motivated by the fact that we assume, following KLM, the so-called *smoothness condition*, which is related to the well-known *limit assumption*. This condition ensures indeed that A-minimal worlds exist, by preventing an infinitely descending chain of worlds. This condition is therefore ensured by the finite-chain condition on the accessibility relation (as in modal logic G).

In this work we describe an implementation of $\mathcal{T}S^{\mathbf{T}}$ calculi in SICStus Prolog. The program, called KLMLean 1.0, is implemented following the methodology of leanTAP, inspired to [1] and [2]. As far as we know, KLMLean 1.0 is the first theorem prover for these logics. KLMLean 1.0 also comprises a graphical user interface written in Java.

The plan of this paper is as follows: in section 2 we briefly recall KLM logics **P** and **CL** and the tableaux calculi $\mathcal{T}S^{\mathbf{T}}$. In section 3 we describe KLMLean 1.0. In section 4 we discuss some related works and suggest some future developments.

2 KLM Logics and the Tableaux Calculi $\mathcal{T}S^{\mathbf{T}}$

2.1 KLM Preferential Logic P

We consider a propositional language $\mathcal{L}$ over a set ATM of propositional variables. Formulas of $\mathcal{L}$ are built from propositional variables by means of the boolean operators $\rightarrow$, $\wedge$, $\vee$, $\neg$ and the conditional operator $\mathrel{|\!\sim}$. The language $\mathcal{L}$

consists just of conditional assertions $A \mathrel{|\!\sim} B$, where A and B are propositional formulas, and of boolean combinations of them. More precisely, the formulas of $\mathcal{L}$ are defined as follows: if A is a propositional formula, $A \in \mathcal{L}$; if A and B are propositional formulas, $A \mathrel{|\!\sim} B \in \mathcal{L}$; if F is a boolean combination of formulas of $\mathcal{L}$, $F \in \mathcal{L}$.

The semantics of **P** is defined by considering possible world structures with a preference relation (a strict partial order) $w < w'$ whose meaning is that w is preferred to w'. We have that $A \mathrel{|\!\sim} B$ holds in a model $\mathcal{M}$ if B holds in all *minimal worlds* (with respect to the relation $<$) where A holds. This definition makes sense provided minimal worlds for A exist whenever there are A-worlds. This is ensured by the *smoothness condition* in the next definition.

Definition 1 (Semantics of P [6]). *A preferential model is a triple $\mathcal{M} = \langle \mathcal{W}, <, V \rangle$ where: $\mathcal{W}$ is a non-empty set of items called worlds; $<$ is an irreflexive and transitive relation on $\mathcal{W}$; V is a function $V : \mathcal{W} \longmapsto pow(ATM)$, which assigns to every world w the set of atoms holding in that world. We define the truth conditions for a formula F as follows:*

- *If F is a boolean combination of formulas, $\mathcal{M}, w \models F$ is defined in the obvious way;*
- *Let A be a propositional formula; we define $Min_<(A) = \{w \in \mathcal{W} \mid \mathcal{M}, w \models A$ and $\forall w', w' < w \quad \mathcal{M}, w' \not\models A\}$;*
- *$\mathcal{M}, w \models A \mathrel{|\!\sim} B$ if for all $w' \in Min_<(A)$, then $\mathcal{M}, w' \models B$.*

The relation $<$ satisfies the following condition, called smoothness*: if $\mathcal{M}, w \models A$ then $w \in Min_<(A)$ or $\exists w' \in Min_<(A)$ such that $w' < w$.*
We say that a formula F is valid *in a model $\mathcal{M}$, denoted with $\mathcal{M} \models F$, if $\mathcal{M}, w \models F$ for every $w \in \mathcal{W}$. A formula is* valid *if it is valid in every model $\mathcal{M}$.*

Notice that the truth conditions for conditional formulas are given with respect to single possible worlds for uniformity sake. Since the truth value of a conditional only depends on global properties of $\mathcal{M}$, we have that: $\mathcal{M}, w \models A \mathrel{|\!\sim} B$ iff $\mathcal{M} \models A \mathrel{|\!\sim} B$.

Now we introduce the language $\mathcal{L}_P$ of the calculus $\mathcal{T}\mathbf{P}^{\mathbf{T}}$. $\mathcal{L}_P$ extends $\mathcal{L}$ by formulas of the form $\Box A$, where A is a propositional formula, whose intuitive meaning is as follows: $\Box A$ holds in a world w if A holds in all the worlds w' such that $w' < w$:

Definition 2 (Truth condition of modality $\Box$, [5]). *We define the truth condition of a boxed formula as follows:*

$$\mathcal{M}, w \models \Box A \text{ if for every } w' \in \mathcal{W} \text{ if } w' < w \text{ then } \mathcal{M}, w' \models A$$

It is easy to see that $\Box$ has the properties of the modal system G: the accessibility relation (defined as xRy iff $y < x$) is transitive and does not have infinite ascending chains. By Definitions 1 and 2, it follows that for any formula A, $w \in Min_<(A)$ iff $\mathcal{M}, w \models A \wedge \Box \neg A$.

2.2 KLM Loop Cumulative Logic CL

The next KLM logic we consider is **CL**, weaker than **P**.

Definition 3 (Loop-cumulative models, [6]). *A loop-cumulative model is a tuple* $\mathcal{M} = \langle S, l, <, V\rangle$. *$S$ is a set, whose elements are called states. Given a set $\mathcal{U}$ of possible worlds, $l : S \mapsto 2^{\mathcal{U}}$ is a function that labels every state with a nonempty set of worlds. $<$ is an irreflexive and transitive relation on S. V is a valuation function $V : \mathcal{U} \longmapsto pow(ATM)$, which assigns to every world w the atoms holding in that world. For $s \in S$ and A propositional formula, we let $s \mid\equiv A$ if $\forall w \in l(s)$, $w \models A$. Let $Min_<(A)$ be the set of minimal states s such that $s \mid\equiv A$. We define $\mathcal{M}, s \mid\equiv A \mathrel{|\!\sim} B$ if $\forall s^{'} \in Min_<(A)$, $s^{'} \mid\equiv B$. We assume that $<$ satisfies the smoothness condition.*

In [5] it is shown that one can map loop-cumulative models into preferential models extended with an additional accessibility relation R. These models are called CL-Preferential structures and are defined below. The idea is to represent states as sets of possible worlds related by R: thus a world plays also a role of a state. By this correspondence we have that a formula is satisfied in a state s just in case it is satisfied in all possible worlds $w^{'}$ accessible from its corresponding w via R. The syntactic counterpart of the extra accessibility relation R is a modality L. Given a loop-cumulative model $\mathcal{M}$ and the corresponding CL-structure $\mathcal{M}^{'}$, $\mathcal{M}, s \mid\equiv A$ iff for its corresponding w, $\mathcal{M}^{'}, w \models LA$. This mapping allows us to use a variant of the tableaux calculus for **P** to deal with system **CL**. As for **P**, the tableaux calculus for **CL** will use boxed formulas. Thus, the formulas that appear in the tableaux for **CL** belong to the language $\mathcal{L}_L$ obtained from $\mathcal{L}$ as follows: (i) if A is a propositional formula, then $A \in \mathcal{L}_L$; $LA \in \mathcal{L}_L$; $\Box\neg LA \in \mathcal{L}_L$; (ii) if A, B are propositional formulas, then $A \mathrel{|\!\sim} B \in \mathcal{L}_L$; (iii) if F is a boolean combination of formulas of $\mathcal{L}_L$, then $F \in \mathcal{L}_L$.

Definition 4 (CL-preferential structures, [5]). *A model has the form $\mathcal{M} = \langle \mathcal{W}, R, <, V\rangle$ where: $\mathcal{W}$, $<$, and V are defined as in Definition 1, and R is a* serial *accessibility relation (i.e. $\forall w \exists w'$ Rww'). We add to the truth conditions for preferential models in Definition 1 the following clause:*

$$\mathcal{M}, w \models LA \text{ if for all } w^{'} \text{ if } wRw^{'} \text{ then } \mathcal{M}, w^{'} \models A$$

The truth condition for conditional formulas is changed as follows: $\mathcal{M}, w \models A \mathrel{|\!\sim} B$ if for all $w^{'} \in Min_<(LA)$ then $\mathcal{M}, w^{'} \models LB$.

Proposition 1 ([5]). *A set of conditional formulas $\{(\neg)A_1 \mathrel{|\!\sim} B_1, \ldots, (\neg)A_n \mathrel{|\!\sim} B_n\}$ is satisfiable in a loop-cumulative model $\langle S, l, <, V\rangle$ iff it is satisfiable in a CL-preferential model $\langle W, R, <, V\rangle$.*

2.3 The Tableaux Calculi $\mathcal{TS}^{\mathbf{T}}$

In Figures 1 and 2 we present the tableaux calculi $\mathcal{T}\mathbf{P}^{\mathbf{T}}$ and $\mathcal{T}\mathbf{CL}^{\mathbf{T}}$ introduced in [5]. In order to obtain terminating calculi, the $(\mathrel{|\!\sim}^{+})$ rule must be applied

under some restrictions that for clarity are not displayed in the figures below, but are explained later in this section.

The rules of the calculi manipulate sets of formulas Γ. We write Γ, F to denote $\Gamma \cup \{F\}$. Moreover, we consider the following sets:

- $\Gamma^{\Box} = \{\Box A \mid \Box A \in \Gamma\}$
- $\Gamma^{\Box^{\downarrow}} = \{A \mid \Box A \in \Gamma\}$
- $\Gamma^{\not\sim^{+}} = \{A \not\sim B \mid A \not\sim B \in \Gamma\}$
- $\Gamma^{\not\sim^{-}} = \{\neg(A \not\sim B) \mid \neg(A \not\sim B) \in \Gamma\}$
- $\Gamma^{\not\sim^{\pm}} = \Gamma^{\not\sim^{+}} \cup \Gamma^{\not\sim^{-}}$
- $\Gamma^{L^{\downarrow}} = \{A \mid LA \in \Gamma\}$

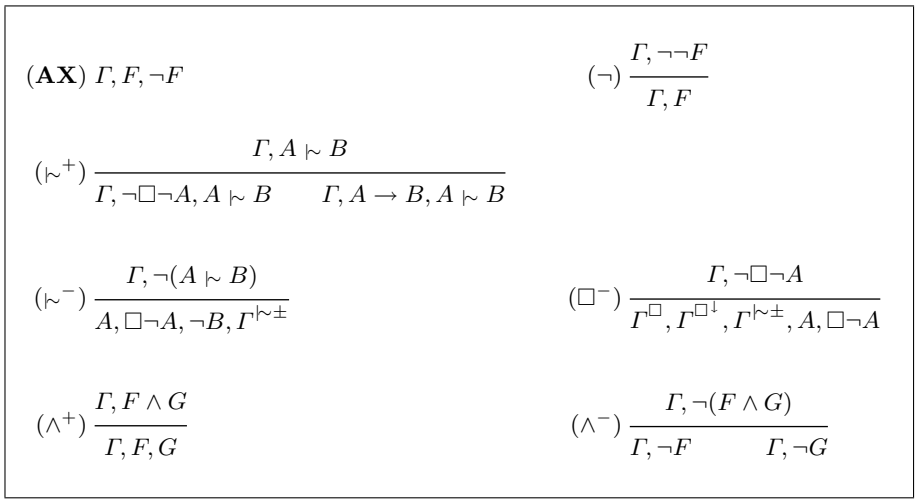


Fig. 1. Tableaux system $\mathcal{T}\mathbf{P}^{\mathbf{T}}$. To save space, we omit rules for the other boolean connectives.

Theorem 1 (Soundness and completeness of $\mathcal{T}\mathbf{P}^{\mathbf{T}}$ and $\mathcal{T}\mathbf{CL}^{\mathbf{T}}$, [5]). *The systems $\mathcal{T}\mathbf{P}^{\mathbf{T}}$ and $\mathcal{T}\mathbf{CL}^{\mathbf{T}}$ are sound and complete with respect to the semantics, i.e. there is a closed tableaux for a set Γ iff Γ is unsatisfiable.*

In order to obtain a decision procedure we have to control the application of the $(\not\sim^{+})$ rule, since it copies its principal formula in its conclusions. To this regard we have the following result:

Theorem 2 ([5]). *$\mathcal{T}\mathbf{P}^{\mathbf{T}}$ and $\mathcal{T}\mathbf{CL}^{\mathbf{T}}$ are sound and complete even if $(\not\sim^{+})$ is applied at most once on the same conditional $A \not\sim B$ in the same world. More*

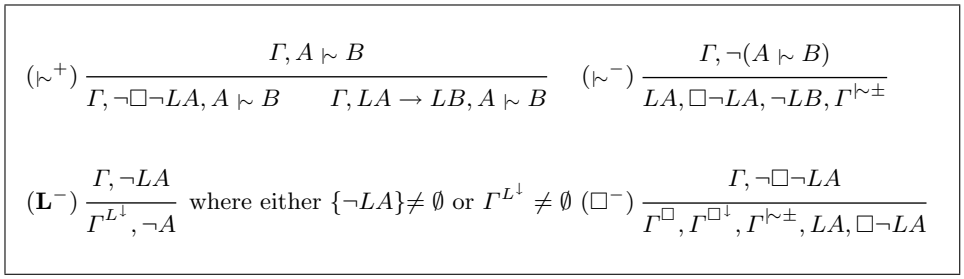


Fig. 2. Tableaux system $\mathcal{T}\mathbf{CL}^{\mathbf{T}}$. If $\neg LA$ is not in the premise of (L^{-}) (i.e. $\{\neg LA\} = \emptyset$) the rule allows to step from Γ to $\Gamma^{L^{\downarrow}}$. The boolean rules are omitted.

precisely, when $(\mid\!\sim^{+})$ *is applied to* $A \mid\!\sim B$*,* $A \mid\!\sim B$ *is marked as* used*; the applications of* $(\mid\!\sim^{+})$ *are restricted to unused conditionals only; when* $(\Box^{-})$ *or* $(\mid\!\sim^{-})$ *is applied, all conditionals become unused*[1]*.*

The generation of infinite branches due to the interplay between $(\mid\!\sim^{+})$ and $(\Box^{-})$ cannot occur, because of the properties of $\Box$ in modal logic G (see [5] for details). These results give a constructive proof of decidability of Preferential and Loop-Cumulative logics.

3 Design of KLMLean 1.0

In this section we present an implementation of the tableaux calculi $\mathcal{T}S^{\mathbf{T}}$ called **KLMLean 1.0**; it is a SICStus Prolog program inspired by leanTAP ([1], [2]) and follows the line of CondLean, a theorem prover for other conditional logics introduced in [9] and [10]. The idea of *lean deduction* [1] is "to achieve maximal efficiency from minimal means", that is to say to write short programs and exploit the power of Prolog's engine as much as possible. In [1] authors remark that another important argument for lean methodology is safety: to verify soundness and completeness of a theorem prover consisting of a couple of lines is easier than doing the same for thousands of lines of C code. Moreover, in [1] it is observed that programs following the lean methodology offer high performances on simple to moderately complex problems, and that they offer a high degree of adaptability to applications.

It is worth noticing that KLMLean 3.1 is only *inspired* by leanTAP, however it does not follow strictly the style of leanTAP. For instance, KLMLean 3.1 makes use of auxiliary Prolog's predicates like `member` and `select`, whereas leanTAP only relies on Prolog's clause indexing scheme and backtracking.

[1] The premise and the conclusion of $(\Box^{-})$ (resp. $(\mid\!\sim^{-})$) represent two different worlds. When $(\Box^{-})$ (resp. $(\mid\!\sim^{-})$) is applied, then all conditional formulas used in the world represented by the premise become unused in the world represented by the conclusion.

The program KLMLean 1.0 comprises a set of clauses, each one of them represents a tableaux rule or axiom. The proof search is provided for free by the mere depth-first search mechanism of Prolog, without any additional ad hoc mechanism. We represent each node of a proof tree by a *list* of formulas. The tableaux calculi are implemented by the predicate **prove(Gamma,Sigma,Tree).** which succeeds if and only if the set of formulas Γ, represented by the Prolog list `Gamma`, is unsatisfiable. `Sigma` is the list representing the set Σ of *used conditionals*, and it is needed in order to control the application of the $(\mid\!\sim^{+})$ rule only to unused conditional, as stated by Theorem 2 above. If the predicate succeeds, `Tree` contains the closed proof tree found by the theorem prover.

For instance, to prove that $A \mid\!\sim B \wedge C, \neg(A \mid\!\sim C)$ is unsatisfiable in $\mathbf{P}$, one queries KLMLean 1.0 with the following goal:

```
prove([a => (b and c), neg (a => c)],[ ],Res)).
```

The string `=>` is used to represent the conditional operator $\mid\!\sim$, `and` is used to denote $\wedge$, and so on. Each clause of `prove` implements one axiom or rule of $\mathcal{T}S^{\mathbf{T}}$; for example, we present the clause implementing the axiom:

```
prove(Gamma,_,tree(ax,F)):-
   member(neg F,Gamma),
   member(F, Gamma),!.
```

If the formula F and its negation $\neg F$ both belong to the set of formulas Γ, then the predicate `prove` succeeds by the above clause. As mentioned above, the third argument of the predicate `prove` contains the proof tree found by the theorem prover; in this case, it corresponds to a leaf represented by the functor `tree(ax,F)`, to keep trace that the tree is closed by the presence of both formulas `F` and `neg F`.

As another example, the clause implementing $(\mid\!\sim^{+})$ is as follows:

```
prove(Gamma,Sigma,tree(entP,A,B,ST1,ST2)):-
   select(A => B,Gamma,NewGamma),!,
   prove([neg box neg A|NewGamma],[A => B|Sigma],ST1),!,
   prove([A -> B|NewGamma],[A => B|Sigma],ST2).
```

The predicate `select(X,List,NewList)` succeeds if and only if an item `X` belongs to `List`, returning the list `NewList` in which `X` has been deleted from `List`. Therefore, if a formula $A \mid\!\sim B$ belongs to Γ, then KLMLean 1.0 applies the $(\mid\!\sim^{+})$ rule, by invoking recursively the predicate `prove` on its two conclusions. In the first one, the formula $\neg\Box\neg A$ is introduced; in the other one, $A \rightarrow B$ is added to the set of formulas to analyze. In both the recursive calls to the predicate `prove` the principal formula $A \mid\!\sim B$ is copied into the auxiliary list `Sigma`, in order to avoid further applications of $(\mid\!\sim^{+})$ in the same world.
The functor `tree(entP,A,B,ST1,ST2)` represents the proof tree obtained by applying the above clause. More in detail, its terms represent that the $(\mid\!\sim^{+})$ rule

has been applied (`entP`) to a conditional formula `A => B`; `ST1` and `ST2` are the closed subtrees obtained by invoking `prove` on the two conclusions of the rule.

To search a closed tree of a set Γ of formulas of $\mathcal{L}$, KLMLean 1.0 proceeds as follows. First of all, if Γ is an axiom, then the goal will succeed immediately by using the clause for the axiom presented above. If it is not, then the first applicable rule will be chosen, e.g. if `Gamma` contains a formula `[A and B]`, then the clause for the $(\wedge^+)$ rule will be used, invoking `prove` on the unique conclusion of $(\wedge^+)$. KLMLean 1.0 proceeds in a similar way for the other rules. The ordering of the clauses is such that the application of the branching rules and of the conditional and box rules is postponed as much as possible. In the above example, it is clear how KLMLean implement the controlled application of $(\mathrel{|\!\sim}^+)$ on a conditional formula $A \mathrel{|\!\sim} B$ (Theorem 2): $A \mathrel{|\!\sim} B$ passes from Γ to the set Σ of used conditionals, therefore the above clause will not further apply to it (it works only on conditionals belonging to Γ).

KLMLean has also a graphical user interface (GUI) implemented in Java. The GUI interacts with the SICStus Prolog implementation by means of the package `se.sics.jasper`. Thanks to the GUI, one does not need to know how to call the predicate `prove`: one just introduces

- formulas of a knowledge base K (or the set of formulas to prove to be unsatisfiable);
- a formula F, in order to prove if one can infer F from K, corresponding to verify if $K \cup \{\neg F\}$ is unsatisfiable

in a text box and searches a derivation by clicking a button. Moreover, one can choose the intended system of conditional logic, namely **P** or **CL**. When the submitted set $K \cup \{\neg F\}$ of formulas is unsatisfiable, KLMLean 1.0 offers these options:

- display a proof tree of the set in a special window;
- build a latex file containing the same proof tree: compiling this file with Latex, one can obtain the closed tree in a pdf file, or ps, or dvi, and then print it.

Some pictures of KLMLean 1.0 are presented in Figures 3, 4, and 5.

4 Conclusions, Comparison with Other Works, and Future Work

In this work we have presented KLMLean 1.0, a theorem prover for KLM Preferential and Loop-Cumulative logics. To the best of our knowledge, this is the first theorem prover for logics of the KLM framework. KLMLean 1.0 is a SICStus Prolog implementation of the analytic tableaux calculi introduced in [5], and follows the leanTAP methodology of [1] and [2].

We briefly remark on some related works. We just mention the program CondLean, a theorem prover for some standard conditional logics ([9], [10]).

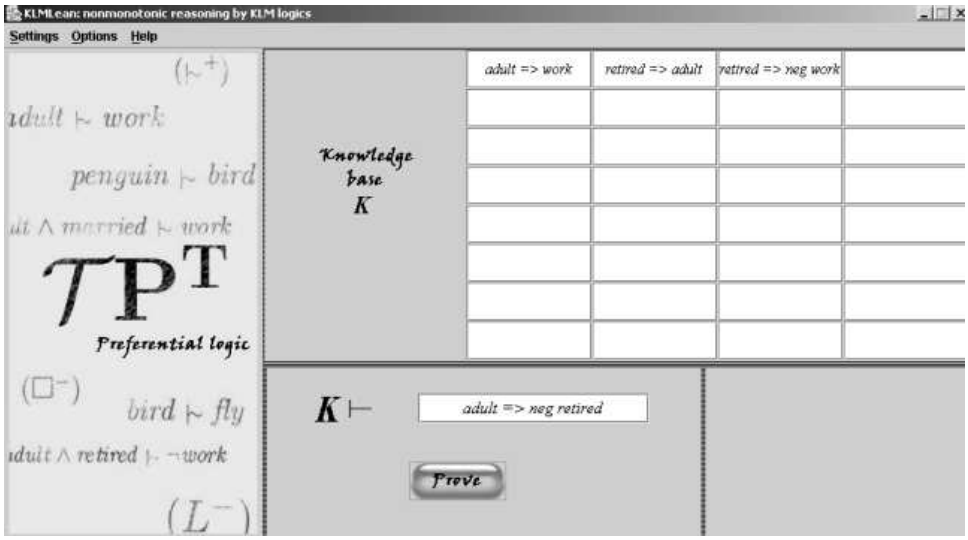


Fig. 3. The main window of KLMLean 1.0.

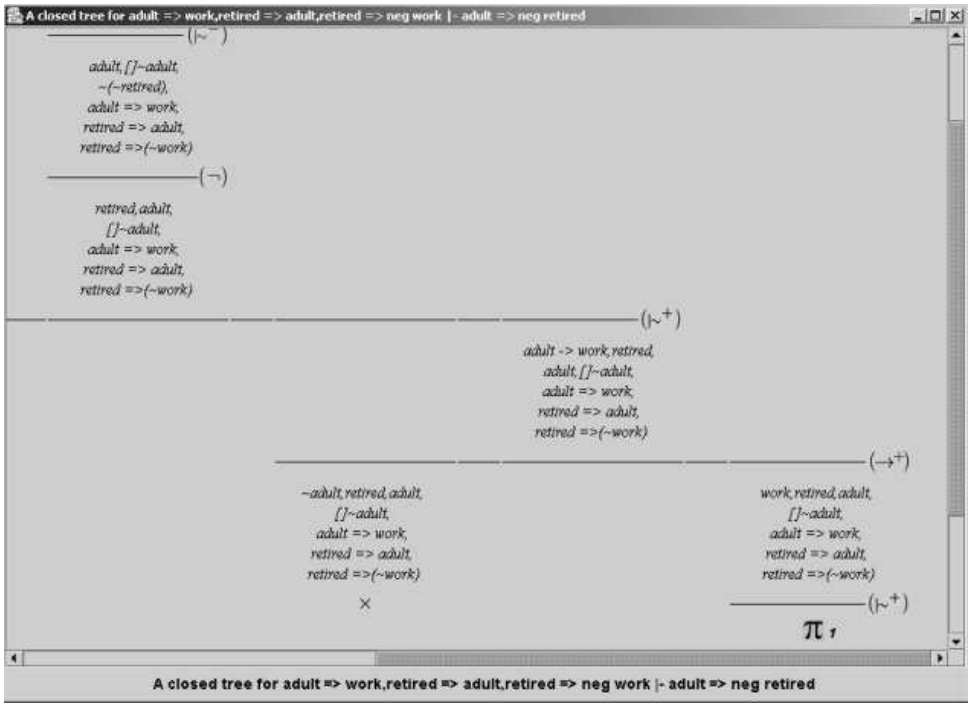


Fig. 4. A proof tree in a special window.

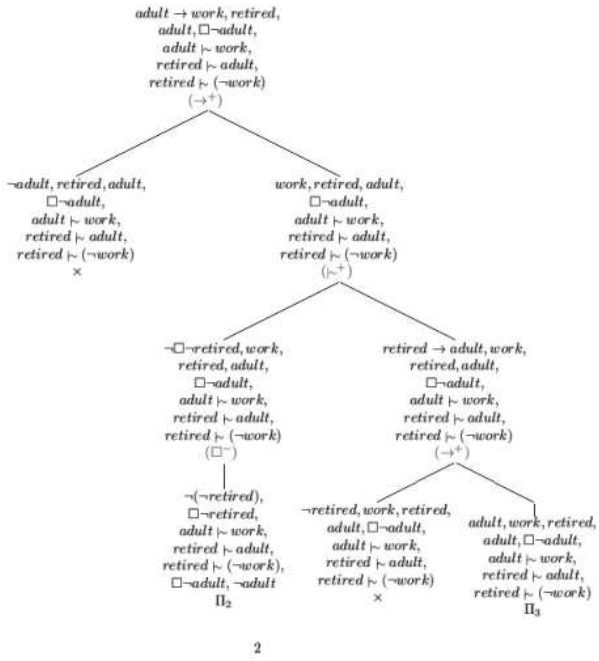


Fig. 5. The proof tree printed in a .dvi file.

CondLean is a SICStus Prolog implementation of some labelled sequent calculi for conditional logics CK, ID, MP, CS, CEM, and some combinations of them. It follows the leanTAP methodology, too. The main difference between CondLean and KLMLean is that they offer a deductive mechanism for different conditional logics: CondLean is a theorem prover for conditional logics whose semantics is based on the so called *selection function* [8], whereas KLMLean is a theorem prover for KLM logics.

In future research, we intend to extend KLMLean to the other KLM conditional logics, namely the weaker Cumulative logic **C** and the stronger Rational logic **R**. Moreover, we intend to make a detailed study on the performances of our theorem prover, and experiment some standard refinements and heuristics to increase its efficiency. Furthermore, we are improving KLMLean 1.0 in such a way that, given a satisfiable set of formulas in input, it shows a model of that satisfiable set.
A further extension is to develop a theorem prover for the first order version of these logics, which have recently received a renewed attention thanks to the results given in [3].

References

1. Beckert, B. and J. Posegga, *leantap: Lean tableau-based deduction*, Journal of Automated Reasoning, 15(3) (1995), pp. 339–358.
2. Fitting, M., *leantap revisited*, Journal of Logic and Computation, 8(1) (1998), pp. 33–47.
3. Friedman, N., J. Y. Halpern and D. Koller, *First-order conditional logic for default reasoning revisited*, ACM TOCL, ACM Press **1** (2000), pp. 175–207.
4. Gabbay, D., *Theoretical foundations for non-monotonic reasoning in expert systems*, Logics and models of concurrent systems, Springer (1985), pp. 439–457.
5. Giordano, L., V. Gliozzi, N. Olivetti and G. L. Pozzato, *Analytic tableaux for klm preferential and cumulative logics*, In Proc. of 12th Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2005), volume 3835 of LNAI, Springer-Verlag (2005), pp. 666–681.
6. Kraus, S., D. Lehmann and M. Magidor, *Nonmonotonic reasoning, preferential models and cumulative logics*, Artificial Intelligence, 44(1-2) (1990), pp. 167–207.
7. Lehmann, D. and M. Magidor, *What does a conditional knowledge base entail?*, Artificial Intelligence, Elsevier Science Publishers Ltd. **55** (1992), pp. 1–60.
8. Nute, D., *Topics in conditional logic*, Reidel (1980).
9. Olivetti, N. and G. L. Pozzato, *Condlean: A theorem prover for conditional logics*, In Proc. of TABLEAUX 2003, vol. 2796 of LNAI, Springer (2003), pp. 264–270.
10. Olivetti, N. and G. L. Pozzato, *Condlean 3.0: Improving condlean for stronger conditional logics*, In Proc. of TABLEAUX 2005, vol. 3702 of LNAI, Springer (2005), pp. 328–332.
11. Shoham, Y., *A semantical approach to nonmonotonic logics*, In Proceedings of Logics in Computer Science (1987), pp. 275–279.

Description Logics and Disjunctive Datalog—More Than just a Fleeting Resemblance?

Boris Motik

Forschungszentrum Informatik
Karlsruhe, Germany
motik@fzi.de

Abstract

As applications of description logics (DLs) proliferate, efficient reasoning with large ABoxes (sets of individuals with descriptions) becomes ever more important. Motivated by the prospects of reusing optimization techniques of deductive databases, we developed a novel algorithm for reasoning in description logic, which reduces a DL knowledge base to a disjunctive datalog program without changing the set of relevant consequences. This allows to answer queries by applying optimization techniques, such as join-order optimizations or magic sets. The algorithm supports the very expressive logic $\mathcal{SHIQ}(\mathbf{D})$, so the reduction is quite technically involved. In this paper we present a simplified algorithm for the basic logic $\mathcal{ALC}$. Whereas this algorithm is much easier to understand, it is based on the same principles as the general one.

1 Introduction

In recent years description logics (DLs) have found their application in various fields of computer science, so several DL reasoners were built and applied to practical problems. The state-of-the-art systems are based on tableau algorithms [11], and they perform quite well when computing the subsumption hierarchy, mainly due to sophisticated heuristics [10].

However, new applications, such as metadata management in the Semantic Web, require efficient query answering over large ABoxes (i.e., sets of ground facts). Query answering is currently implemented by a reduction to ABox consistency checking, which can then be solved by tableau algorithms. Whereas this is quite elegant from a theoretical point of view, it does not provide good performance in practice, mainly due to the following two reasons. First, tableau-based algorithms treat all individuals separately; that is, they do not group individuals on common properties. Second, to answer a query, one

usually does not need to consider all ABox information; rather, only a small subset of the ABox usually suffices to compute the query answer. These deficiencies were already acknowledged, and certain optimization techniques were developed [9]; however, the performance of query answering is still often not satisfactory in practice.

Since techniques for reasoning in deductive databases are nowadays mature, it makes sense to examine if they can be used to improve ABox reasoning in description logics. To facilitate that, we developed a novel technique for reducing a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base to a disjunctive datalog program without changing the set of entailed ground facts [13,14,12]. This algorithm addresses the mentioned points in the following way. First, query answering in disjunctive datalog can be implemented by manipulating individuals in sets, and applying each inference rule to all individuals in a set at once, rather than to each individual separately. This allows to apply the join order optimization, which uses database statistics to compute the data access path promising the least cost [1]. Second, the magic sets transformation [4] can be used to identify the subset of the database relevant to the query; this transformation was recently generalized to disjunctive programs [6].

$\mathcal{SHIQ}(\mathbf{D})$ is a complex logic, with modeling primitives such as number restrictions and concrete datatypes. This makes the reduction algorithm fairly complex, mainly because it is based on *basic superposition* [3,18]—a sophisticated calculus for theorem proving with equality. In this paper, we present the algorithm scaled down to the basic description logic $\mathcal{ALC}$. This makes the algorithm much simpler; in particular, it is based on the well-known ordered resolution calculus [2]. However, $\mathcal{ALC}$ exhibits features characteristic of most DLs, such as boolean concept constructors, existential and universal quantification, as well as general concept inclusion axioms, so this simplified algorithm succinctly demonstrates the important points of the general one.

Our results seem to integrate two fundamentally different logics. Therefore, in this paper we also answer common questions about the apparent mismatch in complexity between the two formalisms, or why a logic based on general first-order semantics and without unique name assumption can be embedded into a logic based on minimal-model semantics and unique name assumption.

2 Preliminaries

In this section we introduce the formalisms used in the paper.

2.1 Description Logics

Let N_R be a set of *roles*, and N_C a set of *concept names*. The set of $\mathcal{ALC}$ concepts is the minimal set satisfying the following conditions: (i) $\top$ and $\bot$ are $\mathcal{ALC}$ concepts; (ii) each $A \in N_C$ is an $\mathcal{ALC}$ concept; and (iii) $\neg C$, $C \sqcap D$, $C \sqcup D$, $\forall R.C$, and $\exists R.C$ are $\mathcal{ALC}$ concepts, for C and D $\mathcal{ALC}$ concepts and R

Table 1
Semantics of $\mathcal{ALC}$ by Mapping to FOL

Translating Concepts to FOL	
$\pi_y(\top, X) = \top$	$\pi_y(\bot, X) = \bot$
$\pi_y(A, X) = A(X)$	$\pi_y(\neg C, X) = \neg\pi_y(C, X)$
$\pi_y(C \sqcap D, X) = \pi_y(C, X) \wedge \pi_y(D, X)$	$\pi_y(C \sqcup D, X) = \pi_y(C, X) \vee \pi_y(D, X)$
$\pi_y(\forall R.C, X) = \forall y : R(X, y) \rightarrow \pi_x(C, y)$	$\pi_y(\exists R.C, X) = \exists y : R(X, y) \wedge \pi_x(C, y)$
Translating Axioms to FOL	
$\pi(C(a)) = \pi_y(C, a)$	$\pi(R(a, b)) = R(a, b)$
$\pi(C \sqsubseteq D) = \forall x : \pi_y(C, x) \rightarrow \pi_y(D, x)$	$\pi(KB) = \bigwedge_{\alpha \in KB_{\mathcal{T}} \cup KB_{\mathcal{A}}} \pi(\alpha)$

X is a meta variable and is substituted by the actual variable.
π_x is obtained from π_y by simultaneously substituting in the definition $x_{(i)}$ for all $y_{(i)}$, respectively, and π_y for π_x.

a role. Concept names are also called *atomic concepts*, and other concepts are also called *complex concept*. A *literal* concept is a possibly negated concept name. A TBox $KB_{\mathcal{T}}$ is a finite set of concept inclusion axioms of the form $C \sqsubseteq D$. An ABox $KB_{\mathcal{A}}$ is a finite set of axioms $C(a)$ and $R(a, b)$, for C an $\mathcal{ALC}$ concept and R a role. An $\mathcal{ALC}$ knowledge base KB is a tuple $(KB_{\mathcal{T}}, KB_{\mathcal{A}})$.

The semantics of KB is given by translating it into first-order logic by the operator π from Table 1. The main inference problem is checking KB satisfiability—that is, determining if a first-order model of $\pi(KB)$ exists. An individual a is an *instance* of a concept C w.r.t. KB if $\pi(KB) \models \pi_y(C, a)$, which is the case if and only if $KB \cup \{\neg C(a)\}$ is unsatisfiable.

We measure the *size* of concepts by their length. An $\mathcal{ALC}$ concept is in *negation-normal form* (NNF) if negations occur only in front of concepts names. An ABox is *extensionally reduced* if all ABox axioms contain only literal concepts. If KB is not extensionally reduced, it can be easily transformed into an extensionally reduced knowledge base: for each axiom $C(a)$ where C is not a literal concept, one should introduce a new atomic concept A_C, add the axiom $A_C \sqsubseteq C$ to the TBox, and replace $C(a)$ with $A_C(a)$. Such a transformation is obviously polynomial, so without losing generality it is safe to assume that a knowledge base is extensionally reduced.

We adapt the notions of positions to DL formulae. A *position* p is a finite sequence of integers; the empty position is denoted with ϵ. If a position p_1 is a proper prefix of a position p_2, then p_1 is *above* p_2, and p_2 is *below* p_1. For a concept α, the subterm at a position p, written $\alpha|_p$, is defined as follows: $\alpha|_\epsilon = \alpha$; $(\neg D)|_{1p} = D|_p$; $(D_1 \circ D_2)|_{ip} = D_i|_p$ for $\circ \in \{\sqcap, \sqcup\}$ and $i \in \{1, 2\}$; and $\alpha|_1 = R$ and $\alpha|_{2p} = D|_p$ for $\alpha = \Diamond R.D$ and $\Diamond \in \{\exists, \forall\}$. A *replacement* of a subterm of α at position p with a term β is defined in the standard way and is denoted as $\alpha[\beta]_p$.

2.2 Disjunctive Datalog

A *datalog term* is a constant or a variable, and a *datalog atom* has the form $A(t_1, \ldots, t_n)$, where t_i are datalog terms. A *disjunctive datalog program* P is a finite set of rules of the form $A_1 \vee ... \vee A_n \leftarrow B_1, ..., B_m$ where A_i and B_j are datalog atoms. Each rule is required to be *safe*; that is, each variable occurring in the rule must occur in at least one body atom. A *fact* is a rule with $m = 0$. For the semantics, we take a rule to be equivalent to a clause $A_1 \vee ... \vee A_n \vee \neg B_1 \vee ... \vee \neg B_m$. We consider only Herbrand models, and say that a model M of P is *minimal* if there is no model M' of P such that $M' \subsetneq M$. A ground literal A is a *cautious answer* of P (written $P \models_c A$) if A is true in all minimal models of P. First-order entailment coincides with cautious entailment for positive ground atoms.

2.3 Ordered Resolution

We assume the reader to be familiar with standard definitions of first-order logic and clausal theorem proving. Ordered resolution [2] is one of the most widely used calculi for theorem proving in first-order logic. It is parameterized with an *admissible* ordering $\succ$ on literals (for details about the condition of admissibility please refer to [2]), and by a *selection function* S which assigns to each clause C a possibly empty subset of negative literals of C. With $\mathcal{R}$ we denote the calculus consisting of the following inference rules:

Positive factoring:
$$\frac{C \vee A \vee B}{C\sigma \vee A\sigma}$$

where (i) $\sigma = \mathsf{MGU}(A, B)$, (ii) $A\sigma$ is strictly maximal with respect to $C\sigma \vee B\sigma$, and no literal is selected in $C\sigma \vee A\sigma \vee B\sigma$.

Ordered resolution:
$$\frac{C \vee A \quad D \vee \neg B}{C\sigma \vee D\sigma}$$

where (i) $\sigma = \mathsf{MGU}(A, B)$, (ii) $A\sigma$ is strictly maximal with respect to $C\sigma$, and no literal is selected in $C\sigma \vee A\sigma$, (iii) $\neg B\sigma$ is either selected in $D\sigma \vee \neg B\sigma$, or it is maximal with respect to $D\sigma$ and no literal is selected in $D\sigma \vee \neg B\sigma$.

The clauses $C \vee A \vee B$ and $D \vee \neg B$ are the *main premises*, $C \vee A$ is the *side premise*, and $C\sigma \vee A\sigma$ and $C\sigma \vee D\sigma$ are the *conclusions*. Ordered resolution is sound and complete: if a set of clauses N is saturated up to redundancy by $\mathcal{R}$, then N is satisfiable if and only if it does not contain the empty clause.

3 The Main Difficulty in Reducing DLs to Datalog

For an $\mathcal{ALC}$ knowledge base KB, our goal is to derive a disjunctive datalog program $\mathsf{DD}(KB)$ such that $KB \models \alpha$ if and only if $\mathsf{DD}(KB) \models \alpha$, for α of

the form $A(a)$ or $R(a, b)$. In other words, KB and $\mathsf{DD}(KB)$ should entail the same set of positive ground facts. We may thus use $\mathsf{DD}(KB)$ instead of KB for query answering, and in doing so we may apply all optimization techniques known from the field of deductive databases.

As shown by the definition of $\mathcal{ALC}$ and by Borgida [5], there is a close correspondence between description logics and first-order logic. Consider a knowledge base $KB = \{A \sqsubseteq \exists R.A,\ \exists R.\exists R.A \sqsubseteq B,\ A(a)\}$. A naïve attempt to reduce KB into disjunctive datalog is to compute a first-order formula $\pi(KB)$, skolemize it, translate it into conjunctive normal form, and rewrite the obtained set of clauses as rules, yielding the following logic program $\mathsf{LP}(KB)$:

$$R(x, f(x)) \leftarrow A(x) \tag{1}$$

$$A(f(x)) \leftarrow A(x) \tag{2}$$

$$B(x) \leftarrow R(x, y), R(y, z), A(z) \tag{3}$$

$$A(a) \tag{4}$$

Clearly, KB and $\mathsf{LP}(KB)$ entail the same set of ground facts. However, $\mathsf{LP}(KB)$ contains a function symbol in a recursive rule (2). This raises the issue of how to answer queries in $\mathsf{LP}(KB)$. Namely, well-known query evaluation techniques, such as bottom-up saturation, will not terminate on $\mathsf{LP}(KB)$: we shall derive $A(f(a))$, $R(a, f(a))$, $A(f(f(a)))$, $R(f(a), f(f(a)))$, $B(a)$, and so on. Note that we need all previously derived facts to derive $B(a)$ from $\mathsf{LP}(KB)$, and that we do not know when all relevant ground facts have been derived.

To eliminate potential problems with termination, our goal is to derive a true disjunctive datalog program $\mathsf{DD}(KB)$—that is, a program without function symbols. For such a program, queries can be evaluated using any standard technique; furthermore, all known optimization strategies can be applied. Hence, the main problem that we address is how to eliminate function symbols from $\mathsf{LP}(KB)$.

To obtain the desired reduction, we start off with a slightly simpler task of deriving a program $\mathsf{DD}(KB)$ that is satisfiable if and only if KB is satisfiable. This we base on the following simple idea. Let us assume that unsatisfiability of KB can be demonstrated by a refutation in some sound and complete calculus $\mathcal{C}$. If it is possible to simulate inferences of $\mathcal{C}$ on KB using a datalog program $\mathsf{DD}(KB)$, a refutation in KB by $\mathcal{C}$ can be reduced to a refutation in $\mathsf{DD}(KB)$. Conversely, if $\mathsf{DD}(KB)$ is unsatisfiable, there is a refutation in $\mathsf{DD}(KB)$. If it is possible to simulate inferences in $\mathsf{DD}(KB)$ by the calculus $\mathcal{C}$ on KB, then a refutation in $\mathsf{DD}(KB)$ can be reduced to a refutation in KB.

To obtain a sound, complete, and terminating algorithm, we must select an appropriate calculus $\mathcal{C}$, capable of effectively deciding satisfiability of KB. Disjunctive datalog is strongly related to clausal first-order logic, so simulating inferences is easier if $\mathcal{C}$ is a clausal refutation calculus. Hence, to obtain a reduction algorithm, we first derive a decision procedure for $\mathcal{ALC}$ based on the ordered resolution calculus $\mathcal{R}$. In particular, in Section 4.1 we show how

to translate KB into an equisatisfiable set of clauses $\Xi(KB)$, and in Section 4.2 we show that exhaustive application of the inference rules of $\mathcal{R}$ on $\Xi(KB)$ eventually terminates. Since $\mathcal{R}$ is sound and complete, $\mathcal{R}$ decides satisfiability of $\Xi(KB)$, and therefore of KB as well.

Based on such a procedure, in Section 5 we derive the desired reduction of KB to a disjunctive datalog program. It turns out that, for $\mathcal{ALC}$, simulating inferences of $\mathcal{R}$ in disjunctive datalog is quite straightforward. After presenting several simple examples in Section 6, in Section 7 we discuss some interesting aspects of our algorithms.

4 Deciding Satisfiability of KB by Resolution

We now present an algorithm for deciding satisfiability of KB by ordered resolution. In the following sections, with Cls we denote the standard clausification operator: for a first-order formula φ, $\mathsf{Cls}(\varphi)$ is the set of clauses obtained by solemizing φ and translating it into conjunctive normal form.

4.1 Translating KB into Clauses

The first step in deciding satisfiability of an $\mathcal{ALC}$ knowledge base KB by the ordered resolution calculus $\mathcal{R}$ is to transform KB into an equisatisfiable set of clauses $\Xi(KB)$. A straightforward way to do so is to apply the operator Cls; however, such an algorithm has two important drawbacks. First, the size of $\mathsf{Cls}(\pi(KB))$ could be exponential in the size of $\pi(KB)$, due to nesting of $\sqcap$ and $\sqcup$ connectives. Second, since KB is an $\mathcal{ALC}$ knowledge base, the formula $\pi(KB)$ is of a particular syntactic structure, which we exploit in the decision procedure. Therefore, we apply the *structural transformation*, introduced in [19]. Next, we present an alternative, but equivalent definition.

Definition 4.1 Let C be an $\mathcal{ALC}$ concept in negation-normal form. A position $p \neq \epsilon$ in C is *eligible for replacement* if $C|_p$ is of the form $\bigsqcup L_i$, $\bigsqcap L_i$, $\forall R.L$, or $\exists R.L$, where $L_{(i)}$ are all literal concepts. The definitorial form of C, written $\mathsf{Def}(C)$, is defined recursively as follows:

$$\mathsf{Def}(C) = \begin{cases} \{C\} & \text{if } C \text{ is a literal concept} \\ \{\neg Q \sqcup C|_p\} \cup \mathsf{Def}(C[Q]_p) & \text{if } p \text{ is eligible for replacement in } C \end{cases}$$

For example, $\mathsf{Def}(\exists R.(\forall S.\neg A)) = \{\exists R.Q,\ \neg Q \sqcup \forall S.\neg A\}$. Note that the concept $\neg Q \sqcup \forall S.\neg A$ can be interpreted as $Q \sqsubseteq \forall S.\neg A$, which defines Q as a new name for $\forall S.\neg A$. Furthermore, as shown by the following lemma, this transformation does not affect satisfiability.

Lemma 4.2 *For a concept C in negation-normal form, the axiom $\top \sqsubseteq C$ is satisfiable if and only if the set of axioms $\{\top \sqsubseteq D_i \mid D_i \in \mathsf{Def}(C)\}$ is satisfiable.*

Table 2
Types of $\mathcal{ALC}$-clauses

1	$\mathbf{P}(x) \vee R(x, f(x))$
2	$\mathbf{P_1}(x) \vee \mathbf{P_2}(\mathbf{f}(x))$
3	$\mathbf{P_1}(x) \vee \neg R(x, y) \vee \mathbf{P_2}(y)$
4	$\mathbf{P}(\mathbf{a})$
5	$(\neg)R(a, b)$

Proof. We apply induction on the number of recursive invocations of Def. The induction base is trivial, so we consider the induction step.

($\Rightarrow$) Let I be a model satisfying $\top \sqsubseteq C$, and let I' be an extension of I such that $Q^{I'} = (C|_p)^I$. Obviously, I' satisfies $\top \sqsubseteq C[Q]_p$ and $\top \sqsubseteq \neg Q \sqcup C|_p$.

($\Leftarrow$) Observe that in each model I of $\top \sqsubseteq \neg Q \sqcup C|_p$, we have $Q^I \subseteq (C|_p)^I$. By induction on the structure of $D = C[Q]_p$, it is easy to show that, for each position q in D, $(D|_q)^I \subseteq (C|_q)^I$. Since $\triangle^I \subseteq (D|_q)^I$ for each q, I is a model of $\top \sqsubseteq C$. □

The set of clauses $\Xi(KB)$, encoding an $\mathcal{ALC}$ knowledge base KB in first-order logic, is defined as follows:

Definition 4.3 The operator Cls is extended to $\mathcal{ALC}$ concepts as follows:

$$\mathsf{Cls}(C) = \bigcup_{D \in \mathsf{Def}(\mathsf{NNF}(C))} \mathsf{Cls}(\forall x : \pi_y(D, x))$$

For an extensionally reduced $\mathcal{ALC}$ knowledge base KB, $\Xi(KB)$ is the smallest set of clauses such that (i) $\mathsf{Cls}(\pi(\alpha)) \subseteq \Xi(KB)$ for each ABox axiom α in KB; and (ii) $\mathsf{Cls}(\neg C \sqcup D) \subseteq \Xi(KB)$ for each TBox axiom $C \sqsubseteq D$ in KB. If KB is not extensionally reduced, then $\Xi(KB) = \Xi(KB')$, where KB' is an extensionally reduced knowledge base obtained from KB as shown in Section 2.1.

We define *$\mathcal{ALC}$-clauses* to be clauses of the form from Table 2. For a term t, $\mathbf{P}(t)$ denotes a possibly empty disjunction $(\neg)P_1(t) \vee \ldots \vee (\neg)P_n(t)$, and $\mathbf{P}(\mathbf{f}(x))$ denotes a possibly empty disjunction $\mathbf{P_1}(f_1(x)) \vee \ldots \vee \mathbf{P_m}(f_m(x))$. Note that this allows each $\mathbf{P_i}(f_i(x))$ to contain positive and negative literals.

We now show that clausification does not affect the semantics of a knowledge base, and that it produces only $\mathcal{ALC}$ clauses:

Lemma 4.4 *Let KB be an $\mathcal{ALC}$ knowledge base. Then, (*i*) KB is satisfiable if and only if $\Xi(KB)$ is satisfiable; (*ii*) $\Xi(KB)$ can be computed in time polynomial in $|KB|$; and (*iii*) each clause in $\Xi(KB)$ is an $\mathcal{ALC}$ clause.*

Proof. Equisatisfiability of KB and $\Xi(KB)$ is an easy consequence of Lemma 4.2. Furthermore, the number of recursive invocations of Def, and the number

of new concepts Q are linear in the number of subconcepts of C. Hence, $|\mathsf{Def}(C)|$ is linear in $|C|$, so $|\Xi(KB)|$ is polynomial in $|KB|$. Finally, observe that $\mathsf{Def}(C)$ can contain only concepts of the form D or $\neg Q \sqcup D$, where D is of the form $\bigsqcup L_i$, $\bigsqcap L_i$, $\forall R.L$, or $\exists R.L$, and all $L_{(i)}$ are literal concepts. Hence, for $D = \exists R.L$, $\mathsf{Cls}(D)$ contains clauses of type 1 and 2; for $D = \forall R.L$ a clause of type 3; and for $D = \bigsqcup L_i$ or $D = \bigsqcap L_i$ clauses of type 2. Clauses of types 4 and 5 are obtained by clausifying ABox axioms. □

4.2 Deciding Satisfiability of $\Xi(KB)$ by $\mathcal{R}$

Since $\mathcal{R}$ is a sound and complete calculus, we can use it to check satisfiability of $\Xi(KB)$. To obtain a decision procedure, we just need to ensure that each saturation of $\Xi(KB)$ by $\mathcal{R}$ terminates. The basic principle to achieve this has been outlined by Joyner [16]: we simply ensure that the number of clauses that can be derived in a saturation is bounded.

There are two main reasons why a calculus might derive an infinite number of clauses in a saturation. First, we might keep deriving clauses with deeper and deeper terms. Consider $N_1 = \{C(a), \neg C(x) \vee C(f(x))\}$. If we select $\neg C(x)$ in the second clause and apply resolution, we shall derive $C(f(a))$, $C(f(f(a)))$, and so forth. Second, we might keep deriving clauses with more and more variables. Consider $N_2 = \{\neg R(x,y) \vee \neg R(y,z) \vee R(x,z), C(x) \vee R(x,y) \vee D(y), E(x) \vee R(x,y) \vee F(y)\}$. If we select $\neg R(x,y)$ and $\neg R(y,z)$ in the first clause, by resolution we obtain $C(x) \vee D(y) \vee E(y) \vee F(z) \vee R(x,z)$, which contains more variables than the side premises involved in the inference; further inferences will additionally increase the number of variables.

By choosing these parameters appropriately, it is possible to restrict the resolution inferences, allowing us to establish a bound on the term depth and on the number of variables. Consider again the set of clauses N_1: instead of selecting $\neg C(x)$, if we ensure that $C(f(x)) \succ \neg C(x)$, then the second clause can participate in an inference only on literal $C(f(x))$. Furthermore, $C(f(x))$ and $C(a)$ do not unify, so no inference of $\mathcal{R}$ is applicable to N_1. Hence, N_1 is saturated, and, since it does not contain the empty clause, it is satisfiable. In the following definition we choose the parameters for $\mathcal{R}$ that achieve such an effect on $\mathcal{ALC}$-clauses.

Definition 4.5 Let $\mathcal{R}_{DL}$ denote the calculus $\mathcal{R}$ parameterized as follows:

- In the literal ordering, $R(x, f(x)) \succ \neg C(x)$ and $D(f(x)) \succ \neg C(x)$, for all function symbols f, and predicates R, C, and D.
- The selection function selects in each clause every negative binary literal.

We now prove the following central lemma:

Lemma 4.6 *Each $\mathcal{R}_{DL}$ inference, when applied to $\mathcal{ALC}$-clauses, produces an $\mathcal{ALC}$-clause.*

Table 3
Possible Inferences by $\mathcal{R}_{DL}$ on $\mathcal{ALC}$-clauses

$$\frac{\mathbf{P_1}(x) \vee \mathbf{P_2}(\mathbf{f}(x)) \vee \neg A(g(x)) \quad A(x) \vee \mathbf{P_3}(x)}{\mathbf{P_1}(x) \vee \mathbf{P_2}(\mathbf{f}(x)) \vee \mathbf{P_3}(g(x))} \textbf{ (2+2=2)}$$

$$\frac{\mathbf{P_1}(x) \vee \neg A(x) \quad A(x) \vee \mathbf{P_2}(x)}{\mathbf{P_1}(x) \vee \mathbf{P_2}(x)} \textbf{ (2+2=2)}$$

$$\frac{\mathbf{P_1}(x) \vee \mathbf{P_2}(\mathbf{f}(x)) \vee \neg A(g(x)) \quad A(g(x)) \vee \mathbf{P_3}(\mathbf{h}(x)) \vee \mathbf{P_4}(x)}{\mathbf{P_1}(x) \vee \mathbf{P_2}(\mathbf{f}(x)) \vee \mathbf{P_3}(\mathbf{h}(x)) \vee \mathbf{P_4}(x)} \textbf{ (2+2=2)}$$

$$\frac{\mathbf{P_1}(x) \vee R(x, f(x)) \quad \mathbf{P_2}(x) \vee \neg R(x, y) \vee \mathbf{P_3}(y)}{\mathbf{P_1}(x) \vee \mathbf{P_2}(x) \vee \mathbf{P_3}(f(x))} \textbf{ (1+3=2)}$$

$$\frac{\mathbf{P_1}(\mathbf{a}) \vee \neg A(b) \quad A(x) \vee \mathbf{P_2}(x)}{\mathbf{P_1}(\mathbf{a}) \vee \mathbf{P_2}(b)} \textbf{ (4+2=4)} \qquad \frac{\mathbf{P_1}(\mathbf{a}) \vee \neg A(b) \quad A(b) \vee \mathbf{P_2}(\mathbf{c})}{\mathbf{P_1}(\mathbf{a}) \vee \mathbf{P_2}(\mathbf{c})} \textbf{ (4+4=4)}$$

$$\frac{R(a, b) \quad \mathbf{P_1}(x) \vee \neg R(x, y) \vee \mathbf{P_2}(y)}{\mathbf{P_1}(a) \vee \mathbf{P_2}(b)} \textbf{ (5+3=4)} \qquad \frac{R(a, b) \quad \neg R(a, b)}{\square} \textbf{ (5+5=2)}$$

Proof. The lemma can easily be proved by considering all possible $\mathcal{R}_{DL}$ inferences on all types of $\mathcal{ALC}$-clauses, which are summarized in Table 3. For the sake of brevity, we omit inferences in which literals participating in inferences are complemented. The notation $(n + m = k)$ next to each inference specifies that the inference premises are of types n and m, and the conclusion is of type k. Observe that, due to the requirement on the literal ordering $\succ$, a literal of the form $(\neg)A(x)$ occurring in a clause C can participate in an inference only if C does not contain a literal of the form $(\neg)B(f(x))$ or $R(x, f(x))$. Furthermore, a ground literal $A(a)$ does not unify with a literal $A(f(x))$, and $R(a, b)$ does not unify with $R(x, f(x))$. Hence, ground clauses can participate only in inferences with clauses not containing terms of the form $f(x)$. □

We now show that, for a finite knowledge base KB, the number of $\mathcal{ALC}$-clauses is finite. In fact, this bound can be used to estimate the complexity of the algorithm.

Lemma 4.7 *For an $\mathcal{ALC}$ knowledge base KB, the longest $\mathcal{ALC}$-clause over the signature of $\Xi(KB)$ is polynomial in $|KB|$, and the number of such clauses different up to variable renaming is exponential in $|KB|$.*

Proof. Let c be the number of unary predicates, r the number of binary predicates, f the number of unary function symbols, and i the number of constants in the signature of $\Xi(KB)$. Then, c is linear in $|KB|$, since each concept introduced in $\mathsf{Def}(C)$ corresponds to one nonliteral subconcept of C.

Similarly, f is linear in $|KB|$, since each function symbol is introduced by skolemizing one concept $\exists R.C$. Finally, i is trivially linear in $|KB|$.

Consider now the the maximal $\mathcal{ALC}$-clause C_m. It may contain at most $t = 2+i+f$ terms: a term can be the variable x, the variable y, an individual, or of the form $f(x)$. Hence, C_m contains at most $\ell = 2ct + 2rt^2$ literals (the factor 2 takes into account that each literal can be either positive or negative), which is polynomial in $|KB|$. Each $\mathcal{ALC}$-clause is a subset of C_m, so there are 2^ℓ such clauses; that is, the number of clauses is exponential in $|KB|$. □

We now state the main result of this subsection:

Theorem 4.8 *For an $\mathcal{ALC}$ knowledge base KB, saturating $\Xi(KB)$ by $\mathcal{R}_{DL}$ decides satisfiability of KB and runs in time exponential in $|KB|$.*

Proof. By Lemma 4.7, the number of clauses derivable by $\mathcal{R}_{DL}$ from $\Xi(KB)$ is exponential in $|KB|$. Each inference step can be performed in time polynomial in the size of clauses. Hence, the saturation terminates after performing at most an exponential number of steps. Since $\mathcal{R}_{DL}$ is sound and complete, it decides satisfiability of $\Xi(KB)$, and by Lemma 4.4 of KB as well, in time exponential in $|KB|$. □

5 Translating $\mathcal{ALC}$ to Disjunctive Datalog

Given a decision procedure for checking satisfiability of an $\mathcal{ALC}$ knowledge base KB, it is now easy to obtain the desired reduction to disjunctive datalog. From Table 3 we see that (*i*) a ground clause cannot participate in an inference with a nonground clause containing function symbols, and (*ii*) as soon as one premise in an inference by $\mathcal{R}_{DL}$ is ground, the conclusion is ground as well. Hence, we may perform all inferences among nonground clauses first, after which we may delete all nonground clauses containing function symbols and rewrite clauses as rules. A minor problem arises if thus obtained rules contain unsafe variables; we deal with them by adding a special literal with a predicate HU, which explicitly enumerates the Herbrand universe.

Definition 5.1 Let KB be an $\mathcal{ALC}$ knowledge base. Then, $\Gamma(KB_{\mathcal{T}})$ is the set of clauses obtained by saturating $\Xi(KB_{\mathcal{T}})$ by $\mathcal{R}_{DL}$, and then deleting all clauses containing function symbols. Furthermore, the operator λ maps clauses to clauses as follows:

$$\lambda(C) = C \cup \{\neg HU(x) \mid \text{ for each unsafe variable } x \text{ in } C\ \}$$

For a set of clauses N, $\lambda(N)$ is the set of clauses obtained by applying λ to each element of N. The function-free version of KB is defined as follows:

$$\mathsf{FF}(KB) = \lambda(\Gamma(KB_{\mathcal{T}})) \cup \Xi(KB_{\mathcal{A}}) \cup \{HU(a) \mid \text{ for each individual } a\ \}$$

Finally, a disjunctive datalog program $\mathsf{DD}(KB)$ is the set of rules obtained by moving in each clause from $\mathsf{FF}(KB)$ all positive literals into the rule head, and all negative literals into the rule body.

We now state the properties of $\mathsf{DD}(KB)$:

Theorem 5.2 *For an $\mathcal{ALC}$ knowledge base KB, the following claims hold:*

(i) *KB is unsatisfiable if and only if $\mathsf{DD}(KB)$ is unsatisfiable.*

(ii) *$KB \models \alpha$ if and only if $\mathsf{DD}(KB) \models_c \alpha$, where α is of the form $A(a)$ or $R(a, b)$, and A is an atomic concept.*

(iii) *$KB \models C(a)$ for a nonliteral concept C if and only if, for Q a new atomic concept, $\mathsf{DD}(KB \cup \{C \sqsubseteq Q\}) \models_c Q(a)$.*

(iv) *The number of literals in each rule in $\mathsf{DD}(KB)$ is at most polynomial, the number of rules in $\mathsf{DD}(KB)$ is at most exponential, and $\mathsf{DD}(KB)$ can be computed in time exponential in $|KB|$.*

Proof. (Claim i) Table 3 shows that each inference with at least one ground premise always produces a ground conclusion. Hence, in saturating $\Xi(KB)$ by $\mathcal{R}_{DL}$, all inferences among nonground clauses can be performed first. Furthermore, Table 3 also shows that ground clauses can participate in inferences only with clauses not containing function symbols. Hence, after performing all inferences among nonground clauses of $\Xi(KB)$, one may delete all clauses containing terms of the form $f(x)$.

By Definition 4.3, $\Xi(KB_{\mathcal{T}})$ is exactly the set of nonground clauses of $\Xi(KB)$, so $\Gamma(KB_{\mathcal{T}})$ is exactly the set of clauses obtained by saturating the nonground part of $\Xi(KB)$, and deleting the clauses containing function symbols. Furthermore, it is easy to see that $\Gamma = \Gamma(KB_{\mathcal{T}}) \cup \Xi(KB_{\mathcal{A}})$ is satisfiable if and only if $\mathsf{FF}(KB)$ is satisfiable. Namely, both Γ and $\mathsf{FF}(KB)$ are function-free sets of clauses, whose sets of ground instances differ only on clauses containing unsafe variables. However, for a clause $C \in \Gamma$ containing an unsafe variable x and a ground substitution $\tau = \{x \mapsto a\}$, $\mathsf{FF}(KB)$ contains clauses $C \vee \neg HU(x)$ and $HU(a)$, which together imply $C\tau$. Hence, the ground instances of $\mathsf{FF}(KB)$ imply all ground instances of Γ, so, if $\mathsf{FF}(KB)$ is satisfiable, Γ is satisfiable as well. Furthermore, since HU is a new predicate not occurring in Γ, each model of Γ can easily be extended to a model of $\mathsf{FF}(KB)$. Hence, we conclude that Γ and $\mathsf{FF}(KB)$ are equisatisfiable.

Finally, $\mathsf{DD}(KB)$ is a positive datalog program whose rules are syntactic variants of the clauses from $\mathsf{FF}(KB)$. Hence, $\mathsf{DD}(KB)$ is satisfiable if and only if $\mathsf{FF}(KB)$ is satisfiable.

(Claim ii) Observe that $KB \models \alpha$ if and only if $KB \cup \{\neg\alpha\}$ is unsatisfiable. The latter is the case if and only if $\mathsf{DD}(KB \cup \{\leftarrow \alpha\}) = \mathsf{DD}(KB) \cup \{\leftarrow \alpha\}$ is unsatisfiable, which is the case if and only if $\mathsf{DD}(KB) \models_c \alpha$.

(Claim iii) Follows in the same manner as Claim ii.

(Claim iv) Follows immediately from Lemma 4.7. □

6 Examples

We now present several rather simple examples, which point out important properties of the reduction algorithm.

Readers familiar with more common DL reasoning algorithms might wonder how are role successors represented in a datalog program. Consider the knowledge base $KB_1 = \{C \sqsubseteq \exists R.D\}$; the corresponding set of clauses is $\Xi(KB_1) = \{\neg C(x) \vee R(x, f(x)), \neg C(x) \vee D(f(x))\}$. Now $\Xi(KB_1)$ is already saturated by $\mathcal{R}_{DL}$, so after removing all clauses containing function symbols, we get $\mathsf{DD}(KB_1) = \emptyset$. This may seem quite confusing: KB_1 implies the existence of at least one R-successor for each member of C, whereas $\mathsf{DD}(KB_1)$ does not reflect that. However, Theorem 5.2 is not invalidated. Namely, the individuals introduced by the existential quantifier in KB_1 are unnamed, so they cannot be used in queries. Hence, for an arbitrary extensionally reduced ABox $KB_{\mathcal{A}}$, $KB_1 \cup KB_{\mathcal{A}}$ does not imply any new facts of the form $A(a)$ or $R(a, b)$. Also, note that the models of $\mathsf{DD}(KB_1)$ are completely unrelated to the models of KB_1. Intuitively, this is so because the reduction algorithm is based on a proof-theoretic correspondence between refutations in $\Xi(KB)$ and $\mathsf{DD}(KB)$. The models of KB and $\mathsf{DD}(KB)$ coincide only on positive ground facts, whereas for unnamed individuals they are completely unrelated.

In order to be able to draw additional consequences from KB_1, we must extend it with additional axioms. For example, $KB_2 = KB_1 \cup \{D \sqsubseteq \bot\}$ yields $\Xi(KB_2) = \Xi(KB_1) \cup \{\neg D(x)\}$. Saturation of $\Xi(KB_2)$ produces one additional clause $\neg C(x)$, so we get $\mathsf{DD}(KB_2) = \{\leftarrow C(x), \leftarrow D(x)\}$. This shows that the reduction is not modular: for arbitrary knowledge bases KB_1 and KB_2, $\mathsf{DD}(KB_1) \cup \mathsf{DD}(KB_2)$ is not necessarily equal to $\mathsf{DD}(KB_1 \cup KB_2)$.

The key step in the reduction is the saturation of $\Xi(KB_{\mathcal{T}})$ by $\mathcal{R}_{DL}$. It computes all relevant nonground consequences of $\Xi(KB_{\mathcal{T}})$, ensuring that subsequent removal of clauses containing function symbols does not change the set of ground consequences. One may think of $\mathcal{R}_{DL}$ as producing shortcut clauses, which derive facts about objects without explicitly expanding the successors of each object. Consider $KB_3 = \{A \sqsubseteq \exists R.B,\ B \sqsubseteq C,\ \exists R.C \sqsubseteq D\}$, producing the following clauses in $\Xi(KB_3)$:

$$\neg A(x) \vee R(x, f(x)) \tag{5}$$

$$\neg A(x) \vee B(f(x)) \tag{6}$$

$$\neg B(x) \vee C(x) \tag{7}$$

$$D(x) \vee \neg R(x, y) \vee \neg C(y) \tag{8}$$

Assuming a literal ordering such that $D(x) \succ C(x) \succ B(x) \succ A(x)$, by saturating $\Xi(KB_3)$ we obtain the following new clauses (the notation $R(xx; yy)$ means that a clause is derived by resolving clauses xx and yy):

$$\neg A(x) \vee D(x) \vee \neg C(f(x)) \quad \text{R(5;8)} \tag{9}$$
$$\neg A(x) \vee D(x) \vee \neg B(f(x)) \quad \text{R(9;7)} \tag{10}$$
$$\neg A(x) \vee D(x) \quad \text{R(10;6)} \tag{11}$$

Eliminating clauses with function symbols yields $DD(KB_3)$ as follows:

$$C(x) \leftarrow B(x) \tag{12}$$
$$D(x) \leftarrow R(x, y), C(y) \tag{13}$$
$$D(x) \leftarrow A(x) \tag{14}$$

It is instructive to consider the role of each rule in $\mathsf{DD}(KB_3)$. Whereas the axiom $B \sqsubseteq C$ in KB_3 is applicable to all individuals in a model, the rule (12) is applicable only to named individuals. The relationship between $\exists R.C \sqsubseteq D$ and (13) is analogous. However, (12) and (13) derive consequences only about named individuals, so $\mathsf{DD}(KB_3)$ contains the rule (14), which is produced in the saturation of $\Xi(KB_{\mathcal{T}})$ by $\mathcal{R}_{DL}$. This rule can be thought of as a shortcut: instead of introducing for each x in A an R-successor y in B, propagating y to C, and then concluding that x is in D, (14) derives that all members of A are members of D in one step, thus ensuring that $\mathsf{DD}(KB_3)$ and KB_3 entail the same set of ground facts.

Finally, we take KB_4 to be the knowledge base introduced in Section 3. The concept $\exists R.\exists R.A$ contains a nonatomic subconcept $\exists R.A$, so we apply structural transformation, and replace $\exists R.A$ with a new atomic concept Q. This yields the following set of clauses $\Xi(KB_4)$:

$$\neg A(x) \vee R(x, f(x)) \tag{15}$$
$$\neg A(x) \vee A(f(x)) \tag{16}$$
$$Q(x) \vee \neg R(x, y) \vee \neg A(y) \tag{17}$$
$$B(x) \vee \neg R(x, y) \vee \neg Q(y) \tag{18}$$

Assuming a literal ordering where $Q(x) \succ B(x) \succ A(x)$, a saturation of $\Xi(KB_4)$ by $\mathcal{R}_{DL}$ produces the following new clauses:

$$\neg A(x) \vee Q(x) \vee \neg A(f(x)) \quad \text{R(15;17)} \tag{19}$$
$$\neg A(x) \vee Q(x) \quad \text{R(19;16)} \tag{20}$$
$$\neg A(x) \vee B(x) \vee \neg Q(f(x)) \quad \text{R(15;18)} \tag{21}$$
$$\neg A(x) \vee B(x) \vee \neg A(f(x)) \quad \text{R(21;20)} \tag{22}$$
$$\neg A(x) \vee B(x) \quad \text{R(22;16)} \tag{23}$$

Eliminating clauses with function symbols yields $\mathsf{DD}(KB)$ as follows:

$$Q(x) \leftarrow R(x,y), A(y) \tag{24}$$
$$B(x) \leftarrow R(x,y), Q(y) \tag{25}$$
$$Q(x) \leftarrow A(x) \tag{26}$$
$$B(x) \leftarrow A(x) \tag{27}$$

The intuitive meaning behind these rules is somewhat obscured because of the predicate Q. However, we are not interested in ground consequences related to Q, and, since Q is a new predicate, it cannot occur in any ABox one might consider in conjunction with KB_4. Hence, it is possible to apply rule unfolding to Q, yielding the following datalog program:

$$B(x) \leftarrow A(x) \tag{28}$$
$$B(x) \leftarrow R(x,y), A(y) \tag{29}$$
$$B(x) \leftarrow R(x,y), R(y,z), A(z) \tag{30}$$

Now the intuition behind these rules can be explained as follows. The rule (28) takes into account that each named individual, which is in A, has a chain of at least two unnamed R-successors. The rule (29) takes into account that a named individual may be explicitly linked through R to another individual in A, which then has at least one unnamed R-successor. Finally, the rule (30) takes into account that a named individual may be explicitly linked through an R-chain of length two to an individual which is in A. Only R-chains of length two are considered, because KB_4 contains the concept $\exists R.\exists R.A$, which effectively checks for an R-chain of length two.

7 Discussion

In this subsection we discuss some aspects of our algorithms that may seem surprising at the first glance.

7.1 Independence of the Reduction from the Query

Theorem 5.2 shows that the program $\mathsf{DD}(KB)$ is independent of the query, as long as the query is a positive atomic concept or a role. Hence, $\mathsf{DD}(KB)$ can be computed once, and can be used to answer any number of atomic queries.

On the contrary, if the query involves a nonatomic concept C (even if C is a negated atomic concept), then query answering must be reduced to entailment of positive ground facts, by introducing a new name Q, and by adding the axiom $C \sqsubseteq Q$ to the TBox. Obviously, $\mathsf{DD}(KB \cup \{C \sqsubseteq Q\})$ depends on the query concept C. Intuitively, a complex concept C used in the query introduces a kind of terminological knowledge, so the reduction is not independent from complex queries.

7.2 Complexity

The complexity of cautious query answering in a nonground disjunctive datalog program P is in co-NEXPTIME^{NP}[7]. Since the number of rules in $|P|$ can be exponential in $|KB|$, one might think that reducing DLs to disjunctive datalog significantly increases the complexity of reasoning. However, we show that this is not the case. Let P^g_{DL} be a program obtained by replacing in $\mathsf{DD}(KB)$ all variables with individuals in all possible ways. The number of variables in a rule from $\mathsf{DD}(KB)$ is at most two, so $|P^g_{DL}|$ is exponential in $|KB|$. Furthermore, the arity of predicates in $\mathsf{DD}(KB)$ is at most two, so the number of ground atoms in P^g_{DL} is polynomial in $|KB|$. Hence, an interpretation I_{DL} for P^g_{DL} can be guessed in time which is polynomial in $|KB|$ by choosing a subset of the ground atoms; furthermore, all such interpretations can be examined in time which is exponential in $|KB|$. Checking if I_{DL} is a model of P^g_{DL} can be performed in time which is exponential in $|KB|$. These two steps combined give us the EXPTIME complexity for satisfiability checking, and also for unsatisfiability checking and query answering. To summarize, even though $|\mathsf{DD}(KB)|$ is exponential in $|KB|$, query answering can be performed in exponential time because (*i*) the length of rules in $\mathsf{DD}(KB)$ is polynomial in $|KB|$, (*ii*) the arity of predicates is bounded, and (*iii*) checking minimality of an interpretation is not necessary for positive queries.

7.3 Minimal vs. Arbitrary Models

Disjunctive datalog programs are usually interpreted under *minimal model* semantics: $P \models_c \alpha$ means that α is true in all minimal models of a program P, where minimality is defined w.r.t. set inclusion. However, description logics assume the standard first-order semantics: $KB \models \alpha$ means that α is true in all models of KB. By minimal models, disjunctive datalog implements a kind of *closed-world* semantics; on the contrary, description logics implement *open-world* semantics. Because these two semantics are quite different, our result may seem surprising.

For P a positive datalog program and α a positive ground atom, $P \models \alpha$ if and only if $P \models_c \alpha$. Namely, if α is true in each model of P, it is true in each minimal model of P as well, and vice versa. Therefore, for entailment of positive ground atoms, it is not important whether the semantics of P is defined w.r.t. minimal or w.r.t. general first-order models.

For α a negative ground atom, the difference between minimal model semantics and first-order semantics is relevant. For example, for $\alpha = \neg A(b)$ and $P = \{A(a)\}$, it is clear that $P \not\models \alpha$. Namely, $\neg A(b)$ is not explicitly derivable from the facts in P: $M_1 = \{A(a), A(b)\}$ is a first-order model of P, and α is false in M_1. However, P has exactly one minimal model $M_2 = \{A(a)\}$, and $\neg A(b)$ is obviously true in M_2, so $P \models_c \alpha$.

Subsumption also depends on the type of chosen semantics. For example, let $\alpha = \forall x : [C(x) \rightarrow D(x)]$ and $P = \{C(a), D(a)\}$. Then, $P \not\models \alpha$: just

consider a first-order model $M_1 = \{C(a), D(a), C(b)\}$ of P, in which α is false. However, the only minimal model of P is $M_2 = \{C(a), D(a)\}$, and α is true in M_2, so $P \models_c \alpha$. The distinction between minimal models and general first-order models fundamentally changes the computational properties of concept subsumption: equivalence of general programs under minimal model semantics is undecidable [20], whereas, under first-order semantics, it is decidable and can be reduced to satisfiability checking using standard transformations.

To summarize, the difference between first-order and minimal model semantics is not relevant for answering positive queries in positive datalog programs; however, it is relevant for queries which involve negation or for concept subsumption. Theorem 5.2 reflects this: it says that all positive ground facts entailed by KB are contained in each minimal model of $\mathsf{DD}(KB)$ and vice versa, but does not make any other stronger statements.

It would therefore be incorrect to check whether $KB \models \neg A(a)$ by checking if $\mathsf{DD}(KB) \models_c \neg A(a)$. To apply our algorithm, we must reduce the problem to entailment of positive ground facts: for a new concept $NotA$, $KB \models \neg A(a)$ if and only if $\mathsf{DD}(KB \cup \{\neg A \sqsubseteq NotA\}) \models_c NotA(a)$.

Similarly, it would be incorrect to check whether $KB \models C \sqsubseteq D$ by checking whether $\mathsf{DD}(KB) \models_c \forall x : [C(x) \rightarrow D(x)]$. We must again reduce the problem to entailment of positive ground facts. If C and D are atomic concepts, by well-known transformations we may show that $KB \models C \sqsubseteq D$ if and only if $\mathsf{DD}(KB) \cup \{C(\iota)\} \models_c D(\iota)$, where ι is a new individual not occurring in KB.

7.4 Descriptive vs. Minimal-Model Semantics

Our reduction preserves the so-called descriptive semantics. Namely, Nebel showed that knowledge bases containing terminological cycles are not definitorial [17]: for a fixed partial interpretation of atomic concepts, several interpretations of nonatomic concepts may exist. In such a case, it might be reasonable to designate a particular interpretation as the intended one, with least and greatest fixpoint models being the obvious candidates. However, Nebel argues that it is not clear which interpretation best matches the intuition, because choosing either of the fixpoint models has its drawbacks. Consequently, most description logic systems implement the descriptive semantics, which coincides with the one we presented in Section 2.

By Theorem 5.2, our decision procedure implements exactly the descriptive semantics. Namely, $\mathsf{DD}(KB)$ entails exactly those ground facts which are derivable using the resolution decision procedure, and the latter implements the descriptive semantics.

7.5 Unique Name Assumption

The semantics of $\mathcal{ALC}$ and $\mathcal{SHIQ}$ does not require *unique name assumption* (UNA)—that is, that different symbols are interpreted as different objects; however, the semantics of disjunctive datalog does require it. Therefore, it

may seem surprising that a logic without UNA can be embedded into a logic which strictly requires it.

To understand why our algorithms are correct, note that UNA can be used to derive new consequences only if the theory employs equality. $\mathcal{ALC}$ provides neither number restrictions, nor explicit individual equality statements, so it actually does not require equality reasoning. Given a knowledge base KB, to enforce UNA we may append an axiom $a_i \not\approx a_j$ for each pair of different individuals a_i and a_j. However, since KB does not contain the equality predicate, these inequality axioms do not participate in any inference with $\mathcal{ALC}$-clauses, so they are not needed in the first place. A model-theoretic explanation is given by the following claim, which holds for any logic without equality: for each model I, in which distinct constants are interpreted by the same objects, there is a model I' in which all constants are interpreted by distinct objects. To summarize, in the case of $\mathcal{ALC}$, we simply do not care whether either KB or $\mathsf{DD}(KB)$ employs UNA, as this does not change the entailed set of facts.

The situation changes slightly for logics such as $\mathcal{SHIQ}$, which do require equality reasoning. Consider $KB = \{\top \sqsubseteq \leq 1\, R,\ R(a,b),\ R(a,c)\}$. Without UNA, KB is satisfiable, and $KB \models b \approx c$. Namely, the first axiom requires R to be functional, so b and c must be interpreted as the same object. On the contrary, KB is unsatisfiable with UNA.

A disjunctive datalog program $\mathsf{DD}(KB)$, which corresponds to KB, contains the rules (31)–(33). Note that $\mathsf{DD}(KB)$ is now a disjunctive datalog program with equality, which is quite different from the type of equality usually considered in disjunctive datalog.

$$y_1 \approx y_2 \leftarrow R(x, y_1), R(x, y_2) \tag{31}$$

$$R(a, b) \tag{32}$$

$$R(a, c) \tag{33}$$

In [7] the authors consider disjunctive datalog in which negated equality atoms can occur in rule bodies. Such programs are interpreted under UNA, so an inequality atom $\neg(x \approx y)$ actually checks whether x and y are bound to syntactically different individuals.

In our case, however, we allow equality in the rule heads as well. Thus, we can actually *derive* two individuals to be equal. Such inferences are not directly supported in disjunctive datalog; however, they can be simulated using the well-known encoding from [8], in which the $\approx$ is treated as an ordinary predicate, with required properties axiomatized explicitly. A disjunctive datalog program P with equality is thus transformed into a disjunctive datalog program $P_{\approx}$ without equality, by stating that $\approx$ is reflexive, symmetric, and transitive, and by appending the following replacement rule, instantiated for each distinct predicate R and position i in it:

$$R(x_1, \ldots, y_i, \ldots, x_n) \leftarrow R(x_1, \ldots, x_i, \ldots, x_n), x_i \approx y_i$$

Since $P_{\approx}$ is now a disjunctive datalog program without equality, we may interpret it either with or without UNA, just as we discussed previously. In each Herbrand model $I_{\approx}$ of $P_{\approx}$, the rules in $P_{\approx} \setminus P$ ensure that $\approx$ is interpreted as a congruence relation, so we can always transform $I_{\approx}$ to an interpretation I by replacing each individual a with the equivalence class of $\approx$ to which a belongs. It is well known that thus obtained I is a model of P.

7.6 The Size of $\mathsf{DD}(KB)$

By Theorem 5.2, the size of the rules of $|\mathsf{DD}(KB)|$ can be exponential in $|KB|$, which may seem discouraging in practice. We address this issue in two ways.

On the theoretical side, in [15] we showed that this blowup is only in the size of the TBox of KB. Hence, a sufficiently large ABox can actually dominate the size of the rules, which leads to novel *data complexity* results: in general, checking satisfiability of KB is NP-complete in $|KB_{\mathcal{A}}|$; furthermore, we identified a new fragment of $\mathcal{SHIQ}$ for which satisfiability checking is even polynomial in $|KB_{\mathcal{A}}|$.

On the practical side, in [13] we presented an important optimization which allows removing many rules from $|\mathsf{DD}(KB)|$ without jeopardizing completeness. Intuitively, the saturation of $\Xi(KB)$ introduces clauses which are entailed by other clauses. Such clauses are needed to derive all relevant clauses without function symbols; however, once the saturation has finished, they may be deleted. We have seen that this drastically reduces the number of rules in $\mathsf{DD}(KB)$ in practice.

8 Conclusion

In this paper, we present a version of our algorithm for reducing a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base to a disjunctive datalog program [13,12,14], scaled down to the basic description logic $\mathcal{ALC}$. This simplified algorithm concisely conveys our basic idea, and may help to understand the general algorithms more easily. The reduction technique allows to perform DL reasoning using techniques developed for deductive databses, such as join order optimization or the magic sets transformation. The latter has been shown to dramatically improve the evaluation of disjunctive datalog programs, as it reduces the number of models of the disjunctive program.

We have implemented our algorithm in a new DL reasoner KAON2 [1], and the results of a performance evaluation are available on the project Web site. For knowledge bases with a relatively small TBox, but a large ABox, we have observed performance improvements over state-of-the-art tableau reasoners of one or more orders of magnitude. For TBox reasoning, our system is not able to solve all problems that tableau provers can; however, it is still able to classify certain nontrivial ontologies.

[1] `http://kaon2.semanticweb.org/`

For our future work, we see two main challenges. On the practical side, the performance of the system might be further improved by applying heuristics based on data statistics. On the theoretical side, we shall investigate possibilities of extending DLs with some kind of nonmonotonic reasoning.

Acknowledgment

This work has been carried out in cooperation with Ullrich Hustadt and Ulrike Sattler.

References

[1] Abiteboul, S., R. Hull and V. Vianu, "Foundations of Databases," Addison Wesley, 1995.

[2] Bachmair, L. and H. Ganzinger, *Resolution Theorem Proving*, , **I**, Elsevier Science, 2001 pp. 19–99.

[3] Bachmair, L., H. Ganzinger, C. Lynch and W. Snyder, *Basic Paramodulation*, Information and Computation **121** (1995), pp. 172–192.

[4] Beeri, C. and R. Ramakrishnan, *On the power of magic*, in: *Proc. PODS '87* (1987), pp. 269–283.

[5] Borgida, A., *On the Relative Expressiveness of Description Logics and Predicate Logics*, Artificial Intelligence **82** (1996), pp. 353–367.

[6] Cumbo, C., W. Faber, G. Greco and N. Leone, *Enhancing the Magic-Set Method for Disjunctive Datalog Programs*, in: B. Demoen and V. Lifschitz, editors, *Proc. ICLP 2004*, LNCS **3132** (2004), pp. 371–385.

[7] Eiter, T., G. Gottlob and H. Mannila, *Disjunctive Datalog*, ACM Transactions on Database Systems **22** (1997), pp. 364–418.

[8] Fitting, M., "First-Order Logic and Automated Theorem Proving, 2nd Edition," Texts in Computer Science, Springer, 1996.

[9] Haarslev, V. and R. Möller, *Optimization Strategies for Instance Retrieval*, in: I. Horrocks, S. Tessaris and J. Z. Pan, editors, *Proc. DL 2002*, CEUR Workshop Proceedings **53**, Toulouse, France, 2002.

[10] Horrocks, I., "Optimising Tableaux Decision Procedures for Description Logics," Ph.D. thesis, University of Manchester, UK (1997).

[11] Horrocks, I., U. Sattler and S. Tobies, *Practical Reasoning for Very Expressive Description Logics*, Logic Journal of the IGPL **8** (2000), pp. 239–263.

[12] Hustadt, U., B. Motik and U. Sattler, *Reasoning in Description Logics with a Concrete Domain in the Framework of Resolution*, in: *Proc. ECAI 2004* (2004), pp. 353–357.

[13] Hustadt, U., B. Motik and U. Sattler, *Reducing $\mathcal{SHIQ}^-$ Description Logic to Disjunctive Datalog Programs*, in: *Proc. KR 2004* (2004), pp. 152–162.

[14] Hustadt, U., B. Motik and U. Sattler, *A Decomposition Rule for Decision Procedures by Resolution-based Calculi*, in: *Proc. LPAR 2004*, LNAI **3452** (2005), pp. 21–35.

[15] Hustadt, U., B. Motik and U. Sattler, *Data Complexity of Reasoning in Very Expressive Description Logics*, in: *Proc. IJCAI 2005* (2005), pp. 466–471.

[16] Jr., W. H. J., *Resolution Strategies as Decision Procedures*, Journal of the ACM **23** (1976), pp. 398–417.

[17] Nebel, B., *Terminological Cycles: Semantics and Computational Properties*, in: J. F. Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, Morgan Kaufmann Publishers, San Mateo, CA, USA, 1991 pp. 331–361.

[18] Nieuwenhuis, R. and A. Rubio, *Theorem Proving with Ordering and Equality Constrained Clauses*, Journal of Symbolic Computation **19** (1995), pp. 312–351.

[19] Plaisted, D. A. and S. Greenbaum, *A Structure-Preserving Clause Form Translation*, Journal of Symbolic Logic and Computation **2** (1986), pp. 293–304.

[20] Shmueli, O., *Equivalence of DATALOG Queries is Undecidable*, Journal of Logic Programming **15** (1993), pp. 231–241.

Theory and practice of theorem proving for monodic first-order temporal logic

Boris Konev

Department of Computer Science, University of Liverpool, UK
B.Konev@csc.liv.ac.uk

Temporal logics have long been recognised as introducing appropriate languages for specifying a wide range of important computational properties in computer science and artificial intelligence [5]. However, until recently, the practical use of temporal logics has largely been restricted to propositional temporal logics. First-order temporal logic has generally been avoided as no complete proof system can exist for this logic. A breakthrough by Hodkinson, Wolter, and Zakharyaschev [8] showed that a specific fragment of first-order temporal logic, called the *monodic* fragment, or *monodic* first-order temporal logic, has the completeness property. Following the definition of the monodic fragment, the work on analysing and extending this fragment has continued rapidly, and holds great promise for increasing the power of logic-based formal methods.

In this talk we give an overview of a particular proof method for monodic FOTL, called *monodic temporal resolution*, which is based on our work on clausal temporal resolution over a number of years [6, 7, 1, 3, 4, 2, 11]. The clausal resolution technique has been shown to be one of the most effective proof techniques for propositional temporal logics [9], and we have every reason to believe that it will be as least as successful in the case of FOTL. This calculus forms the basis of the temporal monodic theorem prover **TeMP** presented in [10]. We also consider case studies and applications.

References

1. A. Degtyarev and M. Fisher. Towards first-order temporal resolution. In *Proceedings of KI 2001*, vol. 2174 of *LNCS*, pp. 18–32. Springer, 2001.
2. A. Degtyarev, M. Fisher, and B. Konev. Monodic temporal resolution. *ACM Transactions on Computational Logic*. To appear.
3. A. Degtyarev, M. Fisher, and B. Konev. A simplified clausal resolution procedure for propositional linear-time temporal logic. In *Tableaux 2002, Proceedings*, vol. 2381 of *Lecture Notes in Computer Science*, pp. 85–99. Springer, 2002.
4. A. Degtyarev, M. Fisher, and B. Konev. Monodic temporal resolution. In *Proc. CADE-19*, vol. 2741 of *LNAI*, pp. 397–411. Springer, 2003.
5. E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 16, pp. 997–1072. Elsevier, 1990.
6. M. Fisher. A resolution method for temporal logic. In J. Myopoulos and R. Reiter, editors, *Proceedings of IJCAI'91*, pp. 99–104. Morgan Kaufman, 1991.
7. M. Fisher, C. Dixon, and M. Peim. Clausal temporal resolution. *ACM Transactions on Computational Logic*, 2(1):12–56, 2001.

8. I. Hodkinson, F. Wolter, and M. Zakharyaschev. Decidable fragments of first-order temporal logics. *Annals of Pure and Applied Logic*, 106:85–134, 2000.
9. U. Hustadt and B. Konev. TRP++ 2.0: A temporal resolution prover. In *Proc. CADE-19*, vol. 2741 of *LNAI*, pp. 274–278. Springer, 2003.
10. U. Hustadt, B. Konev, A. Riazanov, and A. Voronkov. **TeMP**: A temporal monodic prover. In *Proc. IJCAR 2004*, vol. 3097 of *LNAI*, pp. 326–330. Springer, 2004.
11. B. Konev, A. Degtyarev, C. Dixon, M. Fisher, and U. Hustadt. Towards the implementation of first-order temporal resolution: the expanding domain case. In *Proc. TIME-ICTL 2003*, pp. 72–82. IEEE, 2003.

The Typechecking Problem for XML Transformations: Methods and Formal Models

Wim Martens

Hasselt University and
Transnational University of Limburg,
Agoralaan, Gebouw D
B-3590 Diepenbeek, Belgium
wim.martens@uhasselt.be

Abstract

The typechecking problem for XML to XML transformations has recently attracted quite some attention in the literature. Given an input schema, an output schema, and an XML to XML transformation, the typechecking problem asks whether the output of the transformation is always conform to the output schema for input documents satisfying the input schema. The present paper takes a detailed look on the two main ingredients of the typechecking problem: formal models of the XML schemas and the XML to XML transformations. Finally, we give an overview of several formal models and techniques that can be used to tackle the typechecking problem.

1 Introduction

XML is a data format in which documents are essentially structured as labeled, ordered trees. It is intended to be a common data format for a very wide range of applications, and aims at being able to model the most diverse forms of data in an intuitive way. Due to this flexibility, XML has become increasingly popular in the last few years, and at the moment, it is quite safe to say that it has become the standard format for data exchange format on the Web.

However, for many applications, it is important to have restrictions on the structure of XML documents. Consider, for instance, automatic processing of XML documents, where prior knowledge of the document structure can greatly facilitate the implementation of the algorithms; or query optimization, which is in many cases not even possible when the structure of the document is not known in advance. These restrictions are imposed by *XML schemas*, which basically describe sets of XML documents.

On the Web, large user communities usually agree on representing their data using the same XML schema in order to facilitate the exchange of doc-

uments within the community. It does happen quite regularly, however, that documents need to be exchanged between two communities with different schemas. Suppose, for instance, that a certain community A with schema S_A, wants to query an XML database D from a certain community B with schema S_B, and publish the results on the world wide web. In order to answer the incoming query, B applies a certain transformation T to the database D, and sends the resulting document $T(D)$ back to A. Before A can automatically process the document $T(D)$ to publish it, the document has to be dynamically validated against the local schema S_A. If this validation fails, the automated processing of the incoming document is hardly possible. Indeed, validity against the local schema can, for example, be a necessary precondition of the algorithms that process the documents.

When many documents are exchanged between the same two communities using similar transformations, this dynamic validation of incoming documents can be a very time-consuming task. This motivates the need for *static typechecking*: given the schema S_B of the sending community, the schema S_A of the receiving community, and a transformation T, we are asked whether for *every* document D in S_B, $T(D)$ is valid with respect to S_A.

Suppose that a static type checker would decide that for every document D in S_A, we have that $T(D)$ is valid with respect to S_B. Then, B would not anymore have to validate the incoming documents coming from A which are transformed by transformation T. Even better, A can update its document D, and as long as D is valid with respect to S_A, B still does not have to validate the document $T(D)$ coming from A. It is clear that, in such as setting, a static type checker can save an important amount of time.
The static typechecking problem has been investigated in several varieties:

Complete Versus Incomplete Typechecking: Of course, the typechecking problem is undecidable when the transformation language is Turing complete. In this case, our best hope is to approximate the answers to the typechecking problem as well as possible. The usual approach that many typecheckers take, is that they are sound, but not complete. This means that the typechecker only passes transformations that actually typecheck, but it sometimes also unjustly rejects a transformation. The aim is of course to minimize these unjust rejections. We refer to [17] for an excellent tutorial on this area of XML typechecking.

Data Values Versus No Data Values: Sound and complete typechecking for XML transformations which have the ability to compare data values seems to be a very difficult problem. Indeed, even for very restricted cases, the typechecking problem becomes undecidable [1]. However, many XML schemas do not constrain the data values in the documents, and the behavior of many filtering or restructuring transformations does not depend on the actual data values.

In this paper, we focus on sound and complete typechecking algorithms

for abstractions of XML documents in which data values are not considered. The purpose of this paper is twofold. First, we visit the two components of the typechecking problem: the XML schema languages and the XML to XML transformations. We give a detailed overview on formal models for several XML schema languages, such as DTD [2] and XML Schema [23], and discuss some of their properties and alternative characterizations, which are meant to give the reader insight into their exact expressive power. We also take a detailed look at sevaral formalisms that have been used to model XML to XML transformations. Second, we want to to present several useful tools for obtaining complexity/decidability results on the typechecking problem for XML transformations.

The outline of this paper is as follows. In Section 2, we introduce some necessary notation and we formally define the typechecking problem. Section 3 discusses the first main ingredient of the typechecking problem: the schema languages. Among others, we provide formal models for the expressive power of DTDs [2] and XML Schema [23]. We also discuss several alternative characterizations of these formal models which give more insight in their exact expressive power. Finally, we discuss closure properties of these XML schema languages under Boolean operations. The second main ingredient of the typechecking problem is discussed in Section 4: the XML to XML transformations. We treat two formalisms in detail, and give an overview of other important formalisms that have been used in the past. In Section 5, we provide a high-level overview of some techniques that have been used to obtain complexity/decidability results on the typechecking problem. We treat complexity upper bounds as well as complexity lower bounds.

2 Preliminaries

2.1 Trees

In this section we provide the necessary background on trees, automata, and tree transducers. In the remainder of the paper, Σ always denotes a finite alphabet.

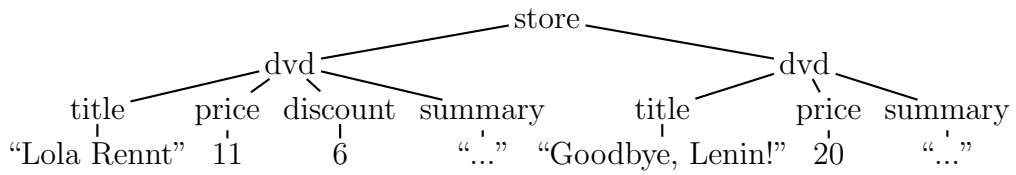
It is common to view XML documents as finite trees with labels from a finite alphabet Σ. There is no limit on the number of children of a node. Figure 1 gives an example of an XML document together with its tree representation. Of course, elements in XML documents can also contain references to nodes. But as XML schema languages usually do not constrain these nor the data values at leaves, it is safe to view schemas as simply defining tree languages over a finite alphabet.

We assume familiarity with DFAs (Deterministic Finite Automata), NFAs (Non-deterministic Finite Automata), and REs (Regular Expressions). Given a DFA, NFA, or RE A, we denote the language accepted by A by $L(A)$.

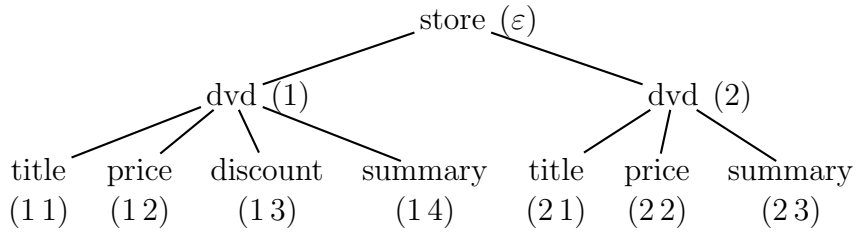
Throughout this paper, Σ always denotes a finite alphabet. The set of

```
<store>
  <dvd>
    <title> "Lola Rennt" </title>
    <price> 11 </price>
    <discount> 6 </discount>
    <summary> ... </summary>
  </dvd>
  <dvd>
    <title> "Goodbye, Lenin!" </title>
    <price> 20 </price>
    <summary> ... </summary>
  </dvd>
</store>
```

(a) An example XML document.



(b) Its tree representation with data values.



(c) The tree of Figure 1(b) without data values. The nodes are annotated next to the labels, between brackets.

Fig. 1. An example of an XML document and its tree representation.

unranked Σ*-trees*, denoted by $\mathcal{T}_\Sigma$, is the smallest set of strings over Σ and the parenthesis symbols "(" and ")" such that, for $a \in \Sigma$ and $w \in \mathcal{T}_\Sigma^*$, $\sigma(w)$ is in $\mathcal{T}_\Sigma$. So, a tree is either ε (empty) or is of the form $a(t_1 \cdots t_n)$ where each t_i is a tree. In the tree $a(t_1 \cdots t_n)$, the subtrees $t_1, \ldots, t_n$ are attached to the root labeled a. We write a rather than $a()$. Note that there is no a priori bound on the number of children of a node in a Σ-tree; such trees are therefore *unranked*. For every $t \in \mathcal{T}_\Sigma$, the *set of nodes* of t, denoted by $\mathrm{Dom}(t)$, is the set defined as follows:

(i) if $t = \varepsilon$, then $\mathrm{Dom}(t) = \emptyset$; and

(ii) if $t = a(t_1 \cdots t_n)$, where each $t_i \in \mathcal{T}_\Sigma$, then $\mathrm{Dom}(t) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{iu \mid$

$u \in \mathrm{Dom}(t_i)\}$.

Figure 1(c) contains a tree in which we annotated the nodes between brackets. Observe that the n child nodes of a node u are always $u1, \ldots, un$, from left to right. The *label* of a node u in the tree $t = a(t_1 \cdots t_n)$, denoted by $\mathrm{lab}^t(u)$, is defined as follows:

(i) if $u = \varepsilon$, then $\mathrm{lab}^t(u) = a$; and

(ii) if $u = iu'$, then $\mathrm{lab}^t(u) = \mathrm{lab}^{t_i}(u')$.

In the sequel, whenever we say tree, we always mean Σ-tree. A *tree language* is a set of trees. In the sequel, we will use the letters $t, t_1, t_2, \ldots$ to denote trees.

2.2 The Typechecking Problem

We define a *tree transformation* to be a mapping $T : \mathcal{T}_\Sigma \to \mathcal{T}_\Sigma$.

Definition 2.1 Let T be a tree transformation and S_{in} and S_{out} be two tree languages. We say that T *typechecks with respect to S_{in} and S_{out}*, if for every $t \in S_{\mathrm{in}}, T(t) \in S_{\mathrm{out}}$.

Definition 2.2 Given S_{in}, S_{out}, and T, the *typechecking problem* consists in deciding whether T typechecks with respect to S_{in} and S_{out}.

Of course, the typechecking problem is undecidable when arbitrary (that is, Turing complete) classes of tree transformations and arbitrary tree languages are used. We therefore restrict these notions in the following sections.

3 Formal Models for XML Schema Languages

We define several common restrictions on tree languages: DTDs, extended DTDs, single-type extended DTDs and restrained competition extended DTDs. Later in this section, we discuss alternate characterizations of these schema languages and their closure properties under Boolean operations.

3.1 The Schema Languages

DTD is the most widely used XML schema language in practice and only imposes *local* restrictions on trees [2]. That is, they only impose restrictions between the label of a node and the labels of its children. We abstract DTDs as extended context-free grammars:

Definition 3.1 A *DTD* is a pair (d, s_d) where d is a function that maps Σ-symbols to regular languages over Σ and $s_d \in \Sigma$ is the start symbol. A tree t *satisfies* d if its root is labeled by s_d and, for every node u with label a, the sequence $a_1 \cdots a_n$ of labels of its children is in $L(d(a))$. By $L(d)$ we denote the set of trees that satisfy d.

We usually denote (d, s_d) by d when the start symbol s_d is understood from the context. We parameterize the definition of DTDs by a class of representations $\mathcal{M}$ of the internal regular string languages such as, for instance, the class of DFAs (Deterministic Finite Automata), NFAs (Non-deterministic Finite Automata), or REs (Regular Expressions). For instance, DTD(DFA) is the class of DTDs, in which the regular languages in the definition of d are represented by DFAs. In examples, we denote the regular languages in the definition of a DTD by regular expressions. For clarity, we write $a \to r$ rather than $d(a) = L(r)$. We also do not list rules of the form $a \to \varepsilon$.

Example 3.2 In this notation, a simple example of a DTD(RE) defining the inventory of a store which sells DVDs is the following:

$$\begin{aligned} \text{store} &\to \text{dvd dvd}^* \\ \text{dvd} &\to \text{title price discount? summary} \end{aligned}$$

where the start symbol is "store". The DTD defines trees of depth 3, where the root is labeled with "store" and has one or more children labeled with "dvd". Every "dvd"-labeled node has three or four children. From left to right, these children are labeled "title", "price", "discount" (which is the optional child), and "summary", respectively. The tree in Figure 1(c) is in the language defined by this DTD.

In this paper, we assume that DTDs do not contain *useless* symbols, unless mentioned otherwise (such as in Propositions 5.1, 5.2, and 5.3). That is, we assume that, for every DTD d and for every $a \in \Sigma$, there exists a tree $t \in L(d)$ and a node $u \in \mathrm{Dom}(t)$ such that $\mathrm{lab}^t(u) = a$.

We now turn to a more expressive formalism for XML schema languages: extended DTDs [20]. [1] The class of tree languages defined by extended DTDs corresponds precisely to the regular (unranked) tree languages [3]. We recall the definition of extended DTDs:

Definition 3.3 ([20]) An *extended DTD* (EDTD) is a 4-tuple $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$, where Σ' is an alphabet of *types*, d is a DTD over Σ', and μ is a mapping from Σ' to Σ. Note that μ can be extended to define a homomorphism on trees. A tree t then *satisfies* an extended DTD if $t = \mu(t')$ for some $t' \in L(d)$. Again, we denote by $L(\mathbf{d})$ the set of trees satisfying $\mathbf{d}$.

As with DTDs, we parametrize the definition of EDTDs by a class of representations $\mathcal{M}$ of the internal regular string languages. For example, EDTD(DFA) is the class of EDTDs $(\Sigma, \Sigma', d, \mu)$ in which d is a DTD(DFA).

For ease of exposition, we always take $\Sigma' = \{a^i \mid 1 \leq i \leq k_a, a \in \Sigma, i \in \mathbb{N}\}$ for some natural numbers k_a, and we set $\mu(a^i) = a$. If a node is labeled by

[1] Papakonstantinou and Vianu used the term *specialized DTD* as types *specialize* Σ-symbols. We prefer the term *extended DTD* as it expresses more clearly that the power of the schemas is amplified.

some $a^i \in \Sigma'$, we call a^i the *type* of the node. For simplicity, we also denote EDTDs in examples in a similar way as DTDs, that is, we write $a^i \to r$ rather than $d(a^i) = L(r)$.

Example 3.4 A simple example of an EDTD(RE) is the following:

$$\begin{aligned} \text{store}^1 &\to (\text{dvd}^1 + \text{dvd}^2)^* \text{dvd}^2 (\text{dvd}^1 + \text{dvd}^2)^* \\ \text{dvd}^1 &\to \text{title}^1\ \text{price}^1\ \text{summary}^1 \\ \text{dvd}^2 &\to \text{title}^1\ \text{price}^1\ \text{discount}^1\ \text{summary}^1 \end{aligned}$$

Here, dvd^1 defines ordinary DVDs, while dvd^2 defines DVDs on sale. The rule for store^1 specifies that there should be at least one DVD on sale.

We consider two restrictions on EDTDs in the remainder of this section, the first of which is the *single-type* EDTD [19]. Single-type EDTDs are an interesting subclass of EDTDs since they correspond to the expressive power of XML Schema [23], which is a widely used XML schema language in practice.

Definition 3.5 ([19]) A *single-type EDTD* (EDTD^{st}) is an EDTD $(\Sigma, \Sigma', d, \mu)$ with the property that for every $a \in \Sigma'$, in the regular expression $d(a)$ no two types b^i and b^j with $i \neq j$ occur.

The EDTD in Example 3.4 is not single-type as both dvd^1 and dvd^2 occur in the rule for store^1.

Example 3.6 A simple example of a single-type EDTD(RE) is the following:

$$\begin{aligned} \text{store}^1 &\to \text{regulars}^1\ \text{discounts}^1 \\ \text{regulars}^1 &\to (\text{dvd}^1)^* \\ \text{discounts}^1 &\to \text{dvd}^2\ (\text{dvd}^2)^* \\ \text{dvd}^1 &\to \text{title}^1\ \text{price}^1\ \text{summary}^1 \\ \text{dvd}^2 &\to \text{title}^1\ \text{price}^1\ \text{discount}^1\ \text{summary}^1 \end{aligned}$$

Although there are still two element definitions dvd^1 and dvd^2, they can only occur in different right hand sides.

The second restricted EDTD that we consider is called the *restrained competition* EDTD. It was originally defined because it is more expressive than a single-type EDTD, while still admitting a fast and simple top-down validation algorithm [19]. Interestingly, it has been shown recently that this restriction corresponds precisely to the semantic notion of *one-pass preorder typeable EDTDs*: it defines the largest class of EDTDs for which every node can be assigned the correct type at the first time it is met in a depth-first left-to-right traversal of the tree [14]. We elaborate on this in Section 3.2.

Definition 3.7 ([19]) Let $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$ be an EDTD. A regular expression r over Σ' *restrains competition* if there are no strings wa^iv and wa^jv' in $L(r)$

with $i \neq j$. We call $\mathbf{d}$ a *restrained competition* EDTD (EDTD^{rc}) if all regular expressions occurring in the definition of d restrain competition.

Example 3.8 An example of a restrained competition EDTD(RE) that is not single-type is given next:

$$\begin{aligned} \text{store}^1 &\rightarrow (\text{dvd}^1)^* \ \text{discounts}^1 \ (\text{dvd}^2)^* \\ \text{dvd}^1 &\rightarrow \text{title}^1 \ \text{price}^1 \ \text{summary}^1 \\ \text{dvd}^2 &\rightarrow \text{title}^1 \ \text{price}^1 \ \text{discount}^1 \ \text{summary}^1 \end{aligned}$$

Note that every single-type EDTD is also restrained competition. The classes of tree languages defined by the grammars introduced above are included as follows: DTD $\subsetneq$ EDTDst $\subsetneq$ EDTD^{rc} $\subsetneq$ EDTD [19]. These strict inclusions can be understood by investigating Examples 3.4–3.8. Using the characterizations in Section 3.2, it is easy to see that every one of these examples defines a tree language that is not expressible in the weaker formalisms.

3.2 Alternative Characterizations of the Schema Languages

The expressive power of the above defined schema languages can be characterized in several manners: types of EDTDs that functionally depend on a part of the tree, DTDs that make use of contextual patterns, and closure properties of tree languages under under subtree exchange [14]. These alternative characterizations make use of the *ancestor-string* and the *ancestor-left-sibling string* of a node, which we define next.

Let t be a tree and v be a node. By ch-str$^t(v)$ we denote the string formed by the children of v, i.e., $\text{lab}^t(v1)\cdots\text{lab}^t(vn)$ if v has n children. By anc-str$^t(v)$ we denote the string formed by the labels on the path from the root to v, i.e., $\text{lab}^t(\varepsilon)\text{lab}^t(i_1)\text{lab}^t(i_1i_2)\cdots\text{lab}^t(i_1i_2\cdots i_k)$ where $v = i_1i_2\cdots i_k$. By l-sib-str(v) we denote the string formed by the labels of the left siblings of v, i.e., $\text{lab}^t(u1)\cdots\text{lab}^t(uk)$ where $v = uk$. By anc-sib-str(v) we denote the string

$$\text{l-sib-str}^t(\varepsilon)\#\text{l-sib-str}^t(i_1)\#\cdots\#\text{l-sib-str}^t(i_1i_2\cdots i_k)$$

formed by concatenating the left-sibling strings of all ancestors starting from the root. Here, we assume that "#" is a special symbol not occurring in Σ.

For a tree t and a node v we denote by preceding$^t(v)$ the tree resulting from t by removing everything below v, all right siblings of v's ancestors and of v, and their respective subtrees. We illustrate the just defined notions in Figure 2.

Definition 3.9 We say that an EDTD $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$ *has ancestor-based types* if there is a (partial) function $f : (\Sigma \cup \{\#\})^* \rightarrow \Sigma'$ such that, for each tree $t \in L(\mathbf{d})$ the following holds:

(1) there is a unique tree $t' \in L(d)$ with $\mu(t') = t$; and

(2) for each node $v \in \text{Dom}(t)$, the label of v in t' is $f(\text{anc-str}^t(v))$.

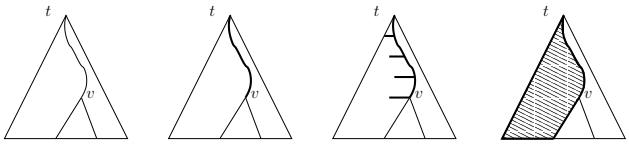


Fig. 2. From left to right: a tree t, the ancestor-string of v in t, the ancestor-sibling-string of v in t and the preceding of v in t.

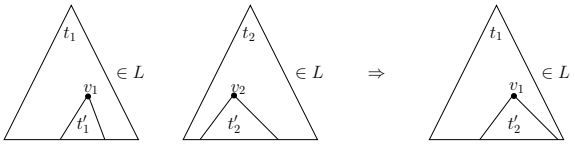


Fig. 3. Label-guarded subtree exchange.

We say that $\mathbf{d}$ *has ancestor-sibling based types* (respectively, *label-based types*) if the same holds with anc-str$^t(v)$ replaced by anc-sib-str$^t(v)$ (respectively, by lab$^t(v)$). We say that $\mathbf{d}$ *has preceding-based types* if there is a (partial) function $f : \mathcal{T}_\Sigma \to \Sigma'$ such that the above statement holds with preceding$^t(v)$ in place of anc-str$^t(v)$.

Definition 3.10 An *ancestor-guarded DTD* $\mathbf{d}$ is a pair (d, s_d) where $s_d \in \Sigma$ is the start symbol as in a DTD. But in contrast to a DTD, d is a finite set of triples (r, a, s), where $a \in \Sigma$ and r and s are regular expressions. A tree t satisfies $\mathbf{d}$ if for every node $v \in \mathrm{Dom}(t)$ the following holds. If lab$^t(v) = a$ then there must be a triple (r, a, s) in d such that anc-str$^t(v)$ matches r and ch-str$^t(v)$ matches s.

An *ancestor-sibling-guarded DTD* is defined in the same way with the difference that r has to be matched by anc-sib-str(v).

Note that it does not make much sense to define a *label-guarded DTD*, as this would simply be a DTD.

In the following definition, we denote by $t_1[u \leftarrow t_2]$ the tree obtained from a tree t_1 by replacing the subtree rooted at $u \in \mathrm{Dom}(t_1)$ by t_2. By subtree$^t(u)$ we denote the subtree of t rooted at u.

Definition 3.11 We say that a tree language L is *closed under ancestor-guarded subtree exchange* if the following holds. Whenever for two trees $t_1, t_2 \in T$ with nodes $u_1 \in \mathrm{Dom}(t_1)$ and $u_2 \in \mathrm{Dom}(t_2)$ it holds that anc-str$^{t_1}(u_1) =$ anc-str$^{t_2}(u_2)$, this implies that $t_1[u_1 \leftarrow \text{subtree}^{t_2}(u_2)] \in T$. We call it *closed under ancestor-sibling-guarded subtree exchange* (respectively, *label-guarded subtree exchange*) if the same holds with anc-sib-str$^{t_1}(u_1) =$ anc-sib-str$^{t_2}(u_2)$ (respectively, lab$^{t_1}(u_1) =$ lab$^{t_2}(u_2)$) as precondition of the implication.

These subtree exchange properties are illustrated in Figures 3, 4, and 5.

The following theorems characterize the expressive power of DTDs, single-type EDTDs, and restrained competition EDTDs. We assume in both theorems that L is a tree language in which every tree has the same root label.

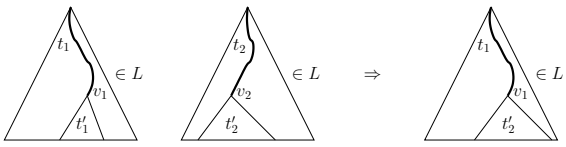


Fig. 4. Ancestor-guarded subtree exchange.

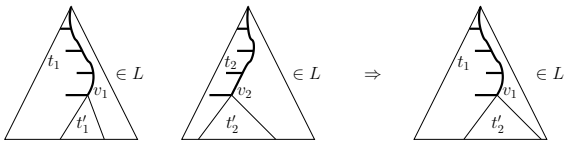


Fig. 5. Ancestor-sibling-guarded subtree exchange.

Theorem 3.12 ([14]) *For a regular tree language L the following are equivalent:*

(a) L is definable by a single-type EDTD;

(b) L is definable by an EDTD with ancestor-based types;

(c) L is closed under ancestor-guarded subtree exchange; and,

(d) L is definable by an ancestor-guarded DTD.

A corresponding theorem also holds for DTDs, by replacing *ancestor* by *label*. The equivalence between (c) and (a) is then already obtained in [20] (and the other equivalences are straightforward).

For restrained competition EDTDs, a similar theorem holds, but it has one extra characterization:

Theorem 3.13 ([14]) *For a regular tree language L the following are equivalent:*

(a) L is definable by a restrained competition EDTD;

(b) L is definable by an EDTD with ancestor-sibling-based types;

(c) L is definable by an EDTD with preceding-based types;

(d) L is closed under ancestor-sibling-guarded subtree exchange; and

(e) L is definable by an ancestor-sibling-guarded DTD.

The extra characterization (c) is quite interesting indeed. It states that an EDTD with ancestor-sibling-based types does not become more expressive when we allow its types to depend on the *entire preceding* of a node. The importance of this characterization becomes apparent in the context of validating XML documents as SAX-streams. Intuitively, the SAX-stream of a tree is obtained from a tree by traversing it in a depth-first left-to-right manner and writing the opening tag `<a>` for every a-labeled node that is met for the first time (when coming from the node's parent) and the closing tag `</a>` for every a-labeled node that is met for the second time in the traversal (when coming from the node's children). Indeed, Theorem 3.13 shows that the class

of restrained competition EDTDs is precisely the class of EDTDs that allow to unambiguously assign a type to a node when its opening tag is met in the SAX-stream (or, in other words, when the node is visited for the first time in the depth-first left-to-right traversal of the tree).

3.3 Closure Properties

It is well-known that unranked regular tree languages (and, hence, EDTDs) are closed under union, difference, complement, and intersection. But does this also hold for DTDs, single-type EDTDs, and restrained competition EDTDs? Most of these questions are investigated by Murata et al. [19,18]. We summarize their results, along with closure properties under complement, in Table 1. Here, we define the complement of a DTD (d, s) to be the language $\{t \in \mathcal{T}_\Sigma \mid \text{lab}^t(\varepsilon) = s\} - L(d)$ to avoid problems with the start symbol. We define the complement of languages defined by EDTDsts and EDTD^{rc}s analogously.

	union	difference	complement	intersection
DTD	not closed	not closed	not closed	closed
EDTDst	not closed	not closed	not closed	closed
EDTD^{rc}	not closed	not closed	not closed	closed
EDTD	closed	closed	closed	closed

Table 1
Closure properties of DTDs, single-type EDTDs, restrained competition EDTDs, and EDTDs.

We give some insights about why these results hold. We start by showing that DTDs, EDTDsts, and EDTD^{rc}s are not closed under complement, which is quite easy using the characterizations of Section 3.2:

Example 3.14 Consider the DTD (d, a) over the alphabet $\Sigma = \{a, b\}$ defined as

$$a \to b^*$$
$$b \to b^*.$$

This DTD defines the set of trees where the symbol a is only allowed to occur at the root. We show that the complement of this DTD is not definable by a EDTD^{rc}, which shows that DTDs, EDTDsts, and EDTD^{rc}s are not closed under complement. Indeed, consider the two trees in Figure 6(a), which are in the complement of the DTD d. By applying ancestor-sibling-guarded subtree exchange on the circled nodes, we obtain the tree $a(b(b)b(b))$, which is not in the complement of d.

As the tree language $\{t \in \mathcal{T}_\Sigma \mid \text{lab}^t(\varepsilon) = a\}$ is definable by a DTD, this example shows that DTDs, EDTDsts, and EDTD^{rc}s are also not closed under difference.

Fig. 6. Counterexample trees for Example 3.14 and Example 3.15.

We can use a similar argument to show that DTDs, EDTDsts, and EDTD^{rc}s are not closed under union:

Example 3.15 Consider the DTD (d_1, a) over alphabet $\{a, b, c\}$ defined as

$$a \to b^* c^*$$

and the DTD (d_2, a) over alphabet $\{a, b, c\}$ defined as

$$a \to b^*$$
$$b \to c.$$

In the union of these DTDs, either every b-child of the root has a c-child, or, none of the b-children of the root has a c-child. We show that this union is not definable by a EDTD^{rc}. Indeed, consider the two trees in Figure 6(b). Note that the left tree is in $L(d_1)$ and the right tree is in $L(d_2)$. However, by applying ancestor-sibling-guarded subtree exchange on the circled nodes, we obtain the tree $a(bb(c))$, which is not in the union of d_1 and d_2.

Finally, the closure of DTDs, EDTDsts, and EDTD^{rc}s under intersection can easily be shown by direct construction, using a straightforward product operation on the characterizations of Theorems 3.12(d) and 3.13(e). An alternative proof can be found in the appendix of [19].

4 Formal Models for XML to XML Transformations

The second main ingredient of the typechecking problem are the tree transformations. In this section, we revisit several formalisms that have been used to model XML to XML transformations.

4.1 Simple Tree Transducers

We first consider a class of tree transducers which was defined in order to investigate settings in which the typechecking problem becomes tractable [10,11]. These tree transducers are not very expressive (in fact, they are the least expressive transducers in this paper), but they are also not meant to model a full-fledged query language. Instead, they are intended to model simple document restructurings that occur often in practice.

For a set Q, denote by $\mathcal{T}_\Sigma(Q)$ the set of Σ-trees where leaf nodes are labeled with elements from $\Sigma \cup Q$ instead of only Σ.

Definition 4.1 A *simple tree transducer* is a 4-tuple $T = (Q, \Sigma, q^0, R)$, where Q is a finite set of states, Σ is the input and output alphabet, $q^0 \in Q$ is the initial state, and R is a finite set of rules of the form $(q, a) \to t$, where $a \in \Sigma$, $q \in Q$, and $t \in \mathcal{T}_\Sigma(Q)$. When $q = q^0$, t is restricted to be either empty, or to have a Σ-symbol as its root label. Simple tree transducers are required to be deterministic: for every pair (q, a), there is at most one rule in R.

The translation defined by $T = (Q, \Sigma, q^0, R)$ on a tree t in state q, denoted by $T^q[t]$, is inductively defined as follows: if $t = \varepsilon$ then $T^q[t] := \varepsilon$; if $t = a(t_1 \cdots t_n)$ and there is a rule $(q, a) \to t' \in R$ then $T^q[t]$ is obtained from t' by replacing every node u in t' labeled with state p by the sequence of trees $T^p[t_1] \cdots T^p[t_n]$. Note that such nodes u can only occur at leaves. So, t' is only extended downwards. If there is no rule $(q, a) \to t' \in R$ then $T^q[t] := \varepsilon$. Finally, the transformation of t by T, denoted by $T(t)$, is defined as $T^{q^0}[t]$.

For simplicity, we have only used *trees* on the right hand sides of rewrite rules in simple tree transducers. Usually, their right hand sides contain *hedges*, which are essentially sequences of trees. For more details, we refer to [10,11].

Example 4.2 We describe a simple tree transducer T that returns, for a "store"-document conforming to the DTD in Example 3.2, the "store"- document in which first the DVD titles occur together with their prices, and further in the document, every DVD title occurs with its summary. Here, we abbreviate "title" by "t", "price" by "p", "discount" by "d", and "summary" by "s".

Let $T = (Q, \Sigma, q_0, R)$ where $Q = \{q_0, q_{\text{tpd}}, q_{\text{ts}}\}$, $\Sigma = \{\text{store}, \text{dvd}, \text{t}, \text{p}, \text{d}, \text{s}\}$, and R contains the rules

$$(q_0, \text{store}) \to \begin{array}{c} \text{store} \\ \diagup \quad \diagdown \\ q_{\text{tpd}} \quad q_{\text{ts}} \end{array}$$

$$\begin{array}{rcl} (q_{\text{tpd}}, \text{dvd}) & \to & q_{\text{tp}} \\ (q_{\text{tpd}}, \text{t}) & \to & \text{t} \\ (q_{\text{tpd}}, \text{p}) & \to & \text{p} \\ (q_{\text{tpd}}, \text{d}) & \to & \text{d} \end{array} \qquad \begin{array}{rcl} (q_{\text{ts}}, \text{dvd}) & \to & q_{\text{ts}} \\ (q_{\text{ts}}, \text{t}) & \to & \text{t} \\ (q_{\text{ts}}, \text{s}) & \to & \text{s} \end{array}$$

Intuitively, q_{tpd} and q_{tp} select the "t", "p" and "d" descendants of the "dvd" node, or the "t" and "s" descendants of the "dvd"-node, respectively.

Example 4.3 In Figure 7 we give the translation of the tree t from Figure 1(c) by the transducer of Example 4.2. For brevity, we again abbreviate "title", "price", "discount", and "summary" by their initial letters, respectively. In order to keep the example simple, we abbreviated sequences of the form $T^q[a_1] \cdots T^q[a_n]$ by $T^q[a_1 \cdots a_n]$.

Simple tree transducers can easily be extended with XPath expressions [11].

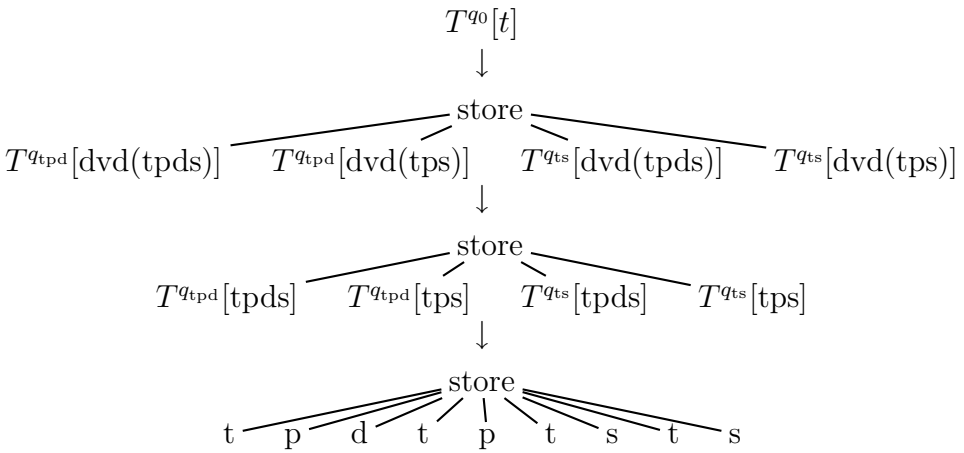


Fig. 7. The translation of the tree in Figure 1(c) by the transducer T of Example 4.2.

In that case, right hand sides of rules in the tree transducer can contain pairs of the form $\langle q, X\rangle$, where X is an XPath expression. The semantics of such a pair is that the transducer continues computation in state q in *all the nodes selected by* X (where the current node is the context node), rather than in *all children of the current node*. For more details, we refer to [11].

4.2 Macro Tree Transducers

We now turn to a much more powerful model of tree transducer for which the typechecking problem has been investigated: the macro tree transducer [5]. The reason why we treat macro tree transducers in this paper, is that they are not only a powerful transformation formalism, but they have also proven to be a useful tool to obtain upper bounds on the complexity of typechecking (cfr. Section 5.1.4).

As opposed to the XML schema languages from Section 3 and the simple tree transducers from Section 4.1, macro tree transducer are defined on *ranked* trees instead of unranked trees. However, it is well-known that unranked trees can be encoded to ranked trees in various ways, which provides a way to define the semantics of macro tree transducers over unranked trees. We will illustrate one such encoding later, when we explain the operation of a macro tree transducer.

Before we define macro tree transducers, we need to introduce some notions on ranked trees. A *ranked alphabet* (or ranked set) is a pair $(\Sigma, \text{rank}_\Sigma)$, where $\text{rank}_\Sigma : \Sigma \to \mathbb{N}$ is a function that maps each symbol a to the number of children that an a-labeled node is allowed to have in a ranked Σ-tree. For $k \geq 0$, we denote the set the set $\{a \in \Sigma \mid \text{rank}_\Sigma(a) = k\}$ by $\Sigma^{(k)}$. We also write $a^{(k)}$ to indicate that that $\text{rank}_\Sigma(a) = k$. For a set S, $\langle \Sigma, S\rangle$ is the ranked set $\Sigma \times S$ with $\text{rank}_{\langle\Sigma,S\rangle}(\langle a, s\rangle) = \text{rank}_\Sigma(a)$ for every $\langle a, s\rangle \in \langle \Sigma, S\rangle$.

Formally, the set $\mathcal{T}^R_\Sigma$ of *ranked* Σ*-trees* is the set of Σ-trees $a(t_1 \cdots t_k)$, where $a \in \Sigma^{(k)}$ and each t_i is a ranked Σ-tree.

We fix a *set of input variables* to be $X = \{x_1, x_2, \ldots\}$ and a *set of output*

variables to be $Y = \{y_1, y_2, \ldots\}$. For $k \geq 0$, $X_k = \{x_1, \ldots, x_k\}$ and $Y_k = \{y_1, \ldots, y_k\}$. We require that $\Sigma \cap X = \Sigma \cap Y = \emptyset$.

Definition 4.4 [[5]] A (non-deterministic) *macro tree transducer (MTT)* is a 5-tuple $M = (Q, \Sigma, \Delta, q_0, R)$, where Q is a ranked alphabet of states, Σ and Δ are the ranked alphabets of input and output symbols, respectively, $q_0 \in Q^{(0)}$ is the initial state, and R is a finite set of rules of the form

$$\langle q, a(x_1, \ldots, x_k)\rangle(y_1, \ldots, y_m) \to \zeta \qquad (*)$$

for $q \in Q^{(m)}$ and $a \in \Sigma^{(k)}$ with $m \geq 0, k \geq 0$. Here, $\zeta \in \mathcal{T}^R_{\langle Q, X_k\rangle \cup \Sigma}(Y_m)$. [2]

A macro tree transducer is called *deterministic* if, for every $q \in Q^{(m)}$ and $a \in \Sigma^{(k)}$, at most one rule of the form $(*)$ exists.

The rules of M can be viewed as term rewriting rules in the obvious way, with the input variables x_i ranging over $\mathcal{T}^R_\Sigma$ and the parameters y_j ranging over $\mathcal{T}^R_\Delta$. Then M induces a derivation relation $\Rightarrow_M$ on $\mathcal{T}^R_{\langle Q, \mathcal{T}^R_\Sigma\rangle \cup \Delta}$ and an input tree $s \in \mathcal{T}^R_\Sigma$ is translated by M into a set of possible output trees $t \in \mathcal{T}^R_\Delta$ with $\langle q_0, s\rangle \Rightarrow^*_M t$.

In the Example 4.5, we illustrate the derivation relation induced by a macro tree transducer in detail.

In order to operate a macro tree transducer over unranked trees, we consider unranked trees as *binary trees over the first-child and next-sibling relation*. That is, the left child of a node in the binary tree is the first child of the node in the unranked tree, and the second child of a node is the next sibling of the node in the unranked tree. We insert the special symbol $\perp$ when no first child or next sibling exists. For instance, the tree of Figure 8(a) then becomes the tree in Figure 8(b).

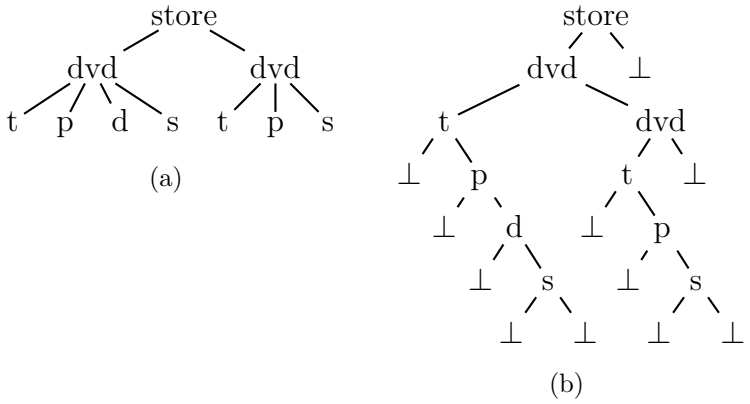


Fig. 8. An unranked tree (Figure 8(a)) as a binary tree over the first-child and next-sibling relation (Figure 8(b)).

[2] Recall that for a set S, we denote by $\mathcal{T}_\Sigma(S)$ the set of Σ-trees where leaf nodes are labeled with elements from $\Sigma \cup S$. Hence, $\mathcal{T}^R_{\langle Q, X_k\rangle \cup \Sigma}(Y_m)$ is the set $\mathcal{T}^R_{\langle Q, X_k\rangle \cup \Sigma \cup Y_m}$ where for every $y \in Y_m$, $\mathrm{rank}_Y(y) = 0$.

Example 4.5 We give an example of a MTT that removes all "dvd" nodes (and their subtrees) that are not on discount from documents satisfying the DTD from Example 3.2. Again, we abbreviate "title", "price", "discount", and "summary" by their initial letters, respectively.

The rules of this MTT are depicted in Figure 9. Intuitively, q_{id} is a state that performs the identity transformation, and $q_{\text{search-d}}$ is a state that searches the subtree for a "d"-labeled node and prunes its left subtree in the output unless the "d"-labeled node is found.

When executed on the tree t in Figure 8(b), we get the derivation in Figure 10. In this derivation, we denoted by t/u the subtree of t rooted at u.

$$\langle q_0, \text{store}(x_1, x_2)\rangle \to \text{store}\big(\langle q_{\text{sel-d}}, x_1\rangle, \bot\big)$$

$$\langle q_{\text{sel-d}}, \text{dvd}(x_1, x_2)\rangle \to \langle q_{\text{search-d}}, x_1\rangle\big(\langle q_{\text{id}}, x_1\rangle, \langle q_{\text{sel-d}}, x_2\rangle\big)$$

$$\langle q_{\text{search-d}}, \text{a}(x_1, x_2)\rangle(y_1, y_2) \to \langle q_{\text{search-d}}, x_2\rangle(y_1, y_2) \quad \text{for } a = \text{t,p}$$

$$\langle q_{\text{search-d}}, \text{d}(x_1, x_2)\rangle(y_1, y_2) \to \text{dvd}(y_1, y_2)$$

$$\langle q_{\text{search-d}}, \text{s}(x_1, x_2)\rangle(y_1, y_2) \to y_2$$

$$\langle q_{\text{id}}, a(x_1, x_2)\rangle \to a\big(\langle q_{\text{id}}, x_1\rangle, \langle q_{\text{id}}, x_2\rangle\big) \quad \text{for } a = \text{t,p,d,s}$$

$$\langle q_{\text{sel-d}}, \bot\rangle \to \bot$$

$$\langle q_{\text{id}}, \bot\rangle \to \bot$$

Fig. 9. Macro tree transducer from Example 4.5.

4.3 Further Tree Transducer Formalisms

4.3.1 k-Pebble Tree Transducers

The research on typechecking for XML transformations was initiated by Milo, Suciu, and Vianu [16]. As a formal model for tree transformations, they defined *k-pebble tree transducers*, which they designed to model many existing XML query languages (without data-value joins), and for which the typechecking problem w.r.t. extended DTDs is decidable.

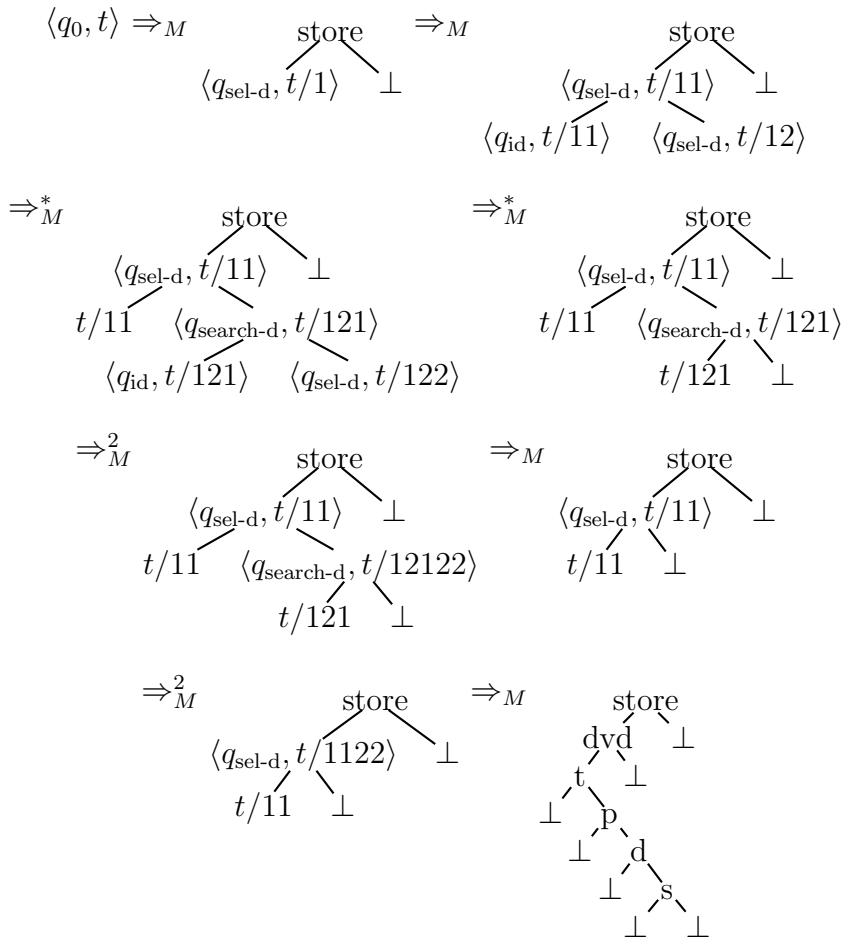


Fig. 10. Derivation of the execution of the MTT in Figure 9 on the tree t in Figure 8(b).

Intuitively, a k-pebble tree transducer is a finite state transducer which uses up to k heads (i.e., *pebbles*) that can be placed on the input. However, the transducer must place and remove the pebbles from the input in a stack-based fashion, and it can only move the highest ranked pebble that is placed on the input. In order to produce output, k-pebble tree transducers are equipped with *output transitions*. Such an output transition writes a symbol to the output, and creates a fresh copy of the k-pebble tree transducer for every child of the current node. These copies then run further in parallel, do not communicate to each other, and inherit the positions of all pebbles on the input.

Suciu presented the functional language *RecQL*, which is meant to characterize precisely the transformations that are expressible by k-pebble tree transducers [25].

Engelfriet and Maneth investigated the connection between macro tree transducers and k-pebble tree transducers [4]. They obtained that the transformation of a k-pebble tree transducer can be carried out by a composition of $k+1$ macro tree transducers. Conversely, they also showed that a macro tree

transducer can be simulated by a composition of four 1-pebble tree transducers (that is, a pebble tree transducer with only one head).

4.3.2 TL-*Transformers*

Recently, Maneth et al. defined TL-transformers, which are a much more powerful version of the simple tree transducers presented in Section 4.1. For instance, TL-transformers are not deterministic, and they can use unary or binary MSO formulas as *match* and *select* patterns, respectively. That is, their programs consist of rules of the form $f(m) \Rightarrow A$, where f is a function name, m is a unary formula that is determine to which nodes the function f is applicable (the *match* pattern), and A is the *action* associated to f. Here, the action A can contain binary MSO formulas to select the nodes in which the program can continue (*select* patterns).

A typechecking algorithm for TL transformations is obtained by decomposing TL transformations as a composition of macro tree transducers (See also Section 5.1.4).

4.3.3 Typechecking with Data Values

Sound and complete typechecking for transformations that have the power to test equality between data values has been investigated by Alon et al. [1]. They defined the query language QL, which defines queries that map trees *with* data values onto trees *without* data values. Queries in QL are tree templates in which each node is labeled with a formula, and a Σ-symbol or a variable. The formulas are existentially quantified conjunctions over *path expressions* and *comparison formulas*. Path expressions are formulas of the form XRY, where X and Y are variables (ranging over nodes in the input tree), and R is a regular expression. Nodes X and Y satisfy such a formula when there exists a path in the input tree from node X to node Y that matches the regular expression R. Comparison formulas are formulas of the form $X = V$ or $X \neq V$. Here, V can either be a variable ranging over nodes in the input tree, or a data value. The output of such a query on an input tree is defined in terms of possible matches of the free variables in the tree template on the input tree. The tree-structure of the output is inferred from the tree-structure of the query itself. For the precise details, we refer to [1].

Alon et al. characterize several fragments for which the typechecking problem for QL transformations is decidable, but they also show that the problem turns undecidable even in very restricted cases [1].

5 Methods for the Typechecking Problem

5.1 Methods for Proving Upper Bounds

In the past, a variety of methods for proving upper and lower bound for the typechecking problem have been devised. In the present section, we survey

some of these methods. With the word "schema", we refer to any of the schema languages presented in Section 3.

5.1.1 Emptiness

The emptiness test of tree automata or EDTDs seems to be a basic building stone of complete typechecking algorithms. Both the type inference and inverse type inference methods discussed in Sections 5.1.2 and 5.1.3, respectively, usually make use of the emptiness test of the schema languages at hand.

Formally, the emptiness test of a schema S is the following:

Emptiness: Given a schema S, is $L(S) = \emptyset$?

A direct reduction to the emptiness test of the schema language in the typechecking problem has been used in [10,11], to show tractability of several fragments of the typechecking problem. Here, it was used that testing emptiness of an EDTD in which the regular languages are represented by NFAs, is PTIME-complete [10].

5.1.2 Type Inference

Type inference might intuitively be the most straightforward technique to obtain a typechecking algorithm. Intuitively, we want to characterize the set of documents that can result from the transformation, when the input is conform to the input schema. Finally, we should test whether this set is contained in the output schema. This idea is used in several incomplete type checkers such as, for instance, in the programming language XDuce [6].

The type inference approach can be summarized more formally as follows:

Type Inference: Given input and output schemas S_{in} and S_{out} and tree transducer T, compute the tree language $L_{\text{out}} = T(L(S_{\text{in}}))$. Then verify whether $L_{\text{out}} \subseteq L(S_{\text{out}})$.

The last inclusion test usually involves computing an automaton A for the language $L_{\text{out}} \cap \overline{L(S_{\text{out}})}$, and testing whether $L(A) = \emptyset$, which is why this approach is usually more involved than the emptiness test. Here, $\overline{L(S_{\text{out}})}$ denotes the complement of $L(S_{\text{out}})$.

The main difficulty in this approach lies in precisely characterizing L_{out}. Indeed, even when S_{in} is a DTD and T is a simple tree transducer, L_{out} can be non-regular. For example, consider the DTD $s \to a^*$ defining S_{in} and the simple tree transducer T with the rules

$$\begin{aligned}(q_0, s) &\to s(q_1 q_2 q_3)\\ (q_1, a) &\to a\\ (q_2, a) &\to b\\ (q_3, a) &\to c.\end{aligned}$$

Then, the language $L_{\text{out}} = \{s(w) \mid w = a^n b^n c^n \text{ for } n \in \mathbb{N}\}$, which is clearly not regular. Note that it is even not possible to approximate L_{out}: there is no smallest regular tree language containing L_{out}. Indeed, for every regular tree

language L that contains L_{out} and for every tree $t \in L_{\text{out}} - L$, $L - \{t\}$ is a better regular approximation for L_{out} than L.

A sound and complete typechecking algorithm based on type inference is presented in [11], to show that arbitrary simple tree transducers can be typechecked w.r.t. DTD(RE$^+$)s in PTIME.[3] However, the main difficulty in this approach was showing the correctness of the algorithm, as it only infers an approximation of the language L_{out}.

5.1.3 Inverse Type Inference

Inverse type inference is inspired on type inference, but here we want to infer the set of input trees t for which $T(t)$ is in the ouput schema (or, *not* in the output schema, depending on the preferred variant).

More formally, inverse type inference can be described as follows:

Inverse Type Inference I: Given input and output schemas S_{in} and S_{out} and tree transducer T, first compute the pre-image $L_{\text{in}} = \{t \in \mathcal{T}_\Sigma \mid T(t) \in L(S_{\text{out}})\}$. Then, test whether $L(S_{\text{in}}) \subseteq L_{\text{in}}$.

Of course, one could also try another variant of inverse type inverence:

Inverse Type Inference II: Given input and output schemas S_{in} and S_{out} and tree transducer T, compute the complement $\overline{L(S_{\text{out}})}$ of $L(S_{\text{out}})$. Then, compute the pre-image of $\overline{L(S_{\text{out}})}$ through T, that is, $L_{\text{in}} = \{t \in \mathcal{T}_\Sigma \mid T(t) \in \overline{L(S_{\text{out}})}\}$. Then verify whether $L(S_{\text{in}}) \cap L_{\text{in}} = \emptyset$.

The preferred variant of the inverse typechecking method depends on the complexities of computing the complement of the output schema, and computing the pre-image of a tree language.

The inverse type inference technique was used for typechecking k-pebble tree transducers [16], macro tree transducers [4], and TL-programs [9]. Even though the transformation languages were quite powerful in all the three cases, it turned out that the inferred set L_{in} is regular when the output schema defines a regular language.

5.1.4 Compositions of Macro Tree Transducers or k-Pebble Tree Transducers

Of course, one can always reduce to instances of the typechecking problem which are known to be decidable, such as for macro tree transducers or for k-pebble tree transducers. Maneth et al., for example, reduced the typechecking problem for TL-transformations to the typechecking problem for a composition of macro tree transducers [9]. More specifically, they showed that a TL program can be rewritten as a composition of *(i)* three deterministic MTTs when the TL program is deterministic, or *(ii)* two MTTs and one *stay* MTT otherwise. Here, a stay MTT is a MTT which does not have to go to the children of the current node in a computation step, but is also allowed to stay at the current node.

[3] Here, RE$^+$ is a restricted fragment of regular expressions.

The time complexity for the inferred typechecking algorithm will be rather high with these approaches. In the case of macro tree transducers, it is hyperexponential in the number of compositions, and in the case of k-pebble tree transducers it is hyperexponential in the number of pebbles [4,16].

5.2 Methods for Proving Lower Bounds

5.2.1 Inclusion, Emptiness, Universality

The most obvious way to obtain lower bounds on the complexity of the typechecking problem is to use a (trivial) reduction from the inclusion problem of the schema language at hand. Indeed, the inclusion problem is simply an instance of the typechecking problem in which the tree transducer performs the identity transformation.

Inclusion: Given schemas S_{in} and S_{out}, is $L(S_{\text{in}}) \subseteq L(S_{\text{out}})$?

Note that, when considering settings of the typechecking problem where the input and/or output schema is fixed, this straightforward approach does not work anymore. When only the output schema is fixed, the emptiness test of the input schema is a lower bound: test whether $L(S_{\text{in}}) \subseteq \emptyset$. When only the input schema is fixed, the universality test for the output schema can be a lower bound: test whether $\mathcal{T}_\Sigma \subseteq L(S_{\text{out}})$. When considering a setting for typechecking in which both schemas are fixed, however, one of the simulation methods in Section 5.2.2 might be more interesting.

The following propositions give an overview on the complexity of the inclusion, emptiness and universality problems for the the schema languages from Section 3:

Proposition 5.1 *The inclusion problem is*

- PTIME-*complete for* $\boldsymbol{D}$*(DFA)s [12,13];*
- PSPACE-*complete for* $\boldsymbol{D}$*(NFA)s and* $\boldsymbol{D}$*(RE)s [7,13,24]; and,*
- EXPTIME-*complete for EDTD(DFA)s, EDTD(NFA)s, and EDTD(RE)s [21],*

where $\boldsymbol{D}$ *stands for DTD, EDTD*st*, or EDTD*^{rc}*.*

Proposition 5.2 *The universality problem is*

- NLOGSPACE-*complete for* $\boldsymbol{D}$*(DFA)s;*
- PSPACE-*complete for* $\boldsymbol{D}$*(NFA)s and* $\boldsymbol{D}$*(RE)s [7,13,15]; and,*
- EXPTIME-*complete for EDTD(DFA)s, EDTD(NFA)s, and EDTD(RE)s [21].*

where $\boldsymbol{D}$ *stands for DTD, EDTD*st*, or EDTD*^{rc}*.*

Proposition 5.3 *The emptiness problem is*

- PTIME-*hard for DTD(DFA) [12]; and,*
- *in* PTIME *for EDTD(NFA)s and EDTD(RE)s [10].*

As opposed to the rest of the paper, we do *not* assume here that DTDs do not contain useless symbols (otherwise, the emptiness problem would be trivial).

Maybe a note on the mentioned references is appropriate: PSPACE completeness of the universality and equivalence problems for regular expressions and NFAs is shown in [7,15,24]. For DFAs, these problems are in PTIME. In the full version of [13], it is shown that these upper bounds carry over to DTDs, EDTDsts, and EDTD^{rc}s (lower bounds carry over trivially).

Proposition 5.2 states that the universality problem is NLOGSPACE-hard for DTD(DFA)s and in NLOGSPACE for EDTD^{rc}(DFA)s. The former is immediate, since graph reachability is NLOGSPACE-hard. The latter follows from the fact that universality for EDTD^{rc}(DFA)s can be decided using a reachability algorithm. Indeed, for a given EDTD^{rc} $\mathbf{d} = (\Sigma, \Sigma', d, \mu)$, we have to test whether the extended context free grammar d contains a symbol s that is *(i)* reachable from d's start symbol, and *(ii)* for which $\mu(d(s)) \neq \Sigma^*$. If this test succeeds, the EDTD^{rc} is non-universal. The result follows, as NLOGSPACE is closed under complement.

Seidl showed that the inclusion and equivalence problem are EXPTIME-complete for standard non-deterministic tree automata [21]. The EXPTIME upper bounds for EDTD(NFA)s and EDTD(RE)s can then be obtained through unranked-to-ranked encodings (which we also used in Section 4.2). Finally, the PTIME-hardness for emptiness of DTD(DFA)s is shown in [12], and PTIME membership for emptiness of EDTD(NFA)s and EDTD(RE)s follows from results in [10].

5.2.2 Simulations with the Tree Transducer

Perhaps the technically most interesting way to obtain lower bounds on the complexity of typechecking, is through simulation of a certain automata model (or, simulation of acceptance for classes of regular expressions) by the tree transducer.

Here, we focus on simulation of finite automata.[4] To this end, let $\mathcal{A}$ be a class of finite automata. The overall idea is to generate all possible inputs of $\mathcal{A}$ with the input schema. Then, we use the tree transducer to simulate one or more automata in $\mathcal{A}$ on the input tree and write to the output whether the input is accepted or not. The ouput schema can then verify whether some property holds for the simulated automata.

In this manner, we can test various decision problems of automata such as emptiness, inclusion, equivalence, and universality. For the inclusion problem of tree automata A_1 and A_2, for instance, we can do the following reduction. Our goal is to construct an instance of the typechecking problem that typechecks if and only if $L(A_1) \subseteq L(A_2)$. The input schema defines all possible

[4] Of course, the techniques we present can also be used to test e.g. satisfiability or validity of logical formulas.

input trees for A_1 or A_2. Given an input tree t, the tree transducer copies the t twice and simulates A_1 and A_2 in parallel on the left and right copy of t, respectively. At the end of this simulation, it can write to the output whether A_1, respectively A_2, accept t or not. Finally, the output schema can verify whether the tree transducer's output always encodes a situation in which A_1 had an accepting computation only if A_2 had an accepting computation. Note that, for this reduction, the tree transducer only has to be able to simulate the automata in $\mathcal{A}$, and copy a subtree of the input twice. Also, when the alphabets of the automata A_1 and A_2 are fixed, the constructed input and output schemas in the reduction do not depend on the given automata. Therefore, it is possible to obtain complexity hardness results using this reduction even in settings where *both the input and output schemas are fixed*.

When the tree transducer has the power to copy a part of its input tree an arbitrary number of times, the following problem can be interesting:

Intersection Emptiness: Given finite automata $A_1, \ldots, A_n$, is

$$L(A_1) \cap \cdots \cap L(A_n) = \emptyset?$$

The reduction in this case is analogous to the reduction that we just sketched. Typically, the tree transducer copies its input tree n times and simulates all the automata $A_1, \ldots, A_n$ in parallel. Finally, the output schema should verify whether the tree transducer's output always encodes a situation in which at least one simulated automaton rejects.

The following proposition gives the complexities of the intersection emptiness problem for various kinds of automata (or schema languages):

Proposition 5.4 *The intersection emptiness problem is*

- PSPACE-*complete for DFAs, NFAs, and REs [8];*
- EXPTIME-*complete for top-down deterministic (standard) tree automata [22];*
- PSPACE-*complete for DTD(DFA)s, DTD(NFA)s, and DTD(RE)s [13];*
- EXPTIME-*hard for EDTDst(DFA)s [13]; and*
- *in* EXPTIME *for EDTD(NFA)s and EDTD(RE)s.*

In the last item, membership in EXPTIME is immediate, as a product automaton for the intersection can be constructed in EXPTIME, and we can test emptiness of this product automaton in polynomial time in the size of the automaton [10].

This technique has been used many times in the literature. Milo, Suciu, and Vianu used a simulation of acceptance by *star-free generalized regular expressions* to obtain a non-elementary lower bound on the complexity of typechecking k-pebble tree transducers, even for fixed input and output schemas [16]. Martens, Neven, and Gyssens used a simulation of top-down deterministic standard tree automata by the simple tree transducers of Section 4.1 to obtain EXPTIME lowerbounds in several settings of the typechecking problem, also for fixed input and output schemas [12].

Acknowledgments

The author wishes to thank Frank Neven for his comments on a previous version of this paper.

References

[1] Alon, N., T. Milo, F. Neven, D. Suciu and V. Vianu, *XML with data values: typechecking revisited*, Journal of Computer and System Sciences **66** (2003), pp. 688–727.

[2] Bray, T., J. Paoli, C. Sperberg-McQueen, E. Maler and F. Yergeau, *Extensible Markup Language (XML)*, Technical report, World Wide Web Consortium (2004), http://www.w3.org/TR/REC-xml/.

[3] Brüggemann-Klein, A., M. Murata and D. Wood, *Regular tree and regular hedge languages over unranked alphabets: Version 1, april 3, 2001*, Technical Report HKUST-TCSC-2001-0, The Hongkong University of Science and Technology (2001).

[4] Engelfriet, J. and S. Maneth, *A comparison of pebble tree transducers with macro tree transducers*, Acta Informatica **39** (2003), pp. 613–698.

[5] Engelfriet, J. and H. Vogler, *Macro tree transducers*, Journal of Computer and System Sciences **31** (1985), pp. 71–146.

[6] Hosoya, H. and B. C. Pierce, *XDuce: A statically typed XML processing language*, ACM Transactions on Internet Technology **3** (2003), pp. 117–148.

[7] Hunt III, H. B., D. J. Rosenkrantz and T. G. Szymanski, *On the equivalence, containment, and covering problems for the regular and context-free languages*, Journal of Computer and System Sciences **12** (1976), pp. 222–268.

[8] Kozen, D., *Lower bounds for natural proof systems*, in: *Proc. 18th Annual Symposium on Foundations of Computer Science (FOCS 1977)*, 1977, pp. 254–266.

[9] Maneth, S., T. Perst, A. Berlea and H. Seidl, *XML type checking with macro tree transducers*, in: *Proc. 24th Symposium on Principles of Database Systems (PODS 2005)*, 2005, pp. 283–294.

[10] Martens, W. and F. Neven, *On the complexity of typechecking top-down XML transformations*, Theoretical Computer Science **336** (2005), pp. 153–180.

[11] Martens, W. and F. Neven, *Frontiers of tractability for typechecking simple XML transformations*, Journal of Computer and System Sciences (2006), to Appear.

[12] Martens, W., F. Neven and M. Gyssens, *On typechecking top-down XML tranformations: Fixed input or output schemas* (2005), submitted, available at http://alpha.uhasselt.be/wim.martens.

[13] Martens, W., F. Neven and T. Schwentick, *Complexity of decision problems for simple regular expressions*, in: *Proc. 29th Symposium on Mathematical Foundations of Computer Science (MFCS 2004)*, 2004, pp. 889–900.

[14] Martens, W., F. Neven and T. Schwentick, *Which XML schemas admit 1-pass preorder typing?*, in: *Proc. 10th International Conference on Database Theory (ICDT 2005)*, 2005, pp. 68–82.

[15] Meyer, A. R. and L. J. Stockmeyer, *The equivalence problem for regular expressions with squaring requires exponential space*, in: *Proc. 13th Annual Symposium on Foundations of Computer Science (FOCS 1972)*, 1972, pp. 125–129.

[16] Milo, T., D. Suciu and V. Vianu, *Typechecking for XML transformers*, Journal of Computer and System Sciences **66** (2003), pp. 66–97.

[17] Møller, A. and M. I. Schwartzbach, *The design space of type checkers for XML transformation languages*, in: *Proc. 10th International Conference on Database Theory (ICDT 2005)*, 2005, pp. 17–36.

[18] Murata, M., D. Lee and M. Mani, *Taxonomy of XML schema languages using formal language theory*, in: *Extreme Markup Languages*, Montreal, Canada, 2001.

[19] Murata, M., D. Lee, M. Mani and K. Kawaguchi, *Taxonomy of XML schema languages using formal language theory*, ACM Transactions on Internet Technology **5** (2005), to Appear, Full version of [18].

[20] Papakonstantinou, Y. and V. Vianu, *DTD inference for views of XML data*, in: *Proc. 19th ACM Symposium on Principles of Database Systems (PODS 2000)*, 2000, pp. 35–46.

[21] Seidl, H., *Deciding equivalence of finite tree automata*, SIAM Journal on Computing **19** (1990), pp. 424–437.

[22] Seidl, H., *Haskell overloading is DEXPTIME-complete*, Information Processing Letters **52** (1994), pp. 57–60.

[23] Sperberg-McQueen, C. and H. Thompson, *XML Schema* (2005), http://www.w3.org/XML/Schema.

[24] Stockmeyer, L. J. and A. R. Meyer, *Word problems requiring exponential time: Preliminary report*, in: *Proc. 5th Annual ACM Symposium on Theory of Computing (STOC 1973)*, 1973, pp. 1–9.

[25] Suciu, D., *Typechecking for semistructured data*, in: *Proc. 8th International Workshop on Database Programming Languages (DBPL 2001)*, 2001, pp. 1–20.

Timed Modal Logics for Specifying and Verifying Real-Time Systems

François Laroussinie

Lab. Spécification & Vérification
ENS de Cachan & CNRS UMR 8643
61, av. Pdt. Wilson, 94235 Cachan Cedex France
email: fl@lsv.ens-cachan.fr

Abstract. The timed modal logic L_ν has been proposed in order to express timed properties over real-time systems modeled as (compositions of) timed automata. In this paper, we present a short survey of results about L_ν: complexity of model checking, expressivity, compositional methods, relationship with strong timed bisimulation etc. We also show how L_ν can be extended in order to express new properties.

1 Introduction

Model checking is widely used for the design and debugging of critical reactive systems [Eme90,CGP99]. During the last fifteen years, it has been extended to *real-time systems*, where quantitative information about time is required.

Timed models. Real-time model checking has been mostly studied and developed in the framework of Alur and Dill's *Timed Automata* (TA) [ACD93,AD94], *i.e.* automata extended with *clocks* that progress synchronously with time. The behavior of a real-time system can often be modeled as a parallel composition of TA. There now exists a large body of theoretical knowledge and practical experience for this class of systems.

Timed specifications. Temporal and modal logics provide a fundamental framework for formally specifying systems and reasoning about them [Eme90,MP92]. For example, a property like "any problem is followed by an alarm" can be easily expressed with most of the temporal logics (*CTL*, *LTL*, μ-calculus, etc.).

When specifying real-time systems, it is necessary to express *timed property* like:

"Any problem is followed by an alarm in at most 10 time units." (1)

In order to express timing aspects of computations, we can extend the classical temporal or modal logics. A first possibility is to use timing constraints tagging temporal modalities [AH92]. For example, with the timed version of *CTL*, namely *TCTL*, one can specify the property (1) as follows:

$$\mathsf{AG}\Big(\mathsf{problem} \;\Rightarrow\; \mathsf{AF}_{\leq 10}\,\mathsf{alarm}\Big)$$

A more expressive method consists in adding clocks – a.k.a. freeze variables – in the specification language [AH94,BCM05]. In this framework, a formula clock (with the $\underline{\texttt{in}}$ operator) can be reset and compared with some constant later on. For example, the previous property can be written as follows:

$$\mathsf{AG}\Big(\mathsf{problem} \;\Rightarrow\; x \;\underline{\texttt{in}}\; \mathsf{AF}(x \leq 10 \;\wedge\; \mathsf{alarm})\Big)$$

The same extensions can be done for modal logics with fixpoint operators.

Outline. In this paper, we consider the timed modal logic L_ν that has been introduced in [LLW95]. It is a natural extension of classical modal logics for finite labeled transition systems with (1) two modalities $\langle\delta\rangle$ and $[\delta]$ in order to deal with delay transitions of the system to be specified (in addition to the standard modalities for action transitions), (2) with formula clocks and (3) maximal fixpoint operators.

We address complexity questions for model checking and present several properties about the expressive power of L_ν. This paper contains results from the following papers [AL02,LL95,LLW95,LL98,BCL05a].

2 Definitions

We first define the *timed automata* proposed by Alur and Dill [AD94] and then we introduce the timed modal logic L_ν.

2.1 Timed Automata

Notations. Let Act be a finite set of *actions*, and let $\mathbb{N}$ and $\mathbb{R}_+$ denote the sets of natural and non-negative real numbers, respectively. Let X be a set of clocks. A *clock valuation* for X is a function from X to $\mathbb{R}_+$, the set of valuations is denoted $\mathbb{R}_+^X$. Given a valuation $v \in \mathbb{R}_+^X$ and $t \in \mathbb{R}_+$, $v+t$ is a valuation assigning $v(x)+t$ for any $x \in X$. Let $r \subseteq X$, $v[r \leftarrow 0]$ be the valuation assigning 0 (resp. $v(x)$) for any $x \in r$ (resp. $x \in X \setminus r$).

We use $\mathcal{C}(X)$ to denote the set of *clock constraints* defined as the boolean combinations of atomic constraints of the form $x \sim p$ or $x-y \sim p$, with $x, y \in X$, $p \in \mathbb{N}$, and $\sim \in \{<, \leq, >, \geq, =\}$. Given $g \in \mathcal{C}(X)$ and $v \in \mathbb{R}_+^X$, we write $v \models g$ when v satisfies g. We write $\mathcal{C}_<(X)$ for the restriction of $\mathcal{C}(X)$ to the positive combinations of constraints $x \leq p$ or $x < p$, with $p \in \mathbb{N}$.

Definition 2.1. *A* timed automaton (TA) *is a tuple* $\mathcal{A} = (L, \ell_0, \mathsf{Act}, X, \mathsf{Inv}, T)$ *where L is a finite set of locations, $\ell_0 \in L$ is the initial location, X is a finite set of clocks, $\mathsf{Inv} : L \to \mathcal{C}_<(X)$ is a function that assigns an invariant to each location, and $T \subseteq L \times \mathcal{C}(X) \times \mathsf{Act} \times 2^X \times L$ is a finite set of edges. A quintuple*

$(\ell, g, a, r, \ell') \in T$ — *denoted by* $\ell \xrightarrow{g,a,r} \ell'$ — *represents an edge from location* ℓ *to location* ℓ' *with action* a*,* g *is the* guard *and* r *is a set of clocks to be reset to* 0*.*

A guard is used to specify when a transition can be performed. An invariant is used to avoid excessive time delays in a location and thus it may enforce action transitions to be performed.

The semantics of TA is defined as a *Timed Transition System* (TTS), that is a tuple $S = (Q, q_0, \mathsf{Act}, \rightarrow_S)$ where Q is a set of states, $q_0 \in Q$ is the initial state, and $\rightarrow_S \subseteq Q \times (\mathsf{Act} \cup \mathbb{R}_+) \times Q$ is a set of transitions (we write $q \xrightarrow{e}_S q'$ when $(q, e, q') \in \rightarrow_S$). The transitions labeled by $a \in \mathsf{Act}$ (resp. $t \in \mathbb{R}_+$) are called *action* (resp. *delay*) transitions. We make the following common assumptions about delay transitions in TTSs:

- 0-delay: $q \xrightarrow{0}_S q'$ if and only if $q = q'$,
- Time-Additivity: if $q \xrightarrow{t}_S q'$ and $q' \xrightarrow{t'}_S q''$ with $t, t' \in \mathbb{R}_+$, then $q \xrightarrow{t+t'}_S q''$,
- Time-Continuity: if $q \xrightarrow{t}_S q'$, then $\forall t', t'' \in \mathbb{R}_+$ with $t = t' + t''$, there exists q'' such that $q \xrightarrow{t'}_S q'' \xrightarrow{t''}_S q'$,
- Time-determinism: if $q \xrightarrow{t}_S q'$ and $q \xrightarrow{t}_S q''$ with $t \in \mathbb{R}_+$, then $q' = q''$.

Standard notions of finite or infinite runs apply to TTS.

Given a TA $\mathcal{A} = (L, \ell_0, \mathsf{Act}, X, \mathsf{Inv}, T)$, we define its semantics as the TTS $S_{\mathcal{A}} = (L \times \mathbb{R}_+^X, (\ell_0, v_0), \mathsf{Act}, \rightarrow_{S_{\mathcal{A}}})$ where $v_0(x) = 0$ for all $x \in X$ and $\rightarrow_{S_{\mathcal{A}}}$ consists of:

1. action transition: $(\ell, v) \xrightarrow{a}_{S_{\mathcal{A}}} (\ell', v')$ if there exists an edge $\ell \xrightarrow{g,a,r} \ell'$ in T s.t. $v \models g$, $v' = v[r \leftarrow 0]$ and $v' \models \mathsf{Inv}(\ell')$;
2. delay transitions: $(\ell, v) \xrightarrow{t}_{S_{\mathcal{A}}} (\ell, v')$ if $t \in \mathbb{R}_+$, $v' = v + t$ and[1] $v' \models \mathsf{Inv}(\ell)$.

Then a *state* (or *configuration*) of a timed automaton A is a pair (ℓ, v), where ℓ is a location – a control state – and v is a clock valuation for X. A key point (for decidability) is the synchronous time elapsing: all clocks have the same speed.

The size of a TA is $|L| + |X| + \sum_{(\ell,g,a,r,\ell') \in T} |g| + \sum_{\ell} |\mathsf{Inv}(\ell)|$ where the size of a constraint is its length (constants are encoded in binary).

Networks of timed automata We model real time systems as parallel compositions of timed automata with n-ary synchronization functions. Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be n timed automata. A *synchronization function* f is a partial function $(\mathsf{Act} \cup \{\bullet\}) \times \ldots \times (\mathsf{Act} \cup \{\bullet\}) \rightarrow \mathsf{Act}$, where $\bullet$ denotes a distinguished no–action symbol. Note that f is a synchronization function with renaming. We write $(\mathcal{A}_1 | \ldots | \mathcal{A}_n)_f$ for the parallel composition of $\mathcal{A}_1, \ldots, \mathcal{A}_n$ w.r.t. function f. A network configuration is a pair $\langle \bar{\ell}, v \rangle$ where $\bar{\ell} = (\ell_1, \ldots, \ell_n)$ is a vector of locations and v is a valuation for $X = \cup_i X_i$, i.e. the clocks of the network (we denote by v_i the restriction $v_{|X_i}$).

[1] due to the definition of the invariants, this entails: $v + t' \models \mathsf{Inv}(\ell)$ for any $0 \leq t' \leq t$.

The semantics of $(\mathcal{A}_1|\ldots|\mathcal{A}_n)_f$ can be defined as a TTS whose states are the configurations of the network and the transitions are given by the two following rules:

$$\begin{array}{ll} \langle \bar{\ell}, v\rangle \xrightarrow{t} \langle \bar{\ell}, v+t\rangle & \text{iff } \forall i \in \{1,\ldots,n\}, \langle \ell_i, v_i\rangle \xrightarrow{t} \langle \ell_i, v_i+t\rangle \\ \langle \bar{\ell}, v\rangle \xrightarrow{b} \langle \bar{\ell}', v'\rangle & \text{iff } \forall i \in \{1,\ldots,n\}, a_i \in \mathsf{Act} \text{ implies } \langle \ell_i, v_i\rangle \xrightarrow{a_i} \langle \ell'_i, v'_i\rangle \text{ and} \\ & \quad a_i = \bullet \text{ implies } (\ell'_i = \ell_i \ \wedge \ v'_i = v_i) \text{ and} \\ & \quad v' = v'_1 \ldots v'_n \text{ and } f(a_1,\ldots,a_n) = b \end{array}$$

Note that the parallel composition does not add expressive power: from any parallel composition, one can build an *equivalent* (i.e. strongly bisimilar, see Section 4.1) timed automaton.

2.2 The Timed Modal Logic L_ν

We now define L_ν, a timed modal μ-calculus [LLW95]:

Definition 2.2. *Let K be a finite set of clocks (disjoint from X), and Id be a countably infinite set of identifiers (ranged over by X, Y). The set L_ν of formulae over Act, K and Id is generated by the following grammar:*

$$\begin{array}{l} \varphi ::= \mathtt{tt} \mid \mathtt{ff} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid r \ \underline{\mathtt{in}} \ \varphi \mid g \mid [a]\,\varphi \mid \langle a\rangle\,\varphi \mid \\ \qquad [\delta] \ \varphi \mid \langle \delta\rangle\,\varphi \mid Z \end{array}$$

where $a \in \mathsf{Act}$, $g \in \mathcal{C}(K)$, $r \subseteq K$ and $Z \in \mathsf{Id}$.

The meaning of the identifiers is specified by a declaration $\mathcal{D}$ assigning an L_ν formula to each identifier in order to define properties with maximal fixpoints. Let $\mathcal{A}$ be a TA or a parallel composition of TA. We interpret L_ν formulae over extended states (ℓ, v, u) where (ℓ, v) is a configuration of A and u is a clock valuation for K. This satisfaction relation $\models_{\mathcal{A},\mathcal{D}}$ is defined as the largest relation satisfying the implications in Table 1 (for $\models_{\mathcal{A},\mathcal{D}}$ the implications are bi-implications). The modalities $\langle e\rangle$ with $e \in \mathsf{Act} \cup \{\delta\}$ correspond to existential quantification over action or delay transitions, and $[e]$ is the counterpart for universal quantification. An extended state satisfies an identifier Z if it belongs to the maximal fixpoint of the equation $Z =_\nu \mathcal{D}(Z)$. Finally the formula clocks are used to measure time elapsing in properties. We write $A \models \varphi$ for $(\ell_0, v_0, u_0) \models \varphi$ where $u_0(x) = 0$ for all $x \in K$.

We can use L_ν formula to express classical temporal (and timed) properties:

- $z \ \underline{\mathtt{in}} \ \langle\delta\rangle \ (z < 10 \wedge (\langle a\rangle \ \mathtt{tt} \vee \langle b\rangle \ \mathtt{tt}))$ specifies that an a-action or a b-action is enabled before 10 time units (without performing any action transition).
- We can express that some formula φ holds for any reachable state ("$\mathsf{ALWAYS}_(\varphi)$", corresponding to the AG operator from CTL). This can be defined by the following equation: $Z \stackrel{\text{def}}{=} \varphi \wedge \bigwedge_{a \in \mathsf{Act}} [a]\,Z \wedge [\delta]\,Z$.

$(\ell, v, u) \models \mathtt{tt} \implies \mathtt{tt}$
$(\ell, v, u) \models \mathtt{ff} \implies \mathtt{ff}$
$(\ell, v, u) \models g \implies v \models g$
$(\ell, v, u) \models \varphi_1 \vee \varphi_2, \implies (\ell, v, u) \models \varphi_1 \quad or \quad (\ell, v, u) \models \varphi_2$
$(\ell, v, u) \models \varphi_1 \wedge \varphi_2, \implies (\ell, v, u) \models \varphi_1 \quad and \quad (\ell, v, u) \models \varphi_2$
$(\ell, v, u) \models r \ \underline{\mathtt{in}} \ \varphi \implies (\ell, v, u[r \leftarrow 0]) \models \varphi$
$(\ell, v, u) \models [a]\, \varphi \implies$ for all $(\ell, v) \xrightarrow{a}_{S_{\mathcal{A}}} (\ell', v')$, $(\ell', v', u) \models \varphi$
$(\ell, v, u) \models \langle a \rangle\, \varphi \implies$ there is some $(\ell, v) \xrightarrow{a}_{S_{\mathcal{A}}} (\ell', v')$, $(\ell', v', u) \models \varphi$
$(\ell, v, u) \models [\delta]\, \varphi \implies$ for all $t \in \mathbb{R}_+$ s.t. $(\ell, v) \xrightarrow{t}_{S_{\mathcal{A}}} (\ell, v + t)$, $(\ell, v + t, u + t) \models \varphi$
$(\ell, v, u) \models \langle \delta \rangle\, \varphi \implies$ there is some $t \in \mathbb{R}_+$ s.t. $(\ell, v) \xrightarrow{t}_{S_{\mathcal{A}}} (\ell, v + t)$, $(\ell, v + t, u + t) \models \varphi$
$(\ell, v, u) \models Z \implies (\ell, v, u) \models \mathcal{D}(Z)$

Table 1. Satisfaction implications for L_ν

– Given two subsets of events Act_1 and Act_2, we can state that any event in Act_1 is followed by an event in Act_2 in less than Δ time units (assume $\Delta \in \mathbb{N}$):

$$Z_1 \stackrel{\text{def}}{=} \textstyle\bigwedge_{a \in \mathsf{Act}_1} [a]\, (y \ \underline{\mathtt{in}} \ Z_2) \wedge \bigwedge_{a \in \mathsf{Act} \setminus \mathsf{Act}_1} [a]\, Z_1 \wedge [\delta]\, Z_1$$
$$Z_1 \stackrel{\text{def}}{=} (y < \Delta) \wedge \textstyle\bigwedge_{a \in \mathsf{Act}_2} [a]\, Z_1 \wedge \bigwedge_{a \in \mathsf{Act} \setminus \mathsf{Act}_2} [a]\, Z_2 \wedge [\delta]\, Z_2$$

– To express that some property φ will hold during at least Δ time units, whatever the transitions performed ($\varphi \mathsf{UpTo}\ \Delta$), we can use the L_ν formula $z \ \underline{\mathtt{in}} \ Z_1$ with:

$$Z_1 \stackrel{\text{def}}{=} (z > \Delta) \vee \Big(\varphi \wedge \bigwedge_{a \in \mathsf{Act}} [a]\, Z_1 \wedge [\delta]\, Z_1 \Big)$$

3 Model checking

Region graph technique. Automatic verification of timed systems modelled as (networks of) timed automata is possible despite the uncountably infinite number of states associated with the semantics of a timed automaton. The decision procedure for the problem $A \models \varphi$ is based on the well known *region technique* (see [ACD93] for a description for *TCTL*). Given A and φ, it is possible to partition the uncountably infinite set of time assignments over $X \cup K$ into a finite number of *regions*, s.t. two extended configurations (ℓ, v, u) and (ℓ, v', u'), where uv and $u'v'$ are in the same region, satisfy the same subformulae of φ. Let c_{max} be the maximal constant occurring in the guards and invariants of A and in the formula φ. The regions can be defined as the equivalence classes induced by the equivalence relation over valuations defined as follows: two valuations $w, w' \in \mathbb{R}_+^{X \cup K}$ are in the same region iff they satisfy the same clock constraints from the set $\mathcal{C}_{c_{max}}(X \cup K)$ containing $\mathcal{C}(X \cup K)$ expressions whose

integer constants belong to $\{0, \ldots, c_{max}\}$. Due to this bound over the constants, the number of equivalence classes is clearly finite.

Then we can define a symbolic semantics [LLW95] for A over a finite transition system, called the *region graph*, whose states are pairs (ℓ, γ): a location and a region. And formulae in L_ν can be interpreted over the states of the region graph.

Thus the region graph technique provides decidability results for many verification problems over timed systems because it allows us to reduce a question over an infinite transition system to a problem over a finite one. But note that the number of regions is in $O(|X \cup K|! \cdot c_{max}^{|X \cup K|})$ and the size of the region graph is in $O((|L| + |T|) \cdot |X \cup K|! \cdot c_{max}^{|X \cup K|})$. Then in practice, the region graph is not built in timed model checkers. and in the tools like Uppaal[LPY97] or Kronos[Yov97], the algorithms are based on an efficient data-structure, the DBMs – Differences Bounded Matrices – that allow to handle convex sets of valuations. Moreover several heuristics have been developed to improve the efficiency of the algorithms (on-the-fly algorithms, reduction of the number of clocks, ...).

Verification of L_ν properties. Model checking L_ν over networks of timed automata is then decidable:

Theorem 3.1 ([AL02]). *Model checking problem for L_ν is EXPTIME-complete. The specification and program complexities are also EXPTIME-complete.*

The EXPTIME membership comes from the fact that applying standard verification algorithms for modal logics over the region graph can be done in time linear in the size of φ and the size of the region graph, this provides an algorithm which is exponential in $|\mathcal{A}|$ and $|\varphi|$.

The EXPTIME-hardness is proved by reducing the acceptance of a word w by a linear (space) bounded alternating Turing machine $\mathcal{M}$ to a model checking problem for L_ν. We assume (w.l.o.g.) that there is a strict alternation of existential and universal states in $\mathcal{M}$ and that the initial state is existential. First one can build a timed automaton $\mathcal{A}_{\mathcal{M},w}$ that represents the behavior of the non-alternating version of $\mathcal{M}$ over the word w (clocks are used to encode the contents of the $|w|$ cells of the tape): any action transition corresponds to a step of $\mathcal{M}$ and between two actions one require a strictly positive delay. We distinguish three labels for actions of $\mathcal{M}$: Init corresponds to the writing of w on the tape at the beginning of the computation, a labels any step of $\mathcal{M}$ and $\mathsf{Accepting}$ labels accepting states of $\mathcal{M}$. Secondly one can use the L_ν formula $\Phi_{na} \stackrel{\text{def}}{=} [\delta]\,[\mathsf{Init}]\,Y$ with :

$$Y \stackrel{\text{def}}{=} [\mathsf{Accepting}]\,\mathtt{ff} \wedge [\delta]\,[\mathsf{a}]\,(\langle\delta\rangle\,\langle\mathsf{a}\rangle\,Y)$$

Then the formula Φ_{na} holds for the initial state of $\mathcal{A}_{\mathcal{M},w}$ iff $\mathcal{M}$ does not accept w. See [AL02] for the full proof.

The main difference with reachability or *TCTL* model checking (which are PSPACE-complete problems for TA) is the ability for L_ν to simulate the alternating behavior of $\mathcal{M}$.

From the previous construction, one can easily deduce that the program complexity is also EXPTIME-hard since the formula used in the reduction does not depend on $\mathcal{M}$ and w.

Finally the specification complexity comes from the ability of L_ν to encode the behavior of timed automata. The previous reduction could have been done for a simple automaton with no clock and no edge and with a more complex formula. This can be obtained as a direct consequence of the properties of L_ν presented in the next section.

Satisfiability. The decidability of the satisfiability problem — given an L_ν formula Φ, does there exists a timed automaton $\mathcal{A}$ s.t. $\mathcal{A} \models \Phi$ — is still an opened problem [LLW95]. Note that for most of the timed branching-time temporal logics (*TCTL*, T_μ) the satisfiability problem is undecidable [ACD93,AFH96,HNSY94]. For the timed linear-time temporal logics like *MTL*, satisfiability is undecidable for the (standard) interval-based semantics [HNSY94], but is decidable for pointwise semantics [OW05].
Finally note that if we bound the *resources* of $\mathcal{A}$ — its number of clocks and the maximal constant occurring in its guards — the problem is decidable for L_ν [LLW95].

4 Expressivity

4.1 Strong timed bisimulation

The standard notion of bisimulation [Mil89,Par81] can be naturally extended to timed systems : let S_A and S_B two TTSs and and Q_A and Q_B be their set of states, the *strong timed bisimulation* $\sim$ is defined as the largest symmetric relation over $Q_A \times Q_B$ such that whenever $q_A \sim q_B$, we have (1) for any $q_A \xrightarrow{a} q'_A$ with $a \in \mathsf{Act}$, there exists $q_B \xrightarrow{a} q'_B$ and $q'_A \sim q'_B$, and (2) for any $q_A \xrightarrow{t} q'_A$ with $t \in \mathbb{R}_+$, there exists $q_B \xrightarrow{t} q'_B$ and $q'_A \sim q'_B$.

We say that two TA $\mathcal{A}_1$ and $\mathcal{A}_2$ are *strong timed bisimilar* if their corresponding TTSs are strong bisimilar. In the following we show that some L_ν can express strong timed bisimilarity of timed automata.

Let $\mathcal{A}_1 = (L^1, \ell_0^1, \mathsf{Act}, X_1, \mathsf{Inv}_1, T_1)$ and $\mathcal{A}_2 = (L^2, \ell_0^2, \mathsf{Act}, X_2, \mathsf{Inv}_2, T_2)$ be two timed automata. Let σ be the binary synchronization function defined as follows : $\forall a \in \mathsf{Act}$, we have $\sigma(a, \bullet) = a_1$ and $\sigma(\bullet, a) = a_2$. With σ, the action transitions of the TA are not synchronized and they are tagged by the number of the active TA. But the delay transitions are synchronized : every clock of the composition progresses synchronously with time.

Now assume $\mathcal{A}_1$ and $\mathcal{A}_2$ have no invariant (for any location ℓ, we have $\mathsf{Inv}_i(\ell) = \mathtt{tt}$). Then we have: $\mathcal{A}_1 \sim \mathcal{A}_2$ iff $(\mathcal{A}_1 | \mathcal{A}_2)_\sigma \models Z$ with:

$$Z \stackrel{\text{def}}{=} \bigwedge_{a \in \mathsf{Act}} \Big([a_1] \langle a_2 \rangle Z \wedge [a_2] \langle a_1 \rangle Z \Big) \wedge [\delta] Z$$

Indeed the definition of Z is precisely the definition of the strong timed bisimilarity.

When the automata contain invariants, the parallel composition $(\mathcal{A}_1|\mathcal{A}_2)_\sigma$ does not contain all the behaviors of $\mathcal{A}_1$ and $\mathcal{A}_2$: from a configuration $(\ell_1, \ell_2, v_1 v_2)$ of $(\mathcal{A}_1|\mathcal{A}_2)_\sigma$, a delay transition $\xrightarrow{t}$ is enabled iff it is enabled for both automata. If ℓ_1 has an invariant, the parallel composition does not contain delays that violate this invariant even if such delays exist in $\mathcal{A}_2$. The solution consists in defining new TA $\mathcal{A}'_i$'s with the same locations L^i, the same clocks X_i, the set of actions $\mathsf{Act} \cup \{\mathsf{InvFail}\}$, but without invariant and with another set of edges T'_i defined as follows:

- For any location $\ell \in L^i$, we add a transition $(\ell, \neg \mathsf{Inv}(\ell), \mathsf{InvFail}, \emptyset, \ell)$ to T'_i.
- For any transition (ℓ, g, a, r, ℓ') in T_i, we add a transition $(\ell, g \wedge \mathsf{Inv}(\ell) \wedge (\mathsf{Inv}(\ell')[r \leftarrow 0]), r, \ell')$ to T'_i. Where $g[r \leftarrow 0]$ denotes the constraint g where every occurrence of the clocks in r are replaced by the constant 0.

The first kind of transitions is used to mark states which are not reachable in $\mathcal{A}_i$ because the current invariant is violated. The second kind of transitions corresponds to original action steps but we require in the guard that the current invariant has to be satisfied and also that the invariant of the target location has to be satisfied by the valuation after the reset: indeed we want to deal only with transitions that are actually enabled.

Finally we have: $\mathcal{A}_1 \sim \mathcal{A}_2$ iff $(\mathcal{A}'_1|\mathcal{A}'_2)_\sigma \models Z$ where Z is defined as above.

4.2 Compositionality

Compositional model checking [And95] is possible for L_ν: given a parallel composition $(\mathcal{A}_1|\ldots|\mathcal{A}_n)_f$ and some L_ν formula Φ, one can build a *quotient* formula $\Phi/\mathcal{A}_1$ s.t. $(\mathcal{A}_1|\ldots|\mathcal{A}_n)_f \models \Phi$ iff $(\mathcal{A}_2|\ldots|\mathcal{A}_n)_f \models \Phi/\mathcal{A}_1$. The formula $\Phi/\mathcal{A}_1$ integrates the initial property and the pertinent part (w.r.t. Φ) of the behavior of $\mathcal{A}_1$. By repeatedly quotienting components from the network into the formula, we will finally be faced with the verification problem: $nil \models \Phi/\mathcal{A}_1/\ldots/\mathcal{A}_n$ where nil is an untimed automaton unable to perform action transitions.

Table 2 presents the definition of quotienting for L_ν [LL95,LL98]. Note that the quotient is defined for a formula, a location and a synchronization function f. Moreover the clocks of the quotiented automaton become formula clocks in the quotient formula. Finally any fixpoint variable of Φ may give rise to $|L|$ variables in $\Phi/\mathcal{A}$.

We have the following theorem:

Theorem 4.1 ([LL98]). *Given two timed automata $\mathcal{A}_1$ et $\mathcal{A}_2$, a synchronization function f and an L_ν formula φ, we have the following property for any configuration $(\ell_1, \ell_2, v_1 v_2)$ of $(\mathcal{A}_1|\mathcal{A}_2)_f$ and $u \in \mathbb{R}_+^K$:*

$(\varphi_1 \wedge \varphi_2)/\ell = (\varphi_1/\ell) \wedge (\varphi_2/\ell)$	$Z/\ell = Z^\ell$
$(\varphi_1 \vee \varphi_2)/\ell = (\varphi_1/\ell) \vee (\varphi_2/\ell)$	
$\Big(\langle a \rangle \varphi\Big)/\ell = \bigvee_{\ell \xrightarrow{g,b,r} \ell' \text{s.t.} f(b,c)=a} g \wedge \langle c \rangle \Big(r \ \underline{\texttt{in}} \ (\mathsf{Inv}(\ell') \wedge \varphi/\ell')\Big)$	
$\Big([a] \varphi\Big)/\ell = \bigwedge_{\ell \xrightarrow{g,b,r} \ell' \text{s.t.} f(b,c)=a} g \Rightarrow [c] \Big(r \ \underline{\texttt{in}} \ (\mathsf{Inv}(\ell') \Rightarrow \varphi/\ell')\Big)$	
$\langle \delta \rangle \varphi/\ell = \langle \delta \rangle \Big(\mathsf{Inv}(\ell) \wedge \varphi/\ell\Big)$	$[\delta] \varphi/\ell = [\delta] \Big(\mathsf{Inv}(\ell) \Rightarrow \varphi/\ell\Big)$
$(x + c \bowtie y + d)/\ell = (x + c \bowtie y + d)$	$(x \ \underline{\texttt{in}} \ \varphi)/\ell = x \ \underline{\texttt{in}} \ \Big(\varphi/\ell\Big)$

Table 2. Quotient construction.

$$\Big(((\ell_1, \ell_2), v_1 v_2, u) \models \varphi\Big) \quad \textit{iff} \quad (\ell_2, v_2, u v_1) \models \varphi/\ell_1$$

This technique allows us to avoid the construction of the (exponential) construction of the product corresponding to the parallel composition $(\mathcal{A}_1 | \ldots | \mathcal{A}_n)_f$, but of course this complexity is reported in the formula: the size of the quotient formula $\Phi/\mathcal{A}_1$ is in $O(|\Phi| \cdot |\mathcal{A}_1|)$. In order to be applied, the compositional method needs to be completed with reductions: after every quotienting operation, we apply reduction laws in order to keep the size of the formula as small as possible (see [LL98] for a description of these reductions).

The tool CMC (Compositional Model Checker)[2] implements this method for the verification of L_ν properties for TA.

This technique has also been extended to linear hybrid systems [CL00].

4.3 Characteristic properties of timed automata

Given a timed automaton $\mathcal{A} = (L, \ell_0, \mathsf{Act}, X, \mathsf{Inv}, T)$, it is possible to build an L_ν formula $\Phi_\mathcal{A}$ that precisely characterizes the behavior of $\mathcal{A}$: a timed automaton is strongly timed-bisimilar to $\mathcal{A}$ iff it satisfies $\Phi_\mathcal{A}$. This construction can be done directly from the definition of $\mathcal{A}$ [LLW95] or it can be seen as a consequence of the ability to express timed bisimilarity and the compositionality. Indeed consider the variable Z defined as above to express strong bisimilarity, then the formula Z^{ℓ_0} defined as the quotient[3] Z/ℓ_0 verifies the following property for any TA $\mathcal{B}$:

$$\mathcal{A} \sim \mathcal{B} \quad \text{iff} \quad \mathcal{B}' \models Z^{\ell_0}$$

where $\mathcal{B}'$ denotes the TA without invariant described in Section 4.1.

Then the timed modal logic L_ν is expressive enough to describe the behavior of timed automata.

[2] http://www.lsv.ens-cachan.fr/~fl/cmcweb.html

[3] with a declaration assigning a definition to every new identifier Z^ℓ.

5 An extension of L_ν

Recently a new operator has been added to L_ν in order to express controllability properties [BCL05a]. More generally the expressivity of L_ν is limited in the way of dealing with delay transitions: the modalities $\langle\delta\rangle$ and $[\delta]$ are respectively too weak or too strong for specifying some natural properties.

For example, consider the classical $\mathsf{E}a\mathsf{W}b$ formula: "there exists a path satisfying $a\mathsf{W}b$" where W stands for the *weak* until operator: either the path satisfies $a\mathsf{U}b$ (there is eventually a b and the previous states satisfy a), or it satisfies $\mathsf{G}a$ ("always a"). In classical Kripke structures, this property can be expressed as the greatest fixpoint of $Y \stackrel{\text{def}}{=} b \vee (a \wedge \mathsf{EX}Y)$ where EX denotes the Next operator. In our framework, the property $\mathsf{E}\varphi\mathsf{W}\psi$ cannot be expressed in L_ν due to the lacks of expressivity of $\langle\delta\rangle$ and $[\delta]$. Indeed, from a current state, we need to state that there is a way of letting time elapse s.t. all visited states satisfy φ, until a state where an action can be performed and so on. But the formula $\langle\delta\rangle\,\xi$ allows us to specify that after some delay, the property ξ holds, but there is no requirement on the intermediary states. And the formula $[\delta]\,\xi$ requires that *any* state reachable via a delay transition has to satisfy ξ. This does not allow us to express $\mathsf{E}\varphi\mathsf{W}\psi$ and which has motivated the introduction of the operator $[\delta\rangle$ defined as follows [BCL05a]:

$$
\begin{array}{r}
(\ell, v, u) \models \varphi\,[\delta\rangle\,\psi \Leftrightarrow \text{either } \forall t \in \mathbb{R}_+,\ (\ell, v) \xrightarrow{t} (\ell, v+t) \Rightarrow (\ell, v+t, u+t) \models \varphi \\
\text{or } \exists t \in \mathbb{R}_+ \text{ s.t. } (\ell, v) \xrightarrow{t} (\ell, v+t) \text{ and } (\ell, v+t, u+t) \models \psi \text{ and} \\
\forall 0 \leq t' < t,\ \text{we have } (\ell, , v+t', u+t') \models \varphi
\end{array}
$$

Note that in [HNSY94] a timed μ-calculus (T_μ) has been defined with a modality $\triangleright$ whose semantics is close to the one of $[\delta\rangle$ [4].

The new operator $[\delta\rangle$ is a kind of weak Until operator over delay transitions. We could have defined the operator $[\delta\rangle\rangle$ corresponding to a (strong) Until requiring that a position satisfying ψ exists. But we can easily define this modality with $[\delta\rangle$ and vice versa:

$$
\begin{array}{rcl}
\varphi\,[\delta\rangle\rangle\,\psi & \equiv & \varphi\,[\delta\rangle\,\psi \wedge \langle\delta\rangle\,\psi \\
\varphi\,[\delta\rangle\,\psi & \equiv & \varphi\,[\delta\rangle\rangle\,\psi \vee [\delta]\,\varphi
\end{array}
$$

Let L_ν^+ be the extension of L_ν with the modality $[\delta\rangle$. This new logic is more expressive than L_ν:

Lemma 5.1 ([BCL05a]). *The L_ν^+ formula $([a]\,\mathtt{ff})\,[\delta\rangle\,(\langle b\rangle\,\mathtt{tt})$ has no equivalent in L_ν.*

The full proof can be found in [BCL05b]. Let Φ be the formula $([a]\,\mathtt{ff})\,[\delta\rangle\,(\langle b\rangle\,\mathtt{tt})$. The difficult point is that it is not possible to find two TA $\mathcal{A}$

[4] The main difference between $\triangleright$ and $[\delta\rangle$ is that $\triangleright$ may include an action transition after the delay

and $\mathcal{B}$ such that $\mathcal{A} \models \Phi$, $\mathcal{B} \not\models \Phi$ and $\mathcal{A} \models \psi \Leftrightarrow \mathcal{B} \models \psi$ for any $\psi \in L_\nu$. Indeed as we have seen in Section 4.3, L_ν formulas allow us to distinguish between two TA that are not bisimilar and if $\mathcal{A} \models \Phi$ and $\mathcal{B} \not\models \Phi$, then $\mathcal{A} \not\approx \mathcal{B}$. This is a classical problem in temporal logic [EMSS91] where one shows that two temporal logics may have different *expressive powers* even if they have the same *distinguishing power* and this makes the proof about expressivity more difficult.

Adding the modality $[\delta\rangle$ does not modify the interesting properties of L_ν:

- The logic L_ν^+ allows compositional model checking. It is sufficient to add the following rule to the quotient definition:

$$\Big(\varphi_1\,[\delta\rangle\,\varphi_2\Big)/\ell \stackrel{\text{def}}{=} \Big(\mathsf{Inv}(\ell) \Rightarrow (\varphi_1/\ell)\Big)\,[\delta\rangle\,\Big(\mathsf{Inv}(\ell) \wedge (\varphi_2/\ell)\Big)$$

- Moreover there is no additional complexity for the verification: Model checking L_ν^+ is EXPTIME-complete.

Control and controllability. The main motivation for adding $[\delta\rangle$ to L_ν was related to *control problems*. In this framework, we address a more general problem than classical model checking: given a plant P and a control objective Φ, one aims at synthesizing a controller C such that $C(P) \models \Phi$ where $C(P)$ denotes the plant supervised by C. Usually P is a (timed) automaton describing a system and its environment and we often distinguish between controllable and uncontrollable events in P: the controller can only act over controllable event in order to satisfy the property Φ. Moreover the notion of *supervision* $C(P)$ can be seen as a simple synchronization function between the two automata. Finally note that we can consider different kinds of controller: the sampled controllers (performing an action every Δ time units — Δ being fixed), or the more general case of the dense-time controllers (we just require that at least Δ time units elapse between two controllable actions).

Note that synthesizing C — when it exists — is closely related to the satisfiability problem and it is an open problem when P, E and C are TA and $\Phi \in L_\nu$.

Nevertheless it has been shown in [BCL05a] that when a control objective belongs to a *deterministic* fragment of L_ν, it is possible to reduce the controllability problem — the existence of a controller — to a model checking problem. More precisely, given P and Φ, the existence of a controller for P and Φ can be reduced to a model checking instance $(P|A_\Delta) \models \Phi'$ where:

- Φ' is an L_ν^+ formula that can be constructed automatically from Φ,
- A_Δ is a simple timed automaton describing the type of the controller (sampled, or dense-time).

If Φ' holds for $(P|A_\Delta)$, then we know that there exists a strategy for choosing controllable actions in P in order to satisfy Φ. This strategy — i.e. the controller — is a timed transition system but it may be non definable as a timed automaton: a TTS is much more general.

Acknowledgments. Special thanks to Luca Aceto, Béatrice Bérard, Patricia Bouyer, Franck Cassez and Kim Guldstrand Larsen for all the discussions about timed automata and timed modal logics.

References

[ACD93] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AFH96] R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.

[AH92] R. Alur and T. A. Henzinger. Logics and models of real time: A survey. In *Real-Time: Theory in Practice, Proc. REX Workshop, Mook, NL, June 1991*, volume 600 of *Lecture Notes in Computer Science*, pages 74–106. Springer, 1992.

[AH94] R. Alur and T. A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–203, 1994.

[AL02] L. Aceto and F. Laroussinie. Is your model checker on time? On the complexity of model checking for timed modal logics. *Journal of Logic and Algebraic Programming*, 52-53:7–51, August 2002.

[And95] H. R. Andersen. Partial model checking (extended abstract). In *Proc. 10th IEEE Symp. Logic in Computer Science (LICS '95), San Diego, CA, USA, June 1995*, pages 398–407. IEEE Comp. Soc. Press, 1995.

[BCL05a] P. Bouyer, F. Cassez, and F. Laroussinie. Modal logics for timed control. In M. Abadi and L. de Alfaro, editors, *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR'05)*, Lecture Notes in Computer Science, San Francisco, CA, USA, August 2005. Springer. To appear.

[BCL05b] P. Bouyer, F. Cassez, and F. Laroussinie. Modal logics for timed control. Research Report LSV-05-04, Laboratoire Spécification et Vérification, ENS Cachan, France, April 2005. 23 pages.

[BCM05] P. Bouyer, F. Chevalier, and N. Markey. On the expressiveness of tptl and mtl. Technical Report 05, Laboratoire Specification et Vérification, May 2005.

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[CL00] F. Cassez and F. Laroussinie. Model-checking for hybrid systems by quotienting and constraints solving. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of the 12th International Conference on Computer Aided Verification (CAV 2000)*, volume 1855 of *Lecture Notes in Computer Science*, pages 373–388, Chicago, Illinois, USA, July 2000. Springer.

[Eme90] E. A. Emerson. Temporal and modal logic. In J. v. Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 16, pages 995–1072. Elsevier Science, 1990.

[EMSS91] E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. In *Proc. 2nd Int. Workshop Computer-Aided Verification (CAV '90), New Brunswick, NJ, USA, June 1990*, volume 531 of *Lecture Notes in Computer Science*, pages 136–145. Springer, 1991.

[HNSY94] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[LL95] F. Laroussinie and K. G. Larsen. Compositional model-checking of real time systems. In I. Lee and S. A. Smolka, editors, *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR'95)*, volume 962 of *Lecture Notes in Computer Science*, pages 529–539, Philadelphia, Pennsylvania, USA, August 1995. Springer.

[LL98] F. Laroussinie and K. G. Larsen. CMC: A tool for compositional model-checking of real-time systems. In S. Budkowski, A. R. Cavalli, and E. Najm, editors, *Proceedings of IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE'XI) and Protocol Specification, Testing and Verification (PSTV'XVIII)*, volume 135 of *IFIP Conference Proceedings*, pages 439–456, Paris, France, November 1998. Kluwer Academic Publishers.

[LLW95] F. Laroussinie, K. G. Larsen, and C. Weise. From timed automata to logic – and back. In J. Wiedermann and P. Hájek, editors, *Proceedings of the 20th International Symposium on Mathematical Fundations of Computer Science (MFCS'95)*, volume 969 of *Lecture Notes in Computer Science*, pages 27–41, Prague, Czech Republic, August 1995. Springer.

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Journal of Software Tools for Technology Transfer*, 1(1–2):134–152, 1997.

[Mil89] R. Milner. A complete axiomatisation for observational congruence of finite-state behaviours. *Information and Computation*, 81(2):227–247, 1989.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1992.

[OW05] J. Ouaknine and J. Worrell. On the decidability of metric temporal logic. In *Proc. 20th IEEE Symp. Logic in Computer Science (LICS 2005), Chicago, IL, USA, June 2005*, 2005.

[Par81] D. Park. Concurrency and automata on infinite sequences. In *Proc. 5th GI Conf. on Theor. Comp. Sci., Karlsruhe, FRG, Mar. 1981*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 1981.

[Yov97] S. Yovine. Kronos: A verification tool for real-time systems. *Journal of Software Tools for Technology Transfer*, 1(1–2):123–133, 1997.