Interval-based Temporal Reasoning with General TBoxes

Carsten Lutz
LuFG Theoretical Computer Science
RWTH Aachen, Germany
lutz@cs.rwth-aachen.de

Abstract

Until now, interval-based temporal Description Logics (DLs) did—if at all—only admit TBoxes of a very restricted form, namely acyclic macro definitions. In this paper, we present a temporal DL that overcomes this deficieny and combines interval-based temporal reasoning with general TBoxes. We argue that this combination is very interesting for many application domains. An automata-based decision procedure is devised and a tight EXPTIME-complexity bound is obtained. Since the presented logic can be viewed as being equipped with a concrete domain, our results can be seen from a different perspective: we show that there exist interesting concrete domains for which reasoning with general TBoxes is decidable.

1 Motivation

Description Logics (DLs) are a family of formalisms well-suited for the representation of and reasoning about conceptual knowledge. Whereas most Description Logics represent only static aspects of the application domain, recent research resulted in the exploration of various Description Logics that allow to, additionally, represent temporal information, see, e.g., [Artale and Franconi,2000] for an overview. One approach for temporal reasoning with DLs is to use so-called concrete domains. Concrete domains have been proposed as an extension of Description Logics that allows reasoning about "concrete qualities" of entities of the application domain such as sizes, weights or temperatures [Baader and Hanschke,1991]. As was first described in [Lutz *et al.*,1997], if a "temporal" concrete domain is employed, then Description Logics with concrete domains are a very useful tool for temporal reasoning. Ontologically, temporal reasoning with concrete domains is usually interval-based but may also be point-based or even both.

In this paper, we define a temporal Description Logic based on concrete domains which uses points as its basic temporal entity, but which may also be used as a full-fledged interval-based temporal DL. More precisely, the presented logic $\mathcal{TDL}$ extends the basic Description Logic $\mathcal{ALC}$ with a concrete domain that is based on the rationals and predicates $<$ and $=$. The well-known Allen relations can be defined in terms of their endpoints [Allen,1983] thus allowing for (qualitative) interval-based temporal reasoning. Since it is an important feature of DLs that reasoning should be decidable, we prove decidability of the standard reasoning tasks by using an automata-theoretic approach which also yields a tight EXPTIME complexity bound.

Most DLs allow for some kind of TBox formalism that is used to represent terminological knowledge as well as background knowledge about the application domain. However, there exist various flavours of TBoxes with vast differences in expressivity. To the best of our knowledge, all interval-based DLs and all DLs with concrete domains defined in the literature admit only a very restricted form of TBox, i.e., sets of acyclic macro definitions. Compared to existing Description Logics that are interval-based or include concrete domains, the distinguishing feature of our logic is that it is equipped with a very general form of TBoxes that allows arbitrary equations over concepts. Thus, the presented work overcomes a major limitation of both families of Description Logics.

Our results can be viewed from the perspective of interval-based temporal reasoning and from the perspective of concrete domains. For the temporal perspective, we claim that the combination of general TBoxes and interval-based temporal reasoning is important for many application areas. In this paper, we present process engineering as an example. From the concrete domain perspective, our results can be viewed as follows: in [Lutz,2001], it is shown that, even for very simple concrete domains, reasoning with general TBoxes is undecidable. It was an open question whether there exist interesting concrete domains for which reasoning with general TBoxes is decidable. In this paper, we answer this question to the affirmative. This paper is accompanied by a technical report containing full proofs [Lutz,2000].

2 Syntax and Semantics

In this section, we introduce syntax and semantics of the Description Logic $\mathcal{TDL}$.

Definition 1. Let N_C, N_R, and N_{cF} be mutually disjoint and countably infinite sets of *concept names*, *roles*, and *concrete features*. Furthermore, let N_{aF} be a countably infinite subset of N_R. The elements of N_{aF} are called *abstract features*. A *path* u is a composition $f_1 \cdots f_n g$ of $n \geq 0$ abstract fea-

tures $f_1, \ldots, f_n$ and one concrete feature g. The set of $\mathcal{TDL}$-concepts is the smallest set such that

1. every concept name is a concept
2. if C and D are concepts, R is a role, g is a concrete feature, u_1, u_2 are paths, and $P \in \{<, =\}$, then the following expressions are also concepts: $\neg C$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$, $\forall R.C$, $\exists u_1, u_2.P$, and $g\uparrow$.

A *TBox axiom* is an expression of the form $C \sqsubseteq D$ with C and D are concepts. A finite set of TBox axioms is a *TBox*.

Throughout this paper, we will denote atomic concepts by the letter A, (possibly complex) concepts by the letters C, D, E, roles by the letter R, abstract features by the letter f, concrete features by the letter g, paths by the letter u, and elements of the set $\{<, =\}$ by the letter P. We will sometimes call the TBox formalism introduced above *general TBoxes* to distinguish it from other, weaker formalisms such as the ones in [Nebel,1990]. As most Description Logics, $\mathcal{TDL}$ is equipped with a Tarski-style semantics.

Definition 2. An *interpretation* $\mathcal{I}$ is a pair $(\Delta_\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta_\mathcal{I}$ is a set called the *domain* and $\cdot^\mathcal{I}$ is the *interpretation function* mapping each concept name C to a subset $C^\mathcal{I}$ of $\Delta_\mathcal{I}$, each role name R to a subset $R^\mathcal{I}$ of $\Delta_\mathcal{I} \times \Delta_\mathcal{I}$, each abstract feature f to a partial function $f^\mathcal{I}$ from $\Delta_\mathcal{I}$ to $\Delta_\mathcal{I}$, and each concrete feature g to a partial function $g^\mathcal{I}$ from $\Delta_\mathcal{I}$ to the rationals $\mathbb{Q}$. For paths $u = f_1 \cdots f_n g$, we set $u^\mathcal{I}(a) := g^\mathcal{I}(f_n^\mathcal{I}(\cdots(f_1^\mathcal{I}(a))\cdots))$. The interpretation function is extended to arbitrary concepts as follows:

$$
\begin{aligned}
(\neg C)^\mathcal{I} &:= \Delta_\mathcal{I} \setminus C^\mathcal{I} \\
(C \sqcap D)^\mathcal{I} &:= C^\mathcal{I} \cap D^\mathcal{I} \qquad (C \sqcup D)^\mathcal{I} := C^\mathcal{I} \cup D^\mathcal{I} \\
(\exists R.C)^\mathcal{I} &:= \{a \in \Delta_\mathcal{I} \mid \{b \mid (a,b) \in R^\mathcal{I}\} \cap C^\mathcal{I} \neq \emptyset\} \\
(\forall R.C)^\mathcal{I} &:= \{a \in \Delta_\mathcal{I} \mid \{b \mid (a,b) \in R^\mathcal{I}\} \subseteq C^\mathcal{I}\} \\
(\exists u_1, u_2.P)^\mathcal{I} &:= \{a \in \Delta_\mathcal{I} \mid \exists q_1, q_2 \in \mathbb{Q} : u_1^\mathcal{I}(a) = q_1, \\
&\qquad\qquad u_2^\mathcal{I}(a) = q_2, q_1 P q_2\} \\
(g\uparrow)^\mathcal{I} &:= \{a \in \Delta_\mathcal{I} \mid g^\mathcal{I}(a) \text{ is undefined}\}
\end{aligned}
$$

An interpretation $\mathcal{I}$ is a *model* of a TBox $\mathcal{T}$ iff it satisfies $C^\mathcal{I} \subseteq D^\mathcal{I}$ for all axioms $C \sqsubseteq D$ in $\mathcal{T}$. $\mathcal{I}$ is a *model* of a concept C iff $C^\mathcal{I} \neq \emptyset$.

If $g(a) = x$ for some $g \in N_{cF}$, $a \in \Delta_\mathcal{I}$, and $x \in \mathbb{Q}$, then we call x a *concrete successor* of a in $\mathcal{I}$. We write $\top$ for $A \sqcup \neg A$ and $\bot$ for $\neg\top$, where A is a concept name. Moreover, we write $u\uparrow$ with $u = f_1 \cdots f_k g$ for $\forall f_1 . \cdots \forall f_k . g\uparrow$.

Definition 3 (Inference Problems). Let C and D be concepts and $\mathcal{T}$ be a TBox. C *subsumes* D *w.r.t.* $\mathcal{T}$ (written $D \sqsubseteq_\mathcal{T} C$) iff $D^\mathcal{I} \subseteq C^\mathcal{I}$ for all models $\mathcal{I}$ of $\mathcal{T}$. C is *satisfiable w.r.t.* $\mathcal{T}$ iff there exists a model of both $\mathcal{T}$ and C.

It is well-known that (un)satisfiability and subsumption can be mutually reduced to each other: $C \sqsubseteq_\mathcal{T} D$ iff $C \sqcap \neg D$ is unsatisfiable w.r.t. $\mathcal{T}$ and C is satisfiable w.r.t. $\mathcal{T}$ iff $C \not\sqsubseteq_\mathcal{T} \bot$.

We now discuss the relationship between $\mathcal{TDL}$ and Description Logics with concrete domains.

Definition 4 (Concrete Domain). A *concrete domain* $\mathcal{D}$ is a pair $(\Delta_\mathcal{D}, \Phi_\mathcal{D})$, where $\Delta_\mathcal{D}$ is a set called the domain, and $\Phi_\mathcal{D}$ is a set of predicate names. Each predicate name $P \in \Phi_\mathcal{D}$ is associated with an arity n and an n-ary predicate $P^\mathcal{D} \subseteq \Delta_\mathcal{D}^n$.

The concrete domain is usually integrated into the logic by a concept constructor $\exists u_1, \ldots, u_n.P$, with semantics

$$
(\exists u_1, \ldots, u_n.P)^\mathcal{I} := \{a \in \Delta_\mathcal{I} \mid u_i^\mathcal{I}(a) = q_i \text{ for } 1 \leq i \leq n \text{ and } (q_1, \ldots, q_n) \in P^\mathcal{D}\}.
$$

It is obvious that $\mathcal{TDL}$ can be viewed as being equipped with the concrete domain $\mathcal{D}_< := (\mathbb{Q}, \{<, =\})$, where $<$ and $=$ are binary predicates with the usual semantics. For most DLs with concrete domains, it is required that the set of predicates is closed under negation and contains a name $\top_\mathcal{D}$ for $\Delta_\mathcal{D}$. This property ensures that every concept can be converted into an equivalent one in the so-called negation normal form (NNF). The NNF of concepts, in turn, is used as a starting point for devising satisfiability algorithms. It is not hard to see that $\mathcal{D}_<$ is not admissible in this sense. However, as we will see in Section 4, the conversion of $\mathcal{TDL}$-concepts into equivalent ones in NNF is nevertheless possible.

3 Temporal Reasoning with $\mathcal{TDL}$

Although $\mathcal{TDL}$ does only provide the relations "$=$" and "$<$" on time points, it is not hard to see that the remaining relations can be defined by writing, e.g., $\exists u_2, u_1.< \sqcup \exists u_1, u_2.=$ for $\exists u_1, u_2.\geq$. However, we claim that $\mathcal{TDL}$ cannot only be used for point-based temporal reasoning but also as a full-fledged interval-based temporal Description Logic. As observed by Allen [1983], there are 13 possible relationships between two intervals such as, for example, the meets relation: two intervals i_1 and i_2 are related by meets iff the right endpoint of i_1 is identical to the left endpoint of i_2—see [Allen,1983] for an exact definition of the other relations. As we shall see, these 13 Allen relations (as well as the additional relations from the Allen algebra, see [Allen,1983]) can be defined in $\mathcal{TDL}$. In the following, we present a framework for mixed interval- and point-based reasoning in $\mathcal{TDL}$ and apply this framework in the application area of process engineering

The representation framework consists of several conventions and abbreviations. We assume that each entity of the application domain is either temporal or atemporal. If it is temporal, its temporal extension may be either a time point or an interval. Left endpoints of intervals are represented by the concrete feature ℓ, right endpoints of intervals are represented by the concrete feature r, and time-points not related to intervals are represented by the concrete feature t. All this can be expressed by the following TBox $\mathcal{T}^*$:

$$
\mathsf{Point} \doteq \exists t, t.= \sqcap \ell\uparrow \sqcap r\uparrow \quad \mathsf{Interval} \doteq \exists \ell, r.< \sqcap t\uparrow
$$
$$
\mathsf{ATemporal} \doteq t\uparrow \sqcap \ell\uparrow \sqcap r\uparrow \quad \mathsf{Interval} \sqsupseteq \exists \ell, \ell, = \sqcup \exists r, r.=
$$

Here, $C \doteq D$ is an abbreviation for $\{C \sqsubseteq D, D \sqsubseteq C\}$. Let us now define the Allen relations as abbreviations. For example, $\exists(F, F').\mathsf{contains}$ is an abbreviation for $\exists F\ell, F'\ell.< \sqcap \exists F'r, Fr.<$ where $F, F' \in (N_{aF})^*$, i.e., F and F' are words over the alphabet N_{aF}. Note that

$$
\exists(F, F').\mathsf{contains} \sqsubseteq_{\mathcal{T}^*} \exists F.\mathsf{Interval} \sqcap \exists F'.\mathsf{Interval}.
$$

Similar abbreviations are introduced for the other Allen relations. We use self to denote the empty word. For example,

$\exists(F, \mathsf{self}).\mathsf{starts}$ is an abbreviation for $\exists F\ell, \ell.= \sqcap \exists Fr, r.<$. Intuitively, self refers to the interval associated with the abstract object at which the $\exists(F, \mathsf{self}).\mathsf{starts}$ concept is "evaluated".

Since we have intervals *and* points available, we should also be able to talk about the relationship of points and intervals. More precisely, there exist 5 possible relations between a point and an interval [Vilain,1982], one example being startsp that holds between a point p and an interval i if p is identical to the left endpoint of i. Hence, we can define $\exists(Ft, F').\mathsf{startsp}$ as an abbreviation for $\exists Ft, F'\ell.=$ and similar abbreviations for beforep, duringp, finishesp, and afterp.

This finishes the definition of the framework. We claim that the combination of interval-based reasoning and general TBoxes is important for many application areas such as reasoning about action and plans [Artale and Franconi,2000]. The examples presented here are from the area of process engineering that was first considered by Sattler in a DL context [Sattler,1998]. However, Sattler's approach does not take into account temporal aspects of the application domain. We show how this can be done using $\mathcal{TDL}$ thus refining Sattler's proposal.

Assume that our goal is to represent information about an automated chemical production process that is carried out by some complex technical device. The device operates each day for some time depending on the number of orders. It needs a complex startup and shutdown process before resp. after operation. Moreover, some weekly maintenance is needed to keep the device functional. Let us first represent the underlying temporal structure that consists of weeks and days.

$$\begin{aligned}\mathsf{Week} \doteq \mathsf{Interval} \sqcap \prod_{1\le i\le 7} \exists \mathsf{day}_i.\mathsf{Day} \sqcap \\ \exists(\mathsf{day}_1, \mathsf{self}).\mathsf{starts} \sqcap \exists(\mathsf{day}_7, \mathsf{self}).\mathsf{finishes} \sqcap \\ \prod_{1\le i<7} \exists(\mathsf{day}_i, \mathsf{day}_{i+1}).\mathsf{meets} \sqcap \\ \exists \mathsf{next}.\mathsf{Week} \sqcap \exists(\mathsf{self}, \mathsf{next}).\mathsf{meets}\end{aligned}$$

The axiom states that each week consists of seven days, where the i'th day is accessible from the corresponding week via the abstract feature day_i. The temporal relationship between the days are as expected: Monday starts the week, Sunday finishes it, and each day temporally meets the succeeding one. Note that this implies that days 2 to 6 are during the corresponding week although this is not explicitly stated. Moreover, each week has a successor week that it temporally meets. We now describe the startup, operation, shutdown, and maintenance phases.

$$\begin{aligned}\mathsf{Day} \doteq \mathsf{Interval} \sqcap \\ \exists \mathsf{start}.\mathsf{Startup} \sqcap \exists \mathsf{op}.\mathsf{Operation} \sqcap \exists \mathsf{shut}.\mathsf{Shutdn} \sqcap \\ \exists \mathsf{start} \circ \ell, \ell.\ge \sqcap \exists(\mathsf{start}, \mathsf{op}).\mathsf{meets} \sqcap \\ \exists(\mathsf{op}, \mathsf{shut}).\mathsf{meets} \sqcap \exists \mathsf{shut} \circ r, r.\le \\ \mathsf{Week} \sqsubseteq \exists \mathsf{maint}.\mathsf{Maintenance} \sqcap \exists(\mathsf{self}, \mathsf{maint}).\mathsf{contains}\end{aligned}$$

Here start, op, shut, and maint are abstract features and "$\circ$" is used for better readability (i.e., paths $f_1 \cdots f_k g$ are written as $f_1 \circ \cdots \circ f_k \circ g$). The TBox implies that phases are related to the corresponding day as follows: startup via starts or during, shutdown via during or finishes, and operation via during. Until now, we did not say anything about the temporal relationship of maintenance and operation. This may be inadequate, if, for example, maintenance and operation are mutually exclusive. We can take this into account by using additional axioms

$$\mathsf{Week} \sqcap \bigsqcup_{1\le i\le 7} \exists(\mathsf{maint}, \mathsf{day}_i \circ \mathsf{op}).\mathsf{OVLP} \sqsubseteq \bot \qquad (*)$$

where OVLP is replaced by equal, overlaps, overlapped-by, during, contains, starts, started-by, finishes, or finished-by yielding 9 axioms.

Until now, we have modelled the very basic properties of our production process. Let us define some more advanced concepts to illustrate reasoning with $\mathcal{TDL}$. For example, we could define a busy week as

$$\begin{aligned}\mathsf{BusyWeek} \doteq \mathsf{Week} \sqcap \prod_{1\le i\le 7} \big(\exists \mathsf{day}_i \circ \mathsf{start}, \mathsf{day}_i.\mathsf{starts} \sqcap \\ \exists \mathsf{day}_i \circ \mathsf{shut}, \mathsf{day}_i.\mathsf{finishes}\big)\end{aligned}$$

i.e., each day, the startup process starts at the beginning of the day and the shutdown finishes at the end of the day. Say now that it is risky to do maintenance during startup and shutdown phases and define

$$\begin{aligned}\mathsf{RiskyWeek} \doteq \mathsf{Week} \sqcap \neg \prod_{1\le i\le 7} \big(\exists \mathsf{maint}, \mathsf{day}_i \circ \mathsf{start}.\mathsf{before} \sqcup \\ \exists \mathsf{maint}, \mathsf{day}_i \circ \mathsf{shut}.\mathsf{after}\big)\end{aligned}$$

expressing that, in a risky week, the maintenance phase is not strictly separated from the startup and shutdown phases. A $\mathcal{TDL}$ reasoner could be used to detect that $\mathsf{BusyWeek} \sqsubseteq \mathsf{RiskyWeek}$, i.e., every busy week is a risky week: in a busy week, the week is partitioned into startup, shutdown, and operation phases. Since maintenance may not OVLP with operation phases (see $(*)$), it must OVLP with startup and/or shutdown phases which means that it is a risky week. We can further refine this model by using mixed point-based and interval-based reasoning, see [Lutz,2000] for examples.

4 The Decision Procedure

In this section, we prove satisfiability of $\mathcal{TDL}$-concepts w.r.t. TBoxes to be decidable and obtain a tight EXPTIME complexity bound. This is done using an automata-theoretic approach: first, we abstract models to so-called Hintikka-trees such that there exists a model for a concept C and a TBox $\mathcal{T}$ iff there exists a Hintikka-tree for C and $\mathcal{T}$. Then, we build, for each $\mathcal{TDL}$-concept C and TBox $\mathcal{T}$, a looping automaton $\mathcal{A}_{(C,\mathcal{T})}$ that accepts exactly the Hintikka-trees for $(C, \mathcal{T})$. In particular, this implies that $\mathcal{A}_{(C,\mathcal{T})}$ accepts the empty language iff C is unsatisfiable w.r.t. $\mathcal{T}$.

Definition 5. Let M be a set and $k \ge 1$. A *k-ary M-tree* is a mapping $T : \{1, \ldots, k\}^* \to M$ that labels each node $\alpha \in \{1, \ldots, k\}^*$ with $T(\alpha) \in M$. Intuitively, the node αi is the i-th child of α. We use ϵ to denote the empty word (corresponding to the root of the tree).

A *looping automaton* $\mathcal{A} = (Q, M, I, \Delta)$ for k-ary M-trees is defined by a set Q of states, an alphabet M, a subset $I \subseteq Q$ of initial states, and a transition relation $\Delta \subseteq Q \times M \times Q^k$. A *run* of $\mathcal{A}$ on an M-tree T is a mapping $r : \{1, \ldots, k\}^* \to Q$ with $r(\epsilon) \in I$ and $(r(\alpha), T(\alpha), r(\alpha 1), \ldots, r(\alpha k)) \in \Delta$ for each $\alpha \in$

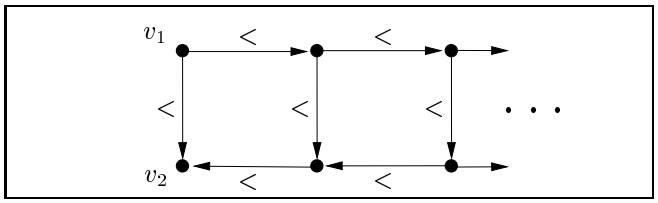

Figure 1: A constraint graph containing no $<$-cycle that is unsatisfiable over $\mathbb{N}$.

$\{1, \ldots, k\}^*$. A looping automaton accepts all those M-trees for which a run exists, i.e., the language $\mathcal{L}(\mathcal{A})$ of M-trees accepted by $\mathcal{A}$ is

$$L(\mathcal{A}) = \{T \mid \text{there is a run of } \mathcal{A} \text{ on } T\}.$$

In [Vardi and Wolper,1986], it is proved that the emptiness problem for looping automata is decidable in polynomial time.

A Hintikka-tree for C and $\mathcal{T}$ corresponds to a canonical model for C and $\mathcal{T}$. Apart from describing the abstract domain $\Delta_{\mathcal{I}}$ of the corresponding canonical model $\mathcal{I}$ together with the interpretation of concepts and roles, each Hintikka-tree induces a directed graph whose edges are labelled with predicates from $\{<, =\}$. These constraint graphs describe the "concrete part" of $\mathcal{I}$ (i.e., concrete successors of domain objects and their relationships).

Definition 6. A *constraint graph* is a pair $G = (V, E)$, where V is a countable set of *nodes* and $E \subseteq V \times V \times \{=, <\}$ a set of *edges*. We generally assume that constraint graphs are *equality closed*, i.e., that $(v_1, v_2, =) \in E$ implies $(v_2, v_1, =) \in E$. A constraint graph $G = (V, E)$ is called *satisfiable over* M, where M is a set equipped with a total ordering $<$, iff there exists a total mapping δ from V to M such that $\delta(v_1)\, P\, \delta(v_2)$ for all $(v_1, v_2, P) \in E$. In this case, δ is called a *solution* for G.

A *$<$-cycle* Q in G is a finite non-empty sequence of nodes $v_0, \ldots, v_{k-1} \in V$ such that (1) for all i with $i < k$, we have $(v_i, v_{i \oplus_k 1}, P) \in E$, where $P \in \{<, =\}$ and $\oplus_k$ denotes addition modulo k and (2) $(v_i, v_{i \oplus_k 1}, <) \in E$ for some $i < k$.

The following theorem will be crucial for proving that, for every Hintikka-tree, there exists a corresponding canonical model. More precisely, it will be used to ensure that the constraint graph induced by a Hintikka-tree, which describes the concrete part of the corresponding model, is satisfiable.

Theorem 7. *A constraint graph G is satisfiable over M with $M \in \{\mathbb{Q}, \mathbb{R}\}$ iff G does not contain a $<$-cycle.*

Note that Theorem 7 does *not* hold if satisfiability over $\mathbb{N}$ is considered due to the absence of density: if there exist two nodes v_1 and v_2 such that the length of $<$-paths (which are defined in the obvious way) between v_1 and v_2 is unbounded, a constraint graph is unsatisfiable over $\mathbb{N}$ even if it contains no $<$-cycle, see Figure 1.

The decidability procedure works on $\mathcal{TDL}$-concepts and TBoxes that are in a certain syntactic form. To define this normal form, we first introduce the well-known negation normal form.

Definition 8 (NNF). A concept C is in *negation normal form* (NNF) if negation occurs only in front of concept names. Every concept can be transformed into an equivalent one in NNF by eliminating double negation and using de Morgan's law, the duality between $\exists$ and $\forall$, and the following equivalences:

$$\begin{array}{rcl} \neg(\exists u_1, u_2.P) & \equiv & \exists u_1, u_2.\widetilde{P} \sqcup \exists u_2, u_1.< \sqcup\ u_1\uparrow \sqcup\ u_2\uparrow \\ \neg(g\uparrow) & \equiv & \exists g, g.= \end{array}$$

where $\widetilde{\cdot}$ denotes the exchange of predicates, i.e., $\widetilde{<}$ is $=$ and $\widetilde{=}$ is $<$. With $\mathsf{nnf}(C)$, we denote the equivalent of C in NNF. A TBox $\mathcal{T}$ is in NNF iff all concepts in $\mathcal{T}$ are in NNF.

We can now extend NNF to the so-called path normal form.

Definition 9 (Path Normal Form). A $\mathcal{TDL}$-concept C is in *path normal form* (PNF) iff it is in NNF, and, for all subconcepts $\exists u_1, u_2.P$ of C, we have either (1) $u_1 = g_1$ and $u_2 = g_2$, (2) $u_1 = fg_1$ and $u_2 = g_2$, or (3) $u_1 = g_1$ and $u_2 = fg_2$ for some $f \in N_{aF}$ and $g_1, g_2 \in N_{cF}$. A $\mathcal{TDL}$ TBox $\mathcal{T}$ is in path normal form iff it is in NNF and all concepts appearing in $\mathcal{T}$ are in path normal form.

Lemma 10. *Satisfiability of $\mathcal{TDL}$-concepts w.r.t. $\mathcal{TDL}$-TBoxes can be reduced to satisfiability of $\mathcal{TDL}$-concepts in PNF w.r.t. $\mathcal{TDL}$-TBoxes in PNF.*

Proof Let C be a $\mathcal{TDL}$-concept. For every path $u = f_1 \cdots f_n g$ in C, we assume that $[g], [f_n g], \ldots, [f_1 \cdots f_n g]$ are concrete features. We inductively define a mapping λ from paths u in C to concepts as follows:

$$\lambda(g) = \top \qquad \lambda(fu) = (\exists [fu], f[u].=) \sqcap \exists f.\lambda(u)$$

For every $\mathcal{TDL}$-concept C, the corresponding concept $\rho(C)$ is obtained by replacing all subconcepts $\exists u_1, u_2.P$ of C with $\exists [u_1], [u_2].P \sqcap \lambda(u_1) \sqcap \lambda(u_2)$ and $g\uparrow$ with $[g]\uparrow$. We extend the mapping ρ to TBoxes in the obvious way. Let C be a $\mathcal{TDL}$-concept and $\mathcal{T}$ a $\mathcal{TDL}$-TBox. By Definition 8, we may assume both C and $\mathcal{T}$ to be in NNF. It is now easy to check that the translation is polynomial and that C is satisfiable w.r.t. $\mathcal{T}$ iff $\rho(C)$ is satisfiable w.r.t. $\rho(\mathcal{T})$ (see [Lutz,2000]). ❑

Hence, it suffices to prove that satisfiability of concepts in PNF w.r.t. TBoxes in PNF is decidable. We often refer to TBoxes $\mathcal{T}$ in their *concept form* $C_{\mathcal{T}}$:

$$C_{\mathcal{T}} = \bigsqcap_{C \sqsubseteq D \in \mathcal{T}} \mathsf{nnf}(\neg C \sqcup D).$$

We now define Hintikka-trees for concepts C and TBoxes $\mathcal{T}$ (both in PNF) and show that there exists Hintikka-tree for C and $\mathcal{T}$ iff there exists a model for C and $\mathcal{T}$.

Let C be a concept and $\mathcal{T}$ a TBox. With $\mathsf{cl}(C, \mathcal{T})$, we denote the set of subconcepts of C and $C_{\mathcal{T}}$. We assume that existential concepts $\exists R.D$ in $\mathsf{cl}(C, \mathcal{T})$ with $R \in N_R \setminus N_{aF}$ are linearly ordered, and that $\mathcal{E}(C, \mathcal{T}, i)$ yields the i-th existential concept in $\mathsf{cl}(C, \mathcal{T})$. Furthermore, we assume the abstract features used in $\mathsf{cl}(C, \mathcal{T})$ to be linearly ordered and use $\mathcal{F}(C, \mathcal{T}, i)$ to denote the i-th abstract feature in $\mathsf{cl}(C, \mathcal{T})$. The set of concrete features used in $\mathsf{cl}(C, \mathcal{T})$ is denoted with $\mathcal{G}(C, \mathcal{T})$. Hintikka-pairs are used as labels of the nodes in Hintikka-trees.

Definition 11 (Hintikka-set, Hintikka-pair). Let C be a concept and $\mathcal{T}$ be a TBox. A set $\Psi \subseteq \mathsf{cl}(C, \mathcal{T})$ is a *Hintikka-set for* $(C, \mathcal{T})$ iff it satisfies the following conditions:

(H1) $C_{\mathcal{T}} \in \Psi$,

(H2) if $C_1 \sqcap C_2 \in \Psi$, then $\{C_1, C_2\} \subseteq \Psi$,

(H3) if $C_1 \sqcup C_2 \in \Psi$, then $\{C_1, C_2\} \cap \Psi \neq \emptyset$,

(H4) $\{A, \neg A\} \not\subseteq \Psi$ for all concept names $A \in \mathsf{cl}(C, \mathcal{T})$,

(H5) if $g\!\uparrow \in \Psi$, then $\exists u_1, u_2.P \notin \Psi$ for all concepts $\exists u_1, u_2.P$ with $u_1 = g$ or $u_2 = g$.

We say that $f \in N_{aF}$ is *enforced* by a Hintikka-set Ψ iff either $\exists f.C \in \Psi$ for some concept C or $\{\exists f g_1, g_2.P, \exists g_1, f g_2.P\} \cap \Psi \neq \emptyset$ for some $g_1, g_2 \in N_{cF}$ and $P \in \{<, =\}$. A *Hintikka-pair* (Ψ, χ) *for* $(C, \mathcal{T})$ consists of a Hintikka-set Ψ for $(C, \mathcal{T})$ and a set χ of tuples (g_1, g_2, P) with $g_1, g_2 \in \mathcal{G}(C, \mathcal{T})$ such that

(H6) if $(g_1, g_2, P) \in \chi$, then $\{g_1\!\uparrow, g_2\!\uparrow\} \cap \Psi = \emptyset$.

With $\Gamma_{(C,\mathcal{T})}$, we denote the set of all Hintikka-pairs for $(C, \mathcal{T})$. A path u (of length 1 or 2) is *enforced* by (Ψ, χ) iff either u is a node in χ or $\{\exists u, u'.P, \exists u', u.P\} \cap \Psi \neq \emptyset$ for some path u' and $P \in \{<, =\}$.

Intuitively, each node α of a (yet to be defined) Hintikka-tree T corresponds to a domain object a of the corresponding canonical model $\mathcal{I}$. The first component Ψ_α of the Hintikka-pair labelling α is the set of concepts from $\mathsf{cl}(C, \mathcal{T})$ satisfied by a. The second component χ_α states restrictions on the relationship between concrete successors of a. If, for example, $(g_1, g_2, <) \in \chi_\alpha$, then we must have $g_1^{\mathcal{I}}(a) < g_2^{\mathcal{I}}(a)$. Note that the restrictions in χ_α are independent from concepts $\exists g_1, g_2.P \in \Psi_\alpha$. As will become clear when Hintikka-trees are defined, the restrictions in χ_α are used to ensure that the constraint graph induced by the Hintikka-tree T, which describes the concrete part of the model $\mathcal{I}$, does not contain a $<$-cycle, i.e., that it is satisfiable. This induced constraint graph can be thought of as the union of smaller constraint graphs, each one being described by a Hintikka-pair labelling a node in T. These pair-graphs are defined next.

Definition 12 (Pair-graph). Let C be a concept, $\mathcal{T}$ a TBox, and $p = (\Psi, \chi)$ a Hintikka-pair for $(C, \mathcal{T})$. The *pair-graph* $G(p) = (V, E)$ of p is a constraint graph defined as follows:

1. V is the set of paths enforced by p
2. $E = \chi \cup \{(u_1, u_2, P) \mid \exists u_1, u_2.P \in \Psi\}$.

An *edge extension* of $G(p)$ is a set $E' \subseteq V \times V \times \{<, =\}$ such that for all $f g_1, f g_2 \in V$, we have either $(f g_2, f g_1, <) \in E'$ or $(f g_1, f g_2, P) \in E'$ for some $P \in \{<, =\}$. If E' is an edge extension of $G(p)$, then the graph $(V, E \cup E')$ is a *completion* of $G(p)$.

Note that, due to path normal form and the definitions of Hintikka-pairs and pair-graphs, we have $E' \cap E = \emptyset$ for every edge extension E' of a pair-graph (V, E).

We briefly comment on the connection of completions and the χ-component of Hintikka-pairs. Let α and β be nodes in a Hintikka-tree T and let a and b be the corresponding domain objects in the corresponding canonical model $\mathcal{I}$. Edges in Hintikka-trees represent role-relationships, i.e., if β is successor of α in T, then there exists an $R \in N_R$ such that $(a, b) \in R^{\mathcal{I}}$. Assume β is successor of α and the edge between α and β represents relationship via the abstract feature f, i.e., we have $f^{\mathcal{I}}(a) = b$. The second component χ_β of the Hintikka-pair labelling β fixes the relationships between all concrete successors of b that "a talks about". For example, if $(\exists f g_1, g_2.=) \in \Psi_\alpha$ and $(\exists f g_3, g_2.<) \in \Psi_\alpha$, where Ψ_α is the first component of the Hintikka-pair labelling α, then "a talks about" the concrete g_1-successor and the concrete g_3-successor of b. Hence, χ_β either contains $(g_3, g_1, <)$ or (g_1, g_3, P) for some $P \in \{<, =\}$. This is formalized by demanding that the pair-graph $G(T(\alpha))$ of the Hintikka-pair labelling α together with all the edges from the χ-components of the successors of α are a completion of $G(T(\alpha))$. Moreover, this completion has to be satisfiable, which is necessary to ensure that the constraint graph induced by T does not contain a $<$-cycle. An appropriate way of thinking about the χ-components is as follows: at α, a completion of $G(T(\alpha))$ is "guessed". The additional edges are then "recorded" in the χ-components of the successor-nodes of α.

Definition 13 (Hintikka-tree). Let C be a concept, $\mathcal{T}$ be a TBox, k the number of existential subconcepts in $\mathsf{cl}(C, \mathcal{T})$, and ℓ be the number of abstract features in $\mathsf{cl}(C, \mathcal{T})$. A $1+k+\ell$-tuple of Hintikka-pairs $(p_0, \ldots, p_{k+\ell})$ with $p_i = (\Psi_i, \chi_i)$ and $G(p_0) = (V, E)$ is called *matching* iff

(H7) if $\exists R.D \in \Psi_0$ and $\mathcal{E}(C, \mathcal{T}, i) = \exists R.D$, then $D \in \Psi_i$

(H8) if $\{\exists R.D, \forall R.E\} \subseteq \Psi_0$ and $\mathcal{E}(C, \mathcal{T}, i) = \exists R.D$, then $E \in \Psi_i$

(H9) if $\exists f.D \in \Psi_0$ and $\mathcal{F}(C, \mathcal{T}, i) = f$, then $D \in \Psi_{k+i}$.

(H10) if f is enforced by Ψ_0, $\mathcal{F}(C, \mathcal{T}, i) = f$, and $\forall f.D \in \Psi_0$, then $D \in \Psi_{k+i}$.

(H11) the constraint graph $(V, E \cup E')$ is a satisfiable completion of $G(p_0)$, where E' is defined as

$$\bigcup_{1 \leq i \leq \ell} \{(f g_1, f g_2, P) \mid \mathcal{F}(C, \mathcal{T}, i) = f, (g_1, g_2, P) \in \chi_{k+i}\}.$$

A $k + \ell$-ary $\Gamma_{(C,\mathcal{T})}$-tree T is a *Hintikka-tree for* $(C, \mathcal{T})$ iff $T(\alpha)$ is a Hintikka-pair for $(C, \mathcal{T})$ for each node α in T, and T satisfies the following conditions:

(H12) $C \in \Psi_\epsilon$, where $T(\epsilon) = (\Psi_\epsilon, \chi_\epsilon)$,

(H13) for all $\alpha \in \{1, \ldots, k + \ell\}^*$, the tuple $(T(\alpha), T(\alpha 1), \ldots, T(\alpha j))$ with $j = k + \ell$ is matching.

For a Hintikka-tree T and node $\alpha \in \{1, \ldots, k + \ell\}^*$ with $T(\alpha) = (\Psi, \chi)$, we use $T_\lhd(\alpha)$ to denote Ψ and $T_\rhd(\alpha)$ to denote χ. Moreover, if $G(\alpha) = (V, E)$, we use $\mathsf{cpl}(T, \alpha)$ to denote the constraint graph $(V, E \cup E')$ as defined in **(H11)**.

Whereas most properties of Hintikka-trees deal with concepts, roles, and abstract features and are hardly surprising, **(H11)** ensures that constraint graphs induced by Hintikka-trees contain no $<$-cycle. By "guessing" a completion as explained above, possible $<$-cycles are anticipated and can be detected locally, i.e., it suffices to check that the completions $\mathsf{cpl}(T, \alpha)$ are satisfiable as demanded by **(H11)**. Indeed, it is crucial that the cycle detection is done by a *local* condition since we need to define an automaton which accepts exactly Hintikka-trees and automata work locally. It is worth noting that the localization of cycle detection as expressed by **(H11)** crucially depends on path normal form.

The following two lemmas show that Hintikka-trees are appropriate abstractions of models.

Lemma 14. *A concept C is satisfiable w.r.t. a TBox $\mathcal{T}$ iff there exists a Hintikka-tree for $(C, \mathcal{T})$.*

To prove decidability, it remains to define a looping automaton $\mathcal{A}_{(C,\mathcal{T})}$ for each concept C and TBox $\mathcal{T}$ such that $\mathcal{A}_{(C,\mathcal{T})}$ accepts exactly the Hintikka-trees for $(C, \mathcal{T})$.

Definition 15. Let C be a concept, $\mathcal{T}$ be a TBox, k the number of existential subconcepts in $\mathsf{cl}(C, \mathcal{T})$, and ℓ be the number of abstract features in $\mathsf{cl}(C, \mathcal{T})$. The looping automaton $\mathcal{A}_{(C,\mathcal{T})} = (Q, \Gamma_{(C,\mathcal{T})}, \Delta, I)$ is defined as follows:

- $Q = \Gamma_{(C,\mathcal{T})}$, $I = \{(\Psi, \chi) \in Q \mid C \in \Psi\}$, and
- $((\Psi, \chi), (\Psi', \chi'), (\Psi_1, \chi_1), \ldots, (\Psi_k, \chi_{k+\ell})) \in \Delta$ iff
  $(\Psi, \chi) = (\Psi', \chi')$ and
  $((\Psi, \chi), (\Psi_1, \chi_1), \ldots, (\Psi_k, \chi_{k+\ell}))$ is matching.

Note that every state is an accepting state, and, hence, every run is accepting. The following lemma is easily obtained.

Lemma 16. *T is Hintikka-tree for $(C, \mathcal{T})$ iff $T \in L(\mathcal{A}_{C,\mathcal{T}})$.*

Since the size of $\mathsf{cl}(C, \mathcal{T})$ is linear in the size of C and $\mathcal{T}$, it is straightforward to verify that the size of $\mathcal{A}_{(C,\mathcal{T})}$ is exponential in the size of C and $\mathcal{T}$. This, together with Lemmas 10, 14, and 16, and the polynomial decidability of the emptiness problem of looping automata [Vardi and Wolper,1986], implies the upper bound given in the following theorem which states the main result of this paper. The lower bound is an immediate consequence of the fact that $\mathcal{ALC}$ with general TBoxes is EXPTIME-hard [Schild,1991].

Theorem 17. *Satisfiability and subsumption of $\mathcal{TDL}$-concepts w.r.t. TBoxes are* EXPTIME*-complete.*

5 Conclusion

There are several perspectives for future work of which we highlight three rather interesting ones: firstly, the presented decision procedure is only valid if a dense strict linear order is assumed as the underlying temporal structure. For example, the concept $\top$ is satisfiable w.r.t. the TBox

$$\mathcal{T} = \{\top \sqsubseteq \exists g_1, g_2, < \sqcap \exists g_1, fg_1, < \sqcap \exists fg_2, g_2, <\}$$

over the temporal structures $\mathbb{Q}$ and $\mathbb{R}$ (with the natural orderings) but not over $\mathbb{N}$. To see this, note that $\mathcal{T}$ induces a constraint graph as in Figure 1. Hence, it would be interesting to investigate if and how the presented algorithm can be modified for reasoning with the temporal structure $\mathbb{N}$.

Secondly, $\mathcal{TDL}$ does only allow for *qualitative* temporal reasoning. It would be interesting to extend the logic to mixed qualitative and quantitative reasoning by additionally admitting unary predicates $<_q$ and $=_q$ for each $q \in \mathbb{Q}$.

Thirdly, we plan to extend $\mathcal{TDL}$ to make it suitable for reasoning about entity relationship (ER) diagrams. As demonstrated in, e.g., [Calvanese *et al.*,1998; Artale and Franconi,1999], Description Logics are well-suited for this task. By using an appropriate extension of $\mathcal{TDL}$, one should be able to capture a new kind of temporal reasoning with ER diagrams, namely reasoning over ER diagrams with "temporal" integrity constraints. For example, a temporal integrity constraint could state that employees birthdays should be before their employment date. An appropriate extension of $\mathcal{TDL}$ for this task could be by (unqualified) number restrictions, inverse roles, and a generalized version of the concrete domain constructor $\exists u_1, u_2.P$. An extension of the presented automata-theoretic decision procedure to this more complex logic seems possible.

Acknowledgements My thanks go to Franz Baader, Ulrike Sattler, and Stephan Tobies for fruitful discussions. The author was supported by the DFG Project BA1122/3-1 "Combinations of Modal and Description Logics".

References

[Allen, 1983] J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11), 1983.

[Artale and Franconi, 1998] A. Artale and E. Franconi. A temporal description logic for reasoning about actions and plans. *Journal of Artificial Intelligence Research (JAIR)*, (9), 1998.

[Artale and Franconi, 1999] A. Artale and E. Franconi. Temporal ER modeling with description logics. In *Proc. of ER'99*, Paris, France, 1999. Springer–Verlag.

[Artale and Franconi, 2000] A. Artale and E. Franconi. Temporal description logics. In *Handbook of Time and Temporal Reasoning in Artificial Intelligence*. MIT Press, To appear.

[Baader and Hanschke, 1991] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In *Proc. of IJCAI-91*, pages 452–457, Sydney, Australia, 1991. Morgan Kaufmann Publ. Inc.

[Calvanese *et al.*, 1998] D. Calvanese, M, Lenzerini, and D. Nardi. Description logics for conceptual data modeling. In *Logics for Databases and Information Systems*, pages 229–263. Kluwer Academic Publisher, 1998.

[Lutz *et al.*, 1997] C. Lutz, V. Haarslev, and R. Möller. A concept language with role-forming predicate restrictions. Technical Report FBI-HH-M-276/97, University of Hamburg, Computer Science Department, Hamburg, 1997.

[Lutz, 2000] C. Lutz Interval-based Temporal Reasoning with General TBoxes. LTCS-Report LTCS-00-06, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 2000. See http://www-lti.informatik.rwth-aachen.de/Forschung/Reports.html.

[Lutz, 2001] C. Lutz. NExpTime-complete description logics with concrete domains. In *Proc. of IJCAR 2001*, LNCS, Siena, Italy, 2001. Springer-Verlag.

[Nebel, 1990] B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

[Sattler, 1998] U. Sattler. *Terminological knowledge representation systems in a process engineering application*. PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, 1998.

[Schild, 1991] K. D. Schild. A correspondence theory for terminological logics. In *Proc. of IJCAI-91*, pages 466–471, Sidney, Australia, 1991. Morgan Kaufmann Publ. Inc.

[Vardi and Wolper, 1986] M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logic of programs. *Journal of Computer and System Sciences*, 32:183–221, 1986.

[Vilain, 1982] M. Vilain. A system for reasoning about time. In *Proceedings of the Second AAAI*, pages 197–201, Pittsburgh, Pennsylvania, 1982.