Practical Reasoning for Very Expressive Description Logics

IAN HORROCKS, *Department of Computer Science, University of Manchester, Manchester, UK. E-mail: horrocks@cs.man.ac.uk*

ULRIKE SATTLER, *LuFG Theoretical Computer Science, RWTH Aachen, Germany. E-mail: sattler@informatik.rwth-aachen.de*

STEPHAN TOBIES, *LuFG Theoretical Computer Science, RWTH Aachen, Germany. E-mail: tobies@informatik.rwth-aachen.de*

Abstract

Description Logics (DLs) are a family of knowledge representation formalisms mainly characterised by constructors to build complex concepts and roles from atomic ones. Expressive role constructors are important in many applications, but can be computationally problematical.

We present an algorithm that decides satisfiability of the DL $\mathcal{ALC}$ extended with transitive and inverse roles and functional restrictions with respect to general concept inclusion axioms and role hierarchies; early experiments indicate that this algorithm is well-suited for implementation. Additionally, we show that $\mathcal{ALC}$ extended with just transitive and inverse roles is still in PSPACE. We investigate the limits of decidability for this family of DLs, showing that relaxing the constraints placed on the kinds of roles used in number restrictions leads to the undecidability of all inference problems. Finally, we describe a number of optimisation techniques that are crucial in obtaining implementations of the decision procedures, which, despite the hight worst-case complexity of the problem, exhibit good performance with real-life problems.

Keywords: description logic, modal logic, automated reasoning, tableaux algorithm

1 Motivation

Description Logics (DLs) are a well-known family of knowledge representation formalisms [17]. They are based on the notion of concepts (unary predicates, classes) and roles (binary relations), and are mainly characterised by constructors that allow complex concepts and roles to be built from atomic ones. Sound and complete algorithms for the interesting inference problems such as subsumption and satisfiability of concepts are known for a wide variety of DLs.

Transitive and inverse roles play an important role not only in the adequate representation of complex, aggregated objects [35], but also for reasoning with conceptual data models [9]. Moreover, defining concepts using general concept inclusion axioms seems natural and is crucial for representing conceptual data models.

The relevant inference problems for (an extension of) $\mathcal{ALC}$ augmented in the described manner are known to be decidable [15], and worst-case optimal inference algorithms have been described [16]. However, to the best of our knowledge, nobody has found efficient means to deal with their high degree of non-determinism, which so far prohibits their use in realistic applications. This is mainly due to the fact that

these algorithms can handle not only transitive roles but also the transitive closure of roles. It has been shown [43] that restricting the DL to transitive roles can lead to a lower complexity, and that transitive roles, even when combined with role hierarchies, allow for algorithms that behave quite well in realistic applications [31]. However, until now it has been unclear if this is still true when inverse roles are also present.

In this paper we present various aspects of our research in this direction. Firstly, we motivate our use of logics with transitive roles instead of transitive closure by contrasting algorithms for several pairs of logics that differ only in the kind of transitivity supported.

Secondly, we present an algorithm that decides satisfiability of $\mathcal{ALC}$ extended with transitive and inverse roles, role hierarchies, and functional restrictions. This algorithm can also be used for checking satisfiability and subsumption with respect to general concept inclusion axioms (and thus cyclic terminologies) because these axioms can be "internalised". The fact that our algorithm needs to deal only with transitive roles, instead of transitive closure, leads to a lower degree of non-determinism, and experiments indicate that the algorithm is well-suited for implementation.

Thirdly, we show that $\mathcal{ALC}$ extended with both transitive *and* inverse roles is still in PSPACE. The algorithm used to prove this result introduces an enhanced blocking technique that should also provide useful efficiency gains in implementations of more expressive DLs.

Fourthly, we investigate the limits of decidability for this family of DLs, showing that relaxing the constraints we will impose on the kind of roles allowed in number restrictions leads to the undecidability of all inference problems.

Finally, we describe a range of optimisation techniques that can be used to produce implementations of our algorithms that exhibit good typical case performance.

2 Preliminaries

In this section, we present the syntax and semantics of the various DLs that are investigated in subsequent sections. This includes the definition of inference problems (concept subsumption and satisfiability, and both of these problems with respect to terminologies) and how they are interrelated.

The logics we will discuss are all based on an extension of the well known DL $\mathcal{ALC}$ [45] to include transitively closed primitive roles [43]; we will call this logic $\mathcal{S}$ due to its relationship with the propositional (multi) modal logic $\mathbf{S4}_{(\mathbf{m})}$ [44].[1] This basic DL is then extended in a variety of ways—see Figure 1 for an overview.

Definition 2.1 *Let N_C be a set of* concept names *and* $\mathbf{R}$ *a set of* role names *with transitive role names $\mathbf{R}_+ \subseteq \mathbf{R}$. The set of $\mathcal{SI}$-*roles *is $\mathbf{R} \cup \{R^- \mid R \in \mathbf{R}\}$. To avoid considering roles such as R^{--}, we define a function* Inv *on roles such that $\mathsf{Inv}(R) = R^-$ if R is a role name, and $\mathsf{Inv}(R) = S$ if $R = S^-$. In the following, when speaking of roles, we refer to $\mathcal{SI}$-roles, as our approach is capable of dealing uniformly with both role names and inverse roles.*

Obviously, a role R is transitive iff $\mathsf{Inv}(R)$ is transitive. We therefore define Trans

[1] This logic has previously been called $\mathcal{ALC}_{R^+}$, but this becomes too cumbersome when adding letters to represent additional features.

to return true *iff R is a transitive role. More precisely,* $\mathsf{Trans}(R) = \text{true}$ *(and we say that R is transitive) iff* $R \in \mathbf{R}_+$ *or* $\mathsf{Inv}(R) \in \mathbf{R}_+$.

*The set of $\mathcal{SI}$-*concepts *is the smallest set such that*

1. *every concept name is a concept, and,*
2. *if C and D are concepts and R is an $\mathcal{SI}$-role, then $(C \sqcap D)$, $(C \sqcup D)$, $(\neg C)$, $(\forall R.C)$, and $(\exists R.C)$ are also concepts.*

A role inclusion axiom *is of the form $R \sqsubseteq S$, where R and S are two roles, each of which can be inverse. A* role hierarchy *is a finite set of role inclusion axioms, and $\mathcal{SHI}$ is obtained from $\mathcal{SI}$ by allowing, additionally, for a role hierarchy $\mathcal{R}$. The* sub-role relation $\sqsubseteq^*$ *is the transitive-reflexive closure of $\sqsubseteq$ over $\mathcal{R} \cup \{\mathsf{Inv}(R) \sqsubseteq \mathsf{Inv}(S) \mid R \sqsubseteq S \in \mathcal{R}\}$.*

$\mathcal{SHIQ}$ is obtained from $\mathcal{SHI}$ by allowing, additionally, for qualified number restrictions [26], i.e., for concepts of the form $\leqslant nR.C$ and $\geqslant nR.C$, where R is a simple *role, C is a concept, and $n \in \mathbb{N}$. A role is called* simple *iff it is neither transitive nor has transitive sub-roles. $\mathcal{SHIN}$ is the restriction of $\mathcal{SHIQ}$ allowing only unqualified number restrictions (i.e., concepts of the form $\leqslant nR$ and $\geqslant nR$), while $\mathcal{SHIF}$ represents a further restriction where, instead of arbitrary number restrictions, only* functional restrictions *of the form $\leqslant 1R$ and their negation $\geqslant 2R$ may occur.*

An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *consists of a set $\Delta^{\mathcal{I}}$, called the* domain *of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$ which maps every concept to a subset of $\Delta^{\mathcal{I}}$ and every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that, for all concepts C, D, roles R, S, and non-negative integers n, the properties in Figure 1 are satisfied, where $\sharp M$ denotes the cardinality of a set M. An interpretation satisfies a role hierarchy $\mathcal{R}$ iff $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ for each $R \sqsubseteq S \in \mathcal{R}$; we denote this fact by $\mathcal{I} \models \mathcal{R}$ and say that $\mathcal{I}$ is a model of $\mathcal{R}$.*

A concept C is called satisfiable *with respect to a role hierarchy $\mathcal{R}$ iff there is some interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{R}$ and $C^{\mathcal{I}} \neq \emptyset$. Such an interpretation is called a* model of C *w.r.t. $\mathcal{R}$. A concept D* subsumes *a concept C w.r.t. $\mathcal{R}$ (written $C \sqsubseteq_{\mathcal{R}} D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for each model $\mathcal{I}$ of $\mathcal{R}$. For an interpretation $\mathcal{I}$, an individual $x \in \Delta^{\mathcal{I}}$ is called an* instance *of a concept C iff $x \in C^{\mathcal{I}}$.*

All DLs considered here are closed under negation, hence subsumption and (un)satisfiability w.r.t. role hierarchies can be reduced to each other: $C \sqsubseteq_{\mathcal{R}} D$ iff $C \sqcap \neg D$ is unsatisfiable w.r.t. $\mathcal{R}$, and C is unsatisfiable w.r.t. $\mathcal{R}$ iff $C \sqsubseteq_{\mathcal{R}} A \sqcap \neg A$ for some concept name A.

In [37; 3; 44; 1], the *internalisation* of terminological axioms is introduced, a technique that reduces reasoning with respect to a (possibly cyclic) *terminology* to satisfiability of concepts. In [31], we saw how role hierarchies can be used for this reduction. In the presence of inverse roles, this reduction must be slightly modified.

Definition 2.2 *A* terminology *$\mathcal{T}$ is a finite set of general concept inclusion axioms, $\mathcal{T} = \{C_1 \sqsubseteq D_1, \ldots, C_n \sqsubseteq D_n\}$, where C_i, D_i are arbitrary $\mathcal{SHIF}$-concepts. An interpretation $\mathcal{I}$ is said to be a* model *of $\mathcal{T}$ iff $C_i^{\mathcal{I}} \subseteq D_i^{\mathcal{I}}$ holds for all $C_i \sqsubseteq D_i \in \mathcal{T}$. A concept C is* satisfiable *with respect to $\mathcal{T}$ iff there is a model $\mathcal{I}$ of $\mathcal{T}$ with $C^{\mathcal{I}} \neq \emptyset$. Finally, D* subsumes *C with respect to $\mathcal{T}$ iff, for each model $\mathcal{I}$ of $\mathcal{T}$, we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.*

The following lemma shows how general concept inclusion axioms can be *internalised* using a "universal" role U, a transitive super-role of all roles occurring in $\mathcal{T}$ and their respective inverses.

Construct Name	Syntax	Semantics	
atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	$\mathcal{S}$
atomic role	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	
transitive role	$R \in \mathbf{R}_+$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^+$	
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	
disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	
exists restriction	$\exists R.C$	$\{x \mid \exists y.\langle x,y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$	
value restriction	$\forall R.C$	$\{x \mid \forall y.\langle x,y\rangle \in R^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\}$	
role hierarchy	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$	$\mathcal{H}$
inverse role	R^-	$\{\langle x,y\rangle \mid \langle y,x\rangle \in R^{\mathcal{I}}\}$	$\mathcal{I}$
number	$\geqslant nR$	$\{x \mid \sharp\{y.\langle x,y\rangle \in R^{\mathcal{I}}\} \geqslant n\}$	$\mathcal{N}$
restrictions	$\leqslant nR$	$\{x \mid \sharp\{y.\langle x,y\rangle \in R^{\mathcal{I}}\} \leqslant n\}$	
qualifying number	$\geqslant nR.C$	$\{x \mid \sharp\{y.\langle x,y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geqslant n\}$	$\mathcal{Q}$
restrictions	$\leqslant nR.C$	$\{x \mid \sharp\{y.\langle x,y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \leqslant n\}$	

FIG. 1. Syntax and semantics of the $\mathcal{SI}$ family of DLs

Lemma 2.3 *Let $\mathcal{T}$ be a terminology, $\mathcal{R}$ a role hierarchy, and C, D $\mathcal{SHIF}$-concepts, and let*

$$C_{\mathcal{T}} := \bigsqcap_{C_i \sqsubseteq D_i \in \mathcal{T}} \neg C_i \sqcup D_i.$$

Let U be a transitive role that does not occur in $\mathcal{T}, C, D$, or $\mathcal{R}$. We set

$$\mathcal{R}_U := \mathcal{R} \cup \{R \sqsubseteq U, \mathsf{Inv}(R) \sqsubseteq U \mid R \text{ occurs in } \mathcal{T}, C, D, \text{ or } \mathcal{R}\}.$$

Then C is satisfiable w.r.t. $\mathcal{T}$ and $\mathcal{R}$ iff $C \sqcap C_{\mathcal{T}} \sqcap \forall U.C_{\mathcal{T}}$ is satisfiable w.r.t. $\mathcal{R}_U$. Moreover, D subsumes C w.r.t. $\mathcal{T}$ and $\mathcal{R}$ iff $C \sqcap \neg D \sqcap C_{\mathcal{T}} \sqcap \forall U.C_{\mathcal{T}}$ is unsatisfiable w.r.t. $\mathcal{R}_U$.

The proof of Lemma 2.3 is similar to the ones that can be found in [44; 3]. Most importantly, it must be shown that, (a) if a $\mathcal{SHIF}$-concept C is satisfiable with respect to a terminology $\mathcal{T}$ and a role hierarchy $\mathcal{R}$, then C, $\mathcal{T}$, and $\mathcal{R}$ have a *connected* model, and (b) if y is reachable from x via a role path (possibly involving inverse roles) in a model of $\mathcal{T}$ and $\mathcal{R}_U$, then $\langle x,y\rangle \in U^{\mathcal{I}}$. These are easy consequences of the semantics and the definition of U.

Theorem 2.4 *Satisfiability and subsumption of $\mathcal{SHIF}$-concepts (resp. $\mathcal{SHI}$-concepts) w.r.t. terminologies and role hierarchies are polynomially reducible to (un)satisfiability of $\mathcal{SHIF}$-concepts (resp. $\mathcal{SHI}$-concepts) w.r.t. role hierarchies.*

3 Blocking

The algorithms we are going to present for deciding satisfiability of $\mathcal{SI}$- and $\mathcal{SHIF}$-concepts use the tableaux method [25], in which the satisfiability of a concept D is

tested by trying to construct a model of D. The model is represented by a tree in which nodes correspond to individuals and edges correspond to roles. Each node x is labelled with a set of concepts $\mathcal{L}(x)$ that the individual x must satisfy, and edges are labelled with (sets of) role names.

An algorithm starts with a single node labelled $\{D\}$, and proceeds by repeatedly applying a set of *expansion rules* that recursively decompose the concepts in node labels, new edges and nodes being added as required in order to satisfy $\exists R.C$ or $(\geqslant 2\ F)$ concepts. The construction terminates either when none of the rules can be applied in a way that extends the tree, or when the discovery of obvious contradictions demonstrates that D has no model.

In order to prove that such an algorithm is a sound and complete decision procedure for concept satisfiability in a given logic, it is necessary to demonstrate that the models it constructs are correct with respect to the semantics, that it will always find a model if one exists, and that it always terminates. The first two points can usually be dealt with by proving that the expansion rules preserve satisfiability, and that in the case of non-deterministic expansion (e.g., of disjunctions) all possibilities are exhaustively searched. For logics such as $\mathcal{ALC}$, termination is mainly due to the fact that the expansion rules can only add new concepts that are strictly smaller than the decomposed concept, so the model must stabilise when all concepts have been fully decomposed. As we will see, this is no longer true in the presence of transitive roles.

3.1 Transitive Roles vs. Transitive Closure

We have argued that reasoning for logics with transitive roles is empirically more tractable than for logics that allow for transitive closure of roles [43; 31]. In this section we will give some justification for that claim. The starting point for our investigations are the logics $\mathcal{SH}$ [31] and $\mathcal{ALC}_+$ [3], which extend $\mathcal{ALC}$ by transitive roles and role hierarchies or transitive closure of roles respectively. Syntactically, $\mathcal{ALC}_+$ is similar to $\mathcal{S}$, where, in addition to transitive and non-transitive roles, the transitive closure R^+ of a role R may appear in existential and universal restrictions. Formally, R^+ is interpreted by

$$(R^+)^{\mathcal{I}} = \bigcup_{i \in \mathbb{N}} (R^{\mathcal{I}})^i, \quad \text{where } (R^{\mathcal{I}})^i = \begin{cases} R^{\mathcal{I}}, & \text{if } i = 1 \\ R^{\mathcal{I}} \circ (R^{\mathcal{I}})^{i-1}, & \text{otherwise} \end{cases}$$

For both $\mathcal{SH}$ and $\mathcal{ALC}_+$, concept satisfiability is an EXPTIME-complete problem. This result is easily derived from the EXPTIME-hardness proof for PDL in [18] and from the proof that PDL is in EXPTIME in [41]. Nevertheless, implementations of algorithms for $\mathcal{SH}$ exhibit good performance in realistic applications [34] whereas, at the moment, this seems to be more problematical for $\mathcal{ALC}_+$. We believe that the main reason for this discrepancy, at least in the case of tableau algorithm implementations, lies in the different complexity of the blocking conditions that are needed to guarantee the termination of the respective algorithms. In the following we are going to survey the blocking techniques needed to deal with $\mathcal{SH}$ and its subsequent extensions to $\mathcal{SHI}$ and $\mathcal{SHIF}$. To underpin our claim that reasoning with transitive roles empirically leads to more efficient implementations than for transitive closure, we will also present the blocking techniques used to deal with transitive closure. These are more complicated

and introduce a larger degree of non-determinism into the tableaux algorithms, leading to inferior performance of implementations.

3.2 Blocking for $\mathcal{S}$ and $\mathcal{SH}$

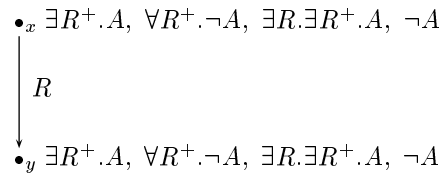
Termination of the expansion process of a tableaux algorithm is not guaranteed for logics that include transitive roles, as the expansion rules can introduce new concepts that are the same size as the decomposed concept. In particular, $\forall R.C$ concepts, where R is a transitive role, are dealt with by propagating the whole concept across R-labelled edges [43]. For example, given a node x labelled $\{C, \exists R.C, \forall R.(\exists R.C)\}$, where R is a transitive role, the combination of the $\exists R.C$ and $\forall R.(\exists R.C)$ concepts would cause a new node y to be added to the tree with a label identical to that of x. The expansion process could then be repeated indefinitely.

This problem can be dealt with by *blocking*: halting the expansion process when a *cycle* is detected [3; 8]. For logics without inverse roles, the general procedure is to check the label of each new node y, and if it is a *subset* [2] of the label of an ancestor node x, then no further expansion of y is performed: x is said to block y. The resulting tree corresponds to a cyclical model in which y is identified with x.

To deal with the transitive closure of roles, tableaux algorithms proceed by non-deterministically expanding a concept $\exists R^+.C$ to either $\exists R.C$ or $\exists R.\exists R^+.C$. Again, since the size of concepts along a path in the tree may not decrease, blocking techniques are necessary to guarantee termination. An adequate blocking condition for $\mathcal{ALC}_+$ is identical as for $\mathcal{SH}$, but one has to distinguish between *good* and *bad* cycles. Consider the following concept:

$$D = \exists R^+.A \sqcap \forall R^+.\neg A \sqcap \neg A$$

While D is obviously not satisfiable, a run of a tableaux algorithm might generate the following tableau in which node y is blocked by node x without generating any obvious contradictions.



The problem is that $\exists R^+.A$ has always been expanded to $\exists R.\exists R^+.A$, postponing the satisfaction of A a further step. To obtain a correct tableaux algorithm for $\mathcal{ALC}_+$, the blocking condition must include a check to ensure that each concept $\exists R^+.C$ appearing in such a cycle is expanded to $\exists R.C$ somewhere in the cycle. Such cycles are called *good* cycles, whereas cycles in which $\exists R^+.C$ has always been expanded to $\exists R.\exists R^+.C$ are called *bad* cycles. A valid model may only contain good cycles.

Summing up, using transitive closure instead of transitive roles has a twofold impact on the empirical tractability: (a) in blocking situations, good cycles have to be distinguished from bad ones, and (b) the non-deterministic expansion of concepts of the form $\exists R^+.C$ increases the size of the search space.

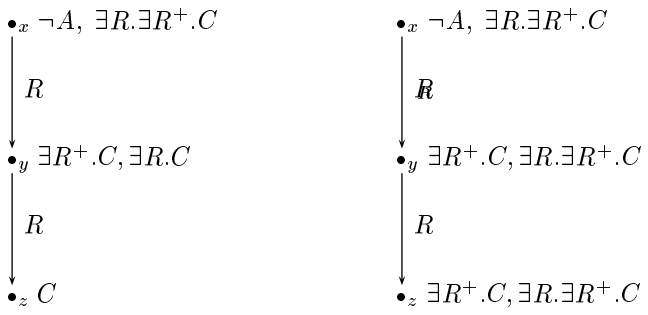


FIG. 2. Dynamic blocking fails in the presence of transitive closure.

3.3 Adding Inverse Roles

Blocking is more problematical when inverse roles are added to the logic, and a key feature of the algorithms presented in [35] was the introduction of a *dynamic blocking* strategy. Besides using label equality instead of subset, this strategy allowed blocks to be established, broken, and re-established. With inverse roles the blocking condition has to be considered more carefully because roles are now bi-directional, and additional concepts in x's label could invalidate the model with respect to y's predecessor. This problem can be overcome by allowing a node x to be blocked by one of its ancestors y if and only if they were labelled with the same sets of concepts.

Dealing with inverse roles is even more complicated in the presence of transitive closure. As an example consider the following concept:

$$D = \neg A \sqcap \exists R.\exists R^+.C$$
$$C = \forall R^-.(\forall R^-.A)$$

Fig. 2 shows two possible tableau expansions of the concept D. Continuing the expansion of the left hand tree will necessarily lead to a clash when concept $C \in \mathcal{L}(z)$ is expanded as this will lead to both A and $\neg A$ appearing in $\mathcal{L}(x)$. The right hand tree is also invalid as it contains a bad cycle: $\mathcal{L}(y) = \mathcal{L}(z)$ but $\exists R^+.D$ has always been expanded to $\exists R.\exists R^+.D$. Nevertheless, D is satisfiable, as it would be shown by continuing the expansion of the right hand path for one more step.

In [16], a solution to this problem for CPDL, a strict superset of $\mathcal{ALCI}_+$ ($\mathcal{ALC}_+$ plus inverse roles) is presented. The solution consists of an additional expansion rule called the *look behind analytical cut*. This rule employs exhaustive non-deterministic guessing to make the past of each node in the tree explicit in the labelling of that node: if y is an R-successor of a node x, then $\exists R^-.C$ or $\forall R^-.\neg C$ is added non-deterministically to the label of y for each concept C that may appear during the expansion process. Obviously, this leads to a further large increase in the size of the search space, with a correspondingly large adverse impact on empirical tractability. Experience with this kind of exhaustive guessing leads us to believe that an implementation of such an algorithm would be disastrously inefficient. The non-existence of implementations for $\mathcal{ALCI}_+$ or CPDL might be taken to support this view.

3.4 Pair-wise Blocking

Further extending the logic $\mathcal{SHI}$ to $\mathcal{SHIF}$ by adding functional restrictions (concepts of the form $(\leqslant 1\ R)$, meaning that an individual can be related to at most one other individual by the role R) introduces new problems associated with the fact that the logic no longer has the finite model property. This means that there are concepts that are satisfiable but for which there exists no finite model. An example of such a concept is

$$\neg C \sqcap \exists F^-.(C \sqcap (\leqslant 1\ F)) \sqcap \forall R^-.(\exists F^-.(C \sqcap (\leqslant 1\ F))),$$

where R is a transitive role and $F \sqsubseteq R$. Any model of this concept must contain an infinite sequence of individuals, each related to a single successors by an F^- role, and each satisfying $C \sqcap \exists F^-.C$, the $\exists F^-.C$ term being propagated along the sequence by the transitive super-role R. Attempting to terminate the sequence in a cycle causes the whole sequence to collapse into a single node due to the functional restrictions $(\leqslant 1\ F)$, and this results in a contradiction as both C and $\neg C$ will be in that node's label.

In order to deal with infinite models—namely to have an algorithm that terminates correctly even if the input concept has only infinite models—a more sophisticated *pair-wise* blocking strategy was introduced in [35], and soundness was proved by demonstrating that a blocked tree always has a corresponding infinite model.[2]

The only known algorithm that is able to deal with the combination of transitive closure, inverse roles, and functional restrictions on roles relies on an elaborate polynomial reduction to a CPDL terminology [13], and the capability of CPDL to internalise the resulting general terminological axioms. The large number and the nature of the axioms generated by this reduction make it very unlikely that an implementation with tolerable runtime behaviour will ever emerge.

4 Reasoning for $\mathcal{SI}$ Logics

In this section, we present two tableaux algorithms: the first decides satisfiability of $\mathcal{SHIF}$-concepts, and can be used for all $\mathcal{SHIF}$ reasoning problems (see Theorem 2.4); the second decides satisfiability (and hence subsumption) of $\mathcal{SI}$-concepts in PSPACE. In this paper we only sketch most of the proofs. For details on the $\mathcal{SHIF}$-algorithm, please refer to [35], for details on the $\mathcal{SI}$- and $\mathcal{SIN}$-algorithm, please refer to [27].

The correctness of the algorithms can be proved by showing that they create a *tableau* for a concept iff it is satisfiable.

For ease of construction, we assume all concepts to be in *negation normal form* (NNF), that is, negation occurs only in front of concept names. Any $\mathcal{SHIF}$-concept can easily be transformed to an equivalent one in NNF by pushing negations inwards [25].

Definition 4.1 *Let D be a $\mathcal{SHIF}$-concept in NNF, $\mathcal{R}$ a role hierarchy, and $\mathbf{R}_D$ the set of roles occurring in D together with their inverses, and $sub(D)$ the subconcepts of D. Then $T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ is a* tableau *for D w.r.t. $\mathcal{R}$ iff $\mathbf{S}$ is a set of individuals, $\mathcal{L} : \mathbf{S} \to 2^{sub(D)}$ maps each individual to a set of concepts, $\mathcal{E} : \mathbf{R}_D \to 2^{\mathbf{S}\times\mathbf{S}}$ maps each role to a set of pairs of individuals, and there is some individual $s \in \mathbf{S}$ such that*

[2] This is not to say that it may not also have a finite model.

$D \in \mathcal{L}(s)$. Furthermore, for all $s,t \in \mathbf{S}$, $C,E \in sub(D)$, and $R,S \in \mathbf{R}_D$, it holds that:

1. *if $C \in \mathcal{L}(s)$, then $\neg C \notin \mathcal{L}(s)$,*
2. *if $C \sqcap E \in \mathcal{L}(s)$, then $C \in \mathcal{L}(s)$ and $E \in \mathcal{L}(s)$,*
3. *if $C \sqcup E \in \mathcal{L}(s)$, then $C \in \mathcal{L}(s)$ or $E \in \mathcal{L}(s)$,*
4. *if $\forall R.C \in \mathcal{L}(s)$ and $\langle s,t\rangle \in \mathcal{E}(R)$, then $C \in \mathcal{L}(t)$,*
5. *if $\exists R.C \in \mathcal{L}(s)$, then there is some $t \in \mathbf{S}$ such that $\langle s,t\rangle \in \mathcal{E}(R)$ and $C \in \mathcal{L}(t)$,*
6. *if $\forall S.C \in \mathcal{L}(s)$ and $\langle s,t\rangle \in \mathcal{E}(R)$ for some $R \sqsubseteq^* S$ with $\mathsf{Trans}(R)$, then $\forall R.C \in \mathcal{L}(t)$,*
7. *$\langle s,t\rangle \in \mathcal{E}(R)$ iff $\langle t,s\rangle \in \mathcal{E}(\mathsf{Inv}(R))$.*
8. *if $\langle x,y\rangle \in \mathcal{E}(R)$ and $R \sqsubseteq^* S$, then $\langle x,y\rangle \in \mathcal{E}(S)$,*
9. *if $\leqslant 1 R \in \mathcal{L}(s)$, then $\sharp\{t \mid \langle s,t\rangle \in \mathcal{E}(R)\} \leq 1$, and*
10. *if $\geqslant 2 R \in \mathcal{L}(s)$, then $\sharp\{t \mid \langle s,t\rangle \in \mathcal{E}(R)\} \geq 2$.*

Tableaux for $\mathcal{SI}$-concepts are defined analogously and must satisfy Properties 1-7, where, due to the absence of a role hierarchy, $\sqsubseteq^*$ is the identity.

Due to the close relationship between models and tableaux, the following lemma can be easily proved by induction on the structure of concepts. As a consequence, an algorithm that constructs (if possible) a tableau for an input concept is a decision procedure for satisfiability of concepts.

Lemma 4.2 *A $\mathcal{SHIF}$-concept (resp. $\mathcal{SI}$-concept) D is satisfiable w.r.t. a role hierarchy $\mathcal{R}$ iff D has a tableau w.r.t. $\mathcal{R}$.*

4.1 Reasoning in $\mathcal{SHIF}$

In the following, we give an algorithm that, given a $\mathcal{SHIF}$-concept D, decides the existence of a tableaux for D. We implicitly assume an arbitrary but fixed role hierarchy $\mathcal{R}$.

Definition 4.3 *A* completion tree *for a $\mathcal{SHIF}$-concept D is a tree where each node x of the tree is labelled with a set $\mathcal{L}(x) \subseteq sub(D)$ and each edge $\langle x,y\rangle$ is labelled with a set $\mathcal{L}(\langle x,y\rangle)$ of (possibly inverse) roles occurring in $sub(D)$.*

Given a completion tree, a node y is called an R-successor *of a node x iff y is a successor of x and $S \in \mathcal{L}(\langle x,y\rangle)$ for some S with $S \sqsubseteq^* R$. A node y is called an* R-neighbour *of x iff y is an R-successor of x, or if x is an $\mathsf{Inv}(R)$-successor of y. Predecessors and ancestors are defined as usual.*

A node is blocked *iff it is directly or indirectly blocked. A node x is* directly blocked *iff none of its ancestors are blocked, and it has ancestors x', y and y' such that*

1. *x is a successor of x' and y is a successor of y'* and
2. *$\mathcal{L}(x) = \mathcal{L}(y)$ and $\mathcal{L}(x') = \mathcal{L}(y')$* and
3. *$\mathcal{L}(\langle x',x\rangle) = \mathcal{L}(\langle y',y\rangle)$.*

In this case we will say that y blocks x.

A node y is indirectly blocked *iff one of its ancestors is blocked, or—in order to avoid wasted expansion after an application of the $\leqslant$-rule—it is a successor of a node x and $\mathcal{L}(\langle x,y\rangle) = \emptyset$.*

$\sqcap$-rule: if 1. $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x is not indirectly blocked, and
2. $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$
then $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C_1, C_2\}$

$\sqcup$-rule: if 1. $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not indirectly blocked, and
2. $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
then, for some $C \in \{C_1, C_2\}$, $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C\}$

$\exists$-rule: if 1. $\exists S.C \in \mathcal{L}(x)$, x is not blocked, and
2. x has no S-neighbour y with $C \in \mathcal{L}(y)$
then create a new node y with
$\mathcal{L}(\langle x, y\rangle) = \{S\}$ and $\mathcal{L}(y) = \{C\}$

$\forall$-rule: if 1. $\forall S.C \in \mathcal{L}(x)$, x is not indirectly blocked, and
2. there is an S-neighbour y of x with $C \notin \mathcal{L}(y)$
then $\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{C\}$

$\forall_+$-rule: if 1. $\forall S.C \in \mathcal{L}(x)$, x is not indirectly blocked,
2. there is some R with $\mathsf{Trans}(R)$ and $R \sqsubseteq^* S$, and
3. x has an R-neighbour y with $\forall R.C \notin \mathcal{L}(y)$
then $\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{\forall R.C\}$

$\geqslant$-rule: if 1. $(\geqslant 2\ R) \in \mathcal{L}(x)$, x is not blocked, and
2. there is no R-neighbour y of x with $A \in \mathcal{L}(y)$
then create two new nodes y_1, y_2 with
$\mathcal{L}(\langle x, y_1\rangle) = \mathcal{L}(\langle x, y_2\rangle) = \{R\}$,
$\mathcal{L}(y_1) = \{A\}$ and $\mathcal{L}(y_2) = \{\neg A\}$

$\leqslant$-rule: if 1. $(\leqslant 1\ R) \in \mathcal{L}(x)$, x is not indirectly blocked,
2. x has two R-neighbours y and z s.t. y is not an ancestor of z,
then 1. $\mathcal{L}(z) \longrightarrow \mathcal{L}(z) \cup \mathcal{L}(y)$ and
2. if z is an ancestor of y
then $\mathcal{L}(\langle z, x\rangle) \longrightarrow \mathcal{L}(\langle z, x\rangle) \cup \mathsf{Inv}(\mathcal{L}(\langle x, y\rangle))$
else $\mathcal{L}(\langle x, z\rangle) \longrightarrow \mathcal{L}(\langle x, z\rangle) \cup \mathcal{L}(\langle x, y\rangle)$
3. $\mathcal{L}(\langle x, y\rangle) \longrightarrow \emptyset$

FIG. 3. The tableaux expansion rules for $\mathcal{SHIF}$

For a node x, $\mathcal{L}(x)$ is said to contain a clash *iff $\{A, \neg A\} \subseteq \mathcal{L}(x)$ or $\{\geqslant 2R, \leqslant 1S\} \subseteq \mathcal{L}(x)$ for roles $R \sqsubseteq^* S$. A completion tree is called* clash-free *iff none of its nodes contains a clash; it is called* complete *iff none of the expansion rules in Figure 3 is applicable.*

For a $\mathcal{SHIF}$-concept D in NNF, the algorithm starts with a completion tree consisting of a single node x with $\mathcal{L}(x) = \{D\}$. It applies the expansion rules, stopping when a clash occurs, and answers "D is satisfiable" iff the completion rules can be applied in such a way that they yield a complete and clash-free completion tree.

The soundness and completeness of the tableaux algorithm is an immediate consequence of Lemmas 4.2 and 4.4.

Lemma 4.4 *Let D be an $\mathcal{SHIF}$-concept.*

1. *The tableaux algorithm terminates when started with D.*
2. *If the expansion rules can be applied to D such that they yield a complete and*

clash-free completion tree, then D has a tableau.

3. *If D has a tableau, then the expansion rules can be applied to D such that they yield a complete and clash-free completion tree.*

Before we sketch the ideas of the proof, we will discuss the different expansion rules and their correspondence to the language constructors.

The $\sqcap$-, $\sqcup$-, $\exists$- and $\forall$-rules are the standard $\mathcal{ALC}$ tableaux rules [45]. The $\forall_+$-rule is used to handle transitive roles, where the $\overset{*}{\sqsubseteq}$-clause deals with the role hierarchy. See [35] for details.

The functional restriction rules merit closer consideration. In order to guarantee the satisfaction of a $\geqslant 2R$-constraint, the $\geqslant$-rule creates two successors and uses a fresh atomic concept A to prohibit identification of these successors by the $\leqslant$-rule. If a node x has two or more R-neighbours and contains a functional restriction $\leqslant 1R$, then the $\leqslant$-rule merges two of the neighbours *and* also merges the edges connecting them with x. Labelling edges with sets of roles allows a single node to be both an R and S-successor of x even if R and S are not comparable by $\overset{*}{\sqsubseteq}$. Finally, contradicting functional restrictions are taken care of by the definition of a clash.

We now sketch the main ideas behind the proof of Lemma 4.4:

1. Termination: Let $m = |sub(D)|$ and $n = |\mathbf{R}_D|$. Termination is a consequence of the following properties of the expansion rules:

(a) The expansion rules never remove nodes from the tree or concepts from node labels. Edge labels can only be changed by the $\leqslant$-rule which either expands them or sets them to $\emptyset$; in the latter case the node below the $\emptyset$-labelled edge is blocked. (b) Successors are only generated for concepts of the form $\exists R.C$ and $\geqslant 2R$. For a node x, each of these concepts triggers the generation of at most two successors. If for one of these successors y the $\leqslant$-rule subsequently causes $\mathcal{L}(\langle x, y\rangle)$ to be changed to $\emptyset$, then x will have some R-neighbour z with $\mathcal{L}(z) \supseteq \mathcal{L}(y)$. This, together with the definition of a clash, implies that the concept that led to the generation of y will not trigger another rule application. Obviously, the out-degree of the tree is bounded by $2m$. (c) Nodes are labelled with non-empty subsets of $sub(D)$ and edges with subsets of $\mathbf{R}_D$, so there are at most 2^{2mn} different possible labellings for a pair of nodes and an edge. Therefore, on a path of length at least 2^{2mn} there must be 2 nodes x, y such that x is directly blocked by y. Since a path on which nodes are blocked cannot become longer, paths are of length at most 2^{2mn}.

2. Soundness: A complete and clash-free tree $\mathbf{T}$ for D induces the existence of a tableaux $T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ for D as follows. Individuals in $\mathbf{S}$ correspond to *paths* in $\mathbf{T}$ from the root node to some node that is not blocked. Instead of going to a directly blocked node, these paths jump back to the blocking node, which yields paths of arbitrary length. Thus, if blocking occurs, this construction yields an infinite tableau. This rather complicated tableau construction is necessary due to the presence of functional restrictions; its validity is ensured by the blocking condition, which considers both the blocked node and its predecessor.

3. Completeness: A tableau $T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ for D can be used to "steer" the application of the non-deterministic $\sqcup$- and $\leqslant$-rules in a way that yields a complete and clash-free tree.

The following theorem is an immediate consequence of Lemma 4.4, Lemma 4.2, and Lemma 2.3.

Theorem 4.5 *The tableaux algorithm is a decision procedure for the satisfiability and subsumption of $\mathcal{SHIF}$-concepts with respect to terminologies and role hierachies.*

4.2 A PSPACE*-algorithm for $\mathcal{SI}$*

To obtain a PSPACE-algorithm for $\mathcal{SI}$, the $\mathcal{SHIF}$ algorithm is modified as follows: (a) As $\mathcal{SI}$ does not allow for functional restrictions, the $\geqslant$- and the $\leqslant$-rule can be omitted; blocking no longer involves two pairs of nodes with identical labels but only two nodes with "similar" labels. (b) Due to the absence of role hierarchies, edge labels can be restricted to roles (instead of sets of roles). (c) To obtain a PSPACE algorithm, we employ a refined blocking strategy which necessitates a second label $\mathcal{B}$ for each node. This blocking technique, while discovered independently, is based on ideas similar to those used in [46] to show that satisfiability for $\mathbf{K4_t}$ can be decided in PSPACE.[3] In the following, we will describe and motivate this blocking technique; detailed proofs as well as a similar result for $\mathcal{SIN}$ can be found in [27].

Please note that naively using a cut rule does not yield a PSpace algorithm: a cut rule similar to the *look behind analytical cut* presented in [16] (non-deterministically) guesses which constraints will be propagated "up" the completion tree by universal restrictions on inverted roles. For $\mathcal{SI}$, this technique may lead to paths of exponential length due to equality blocking. A way to avoid these long paths would be to stop the investigation of a path at some polynomial bound. However, to prove the correctness of this approach, it would be necessary to establish a "short-path-model" property similar to Lemma 4.8. Furthermore, we believe that our algorithm is better suited for an implementation since it makes less use of "don't-know" non-determinism. This also distinguishes our approach from the algorithm presented in [46], which is not intended to form the basis for an efficient implementation.

Definition 4.6 *A* completion tree *for a $\mathcal{SI}$ concept D is a tree where each node x of the tree is labelled with two sets $\mathcal{B}(x) \subseteq \mathcal{L}(x) \subseteq sub(D)$ and each edge $\langle x, y\rangle$ is labelled with a (possibly inverse) role $\mathcal{L}(\langle x, y\rangle)$ occurring in $sub(D)$.*

R-neighbours, -successors, and -predecessors are defined as in Definition 4.3. Due to the absence of role hierarchies, $\sqsubseteq^*$ is the identity on $\mathbf{R}_D$.

A node x is blocked *iff, for an ancestor y, y is blocked or*

$$\mathcal{B}(x) \subseteq \mathcal{L}(y) \quad \text{and} \quad \mathcal{L}(x)/\mathsf{Inv}(S) = \mathcal{L}(y)/\mathsf{Inv}(S),$$

where x' is the predecessor of x, $\mathcal{L}(\langle x', x\rangle) = S$, and $\mathcal{L}(x)/\mathsf{Inv}(S) = \{\forall\, \mathsf{Inv}(S).C \in \mathcal{L}(x)\}$.

For a node x, $\mathcal{L}(x)$ is said to contain a clash *iff $\{A, \neg A\} \subseteq \mathcal{L}(x)$. A completion tree to which none of the expansion rules given in Figure 4 is applicable is called* complete.

For an $\mathcal{SI}$-concept D, the algorithm starts with a completion tree consisting of a single node x with $\mathcal{B}(x) = \mathcal{L}(x) = \{D\}$. It applies the expansion rules in Figure 4, stopping when a clash occurs, and answers "D is satisfiable" iff the completion rules can be applied in such a way that they yield a complete and clash-free completion tree.

As for $\mathcal{SHIF}$, correctness of the algorithm is proved by first showing that a $\mathcal{SI}$-concept is satisfiable iff it has a tableau, and next proving the $\mathcal{SI}$-analogue of Lemma 4.4.

[3] The modal logic $\mathbf{K4_t}$ is a syntactic variant of $\mathcal{SI}$ with only a single transitive role name.

$\sqcap$-rule: if 1. $C_1 \sqcap C_2 \in \mathcal{L}(x)$ and
2. $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$
then $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C_1, C_2\}$

$\sqcup$-rule: if 1. $C_1 \sqcup C_2 \in \mathcal{L}(x)$ and
2. $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
then $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C\}$ for some $C \in \{C_1, C_2\}$

$\forall$-rule: if 1. $\forall S.C \in \mathcal{L}(x)$ and
2. there is an S-successor y of x with $C \notin \mathcal{B}(y)$
then $\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{C\}$ and
$\mathcal{B}(y) \longrightarrow \mathcal{B}(y) \cup \{C\}$ or
2'. there is an S-predecessor y of x with $C \notin \mathcal{L}(y)$
then $\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{C\}$.

$\forall_+$-rule: if 1. $\forall S.C \in \mathcal{L}(x)$ and $\mathsf{Trans}(S)$ and
2. there is an S-successor y of x with $\forall S.C \notin \mathcal{B}(y)$
then $\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{\forall S.C\}$ and
$\mathcal{B}(y) \longrightarrow \mathcal{B}(y) \cup \{\forall S.C\}$ or
2'. there is an S-predecessor y of x with $\forall S.C \notin \mathcal{L}(y)$
then $\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{\forall S.C\}$.

$\exists$-rule: if 1. $\exists S.C \in \mathcal{L}(x)$, x is not blocked and no other rule is applicable to any of its ancestors, and
2. x has no S-neighbour y with $C \in \mathcal{L}(y)$
then create a new node y with
$\mathcal{L}(\langle x, y\rangle) = S$ and $\mathcal{L}(y) = \mathcal{B}(y) = \{C\}$

FIG. 4. Tableaux expansion rules for $\mathcal{SI}$

Theorem 4.7 *The tableaux algorithm is a decision procedure for satisfiability and subsumption of $\mathcal{SI}$-concepts.*

The dynamic blocking technique for $\mathcal{SI}$ and $\mathcal{SHI}$ described in Section 3, which is based on label equality, may lead to completion trees with exponentially long paths because there are exponentially many possibilities to label sets on such a path. Due to the non-deterministic $\sqcup$-rule, these exponentially many sets may actually occur.

This non-determinism is not problematical for $\mathcal{S}$ because disjunctions need not be completely decomposed to yield a subset-blocking situation. For an optimal $\mathcal{SI}$ algorithm, the additional label $\mathcal{B}$ was introduced to enable a sort of subset-blocking which is independent of the $\sqcup$-non-determinism. Intuitively, $\mathcal{B}(x)$ is the restriction of $\mathcal{L}(x)$ to those non-decomposed concepts that x must satisfy, whereas $\mathcal{L}(x)$ contains boolean decompositions of these concepts as well as those that are imposed by value restrictions in descendants. If x is blocked by y, then all concepts in $\mathcal{B}(x)$ are eventually decomposed in $\mathcal{L}(y)$ (if no clash occurs). However, in order to substitute x by y, x's constraints on predecessors must be at least as strong as y's; this is taken care of by the second blocking condition.

Let us consider a path $x_1, \ldots, x_n$ where all edges are labelled R with $\mathsf{Trans}(R)$, the only kind of paths along which the length of the longest concept in the labels might not decrease. If no rules can be applied, we have $\mathcal{L}(x_{i+1})/\mathsf{Inv}(R) \subseteq \mathcal{L}(x_i)/\mathsf{Inv}(R)$ and $\mathcal{B}(x_i) \subseteq \mathcal{B}(x_{i+1}) \cup \{C_i\}$ (where $\exists R.C_i$ triggered the generation of x_{i+1}). This limits the number of labels and guarantees blocking after a polynomial number of steps.

Lemma 4.8 *The paths of a completion tree for a concept D have a length of at most m^4 where $m = |sub(D)|$.*

Finally, a slight modification of the expansion rules given in Figure 4 yields a PSPACE algorithm. This modification is necessary because the original algorithm must keep the whole completion tree in its memory—which needs exponential space even though the length of its paths is polynomially bounded. The original algorithm may not forget about branches because restrictions which are pushed *upwards* in the tree might make it necessary to revisit paths which have been considered before. We solve this problem as follows:

Whenever the $\forall$- or the $\forall_+$-rule is applied to a node x and its *predecessor* y (Case 2' of these rules), we delete all successors of y from the completion tree. While this makes it necessary to restart the generation of successors for y, it makes it possible to implement the algorithm in a depth-first manner which facilitates the re-use of space.

This modification does not affect the proof of soundness and completeness for the algorithm, but we have to re-prove termination [27] as it relied on the fact that we never removed any nodes from the completion tree. Summing up we get:

Theorem 4.9 *The modified algorithm is a* PSPACE *decision procedure for satisfiability and subsumption of $\mathcal{SI}$-concepts.*

5 The Undecidability of Unrestricted $\mathcal{SHN}$

In [28] we describe an algorithm for $\mathcal{SHIQ}$ based on the $\mathcal{SHIF}$-algorithm already presented. Like earlier DLs that combine a hierarchy of (transitive and non-transitive) roles with some form of number restrictions [35; 27] and $\mathcal{SHIF}$, the DL $\mathcal{SHIQ}$ allows only *simple* roles in number restrictions. The justification for this limitation has been partly on the grounds of a doubtful semantics (of transitive functional roles) and partly to simplify decision procedures. In this section we will show that, even for the simpler $\mathcal{SHN}$ logic, allowing arbitrary roles in number restrictions leads to undecidability, while decidability for the corresponding variant of $\mathcal{SHIF}$ is still an open problem. For convenience, we will refer to $\mathcal{SHN}$ with arbitrary roles in number restrictions as $\mathcal{SHN}^+$.

The undecidability proof uses a reduction of the domino problem [7] adapted from [4]. This problem asks if, for a set of domino types, there exists a *tiling* of an $\mathbb{N}^2$ grid such that each point of the grid is covered with one of the domino types, and adjacent dominoes are "compatible" with respect to some predefined criteria.

Definition 5.1 *A domino system $\mathcal{D} = (D, H, V)$ consists of a non-empty set of domino types $D = \{D_1, \ldots, D_n\}$, and of sets of horizontally and vertically matching pairs $H \subseteq D \times D$ and $V \subseteq D \times D$. The problem is to determine if, for a given $\mathcal{D}$, there exists a* tiling *of an $\mathbb{N} \times \mathbb{N}$ grid such that each point of the grid is covered with a domino type in D and all horizontally and vertically adjacent pairs of domino types are in H and V respectively, i.e., a mapping $t : \mathbb{N} \times \mathbb{N} \to D$ such that for all $m, n \in \mathbb{N}$, $\langle t(m,n), t(m+1,n)\rangle \in H$ and $\langle t(m,n), t(m,n+1)\rangle \in V$.*

This problem can be reduced to the satisfiability of $\mathcal{SHN}^+$-concepts, and the undecidability of the domino problem implies undecidability of satisfiability of $\mathcal{SHN}^+$-concepts.

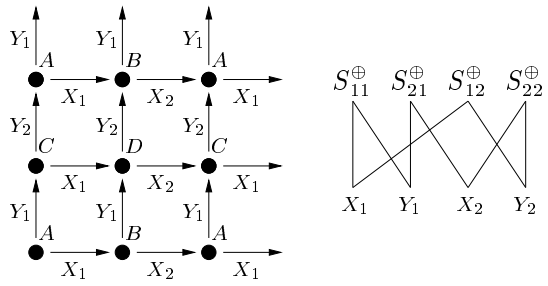


FIG. 5. Visualisation of the grid and role hierarchy.

Ensuring that a given point satisfies the compatibility conditions is simple for most logics (using value restrictions and boolean connectives), and applying such conditions throughout the grid is also simple in a logic such as $\mathcal{SHN}^+$ which can deal with arbitrary axioms. The crucial difficulty is representing the $\mathbb{N} \times \mathbb{N}$ grid using "horizontal" and "vertical" roles X and Y, and in particular forcing the coincidence of $X \circ Y$ and $Y \circ X$ successors. This can be accomplished in $\mathcal{SHN}^+$ using an alternating pattern of two horizontal roles X_1 and X_2, and two vertical roles Y_1 and Y_2, with disjoint primitive concepts A, B, C, and D being used to identify points in the grid with different combinations of successors. The coincidence of $X \circ Y$ and $Y \circ X$ successors can then be enforced using number restrictions on transitive super-roles of each of the four possible combinations of X and Y roles. A visualisation of the resulting grid and a suitable role hierarchy is shown in Figure 5, where $S^{\oplus}_{ij}$ are transitive roles.

The alternation of X and Y roles in the grid means that one of the transitive super-roles $S^{\oplus}_{ij}$ connects each point (x, y) to the points $(x+1, y)$, $(x, y+1)$ and $(x+1, y+1)$, and to no other points. A number restriction of the form $\leqslant 3 S^{\oplus}_{ij}$ can thus be used to enforce the necessary coincidence of $X \circ Y$ and $Y \circ X$ successors. A complete specification of the grid is given by the following axioms:

$$A \sqsubseteq \neg B \sqcap \neg C \sqcap \neg D \sqcap \exists X_1.B \sqcap \exists Y_1.C \sqcap \leqslant 3 S^{\oplus}_{11},$$
$$B \sqsubseteq \neg A \sqcap \neg C \sqcap \neg D \sqcap \exists X_2.A \sqcap \exists Y_1.D \sqcap \leqslant 3 S^{\oplus}_{21},$$
$$C \sqsubseteq \neg A \sqcap \neg B \sqcap \neg D \sqcap \exists X_1.D \sqcap \exists Y_2.A \sqcap \leqslant 3 S^{\oplus}_{12},$$
$$D \sqsubseteq \neg A \sqcap \neg B \sqcap \neg C \sqcap \exists X_2.C \sqcap \exists Y_2.B \sqcap \leqslant 3 S^{\oplus}_{22}.$$

It only remains to add axioms which encode the local compatibility conditions (as described in [4]) and to assert that A is subsumed by the disjunction of all domino types. The $\mathcal{SHN}^+$-concept A is now satisfiable w.r.t. the various axioms (which can be internalised as described in Lemma 2.3) iff there is a compatible tiling of the grid.

6 Implementation and Optimisation

The development of the $\mathcal{SI}$ family of DLs has been motivated by the desire to implement systems with good typical case performance. As discussed in Section 3, this is achieved in part through the design of the logics and algorithms themselves, in particular by using transitive roles and by reasoning with number restrictions directly, rather

than via encodings. Another important feature of these algorithms is that their relative simplicity facilitates the application of a range of optimisation techniques. Several systems based on $\mathcal{S}$ logics have now been implemented (e.g., FaCT [30], DLP [40] and RACE [23]), and have demonstrated that suitable optimisation techniques can lead to a dramatic improvement in the performance of the algorithms when used in realistic applications. A system based on the $\mathcal{SHIF}$ logic has also been implemented (iFaCT [32]) and has been shown to be similarly amenable to optimisation.

DL systems are typically used to classify a KB, and the optimisation techniques used in such systems can be divided into four categories based on the stage of the classification process at which they are applied.

1. Preprocessing optimisations that try to modify the KB so that classification and subsumption testing are easier.
2. Partial ordering optimisations that try to minimise the number of subsumption tests required in order to classify the KB.
3. Subsumption optimisations that try to avoid performing a potentially expensive satisfiability test, usually by substituting a cheaper test.
4. Satisfiability optimisations that try to improve the typical case performance of the underlying satisfiability testing algorithm.

Many optimisations in the first three categories are relatively independent of the underlying subsumption (satisfiability) testing algorithm and could be applied to any DL system. As we are mostly concerned with algorithms for the $\mathcal{SI}$ family of DLs we will concentrate on the fourth kind of optimisation, those that try to improve the performance of the algorithm itself. Most of these are aimed at reducing the size of the search space explored by the algorithm as a result of applying non-deterministic tableaux expansion rules.

6.1 Semantic Branching Search

Implementations of the algorithms described in the previous sections typically use a search technique called *syntactic branching*. When expanding the label of a node x, syntactic branching works by choosing an unexpanded disjunction $(C_1 \sqcup \ldots \sqcup C_n)$ in $\mathcal{L}(x)$ and searching the different models obtained by adding each of the disjuncts $C_1, \ldots, C_n$ to $\mathcal{L}(x)$ [22]. As the alternative branches of the search tree are not disjoint, there is nothing to prevent the recurrence of an unsatisfiable disjunct in different branches. The resulting wasted expansion could be costly if discovering the unsatisfiability requires the solution of a complex sub-problem. For example, tableaux expansion of a node x, where $\{(A \sqcup B), (A \sqcup C)\} \subseteq \mathcal{L}(x)$ and A is an unsatisfiable concept, could lead to the search pattern shown in Figure 6, in which the unsatisfiability of $\mathcal{L}(x) \cup \{A\}$ must be demonstrated twice.

This problem can be dealt with by using a *semantic branching* technique adapted from the Davis-Putnam-Logemann-Loveland procedure (DPL) commonly used to solve propositional satisfiability (SAT) problems [12; 21]. Instead of choosing an unexpanded disjunction in $\mathcal{L}(x)$, a single disjunct D is chosen from one of the unexpanded disjunctions in $\mathcal{L}(x)$. The two possible sub-trees obtained by adding either D or $\neg D$ to $\mathcal{L}(x)$ are then searched. Because the two sub-trees are strictly disjoint, there is no possibility of wasted search as in syntactic branching. Note that the order

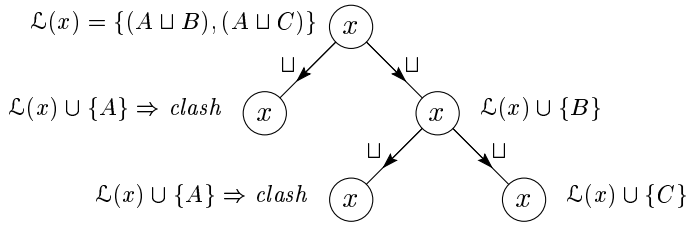


FIG. 6. Syntactic branching search

in which the two branches are explored is irrelevant from a theoretical viewpoint, but may offer further optimisation possibilities (see Section 6.4).

Semantic branching search has the additional advantage that a great deal is known about the implementation and optimisation of the DPL algorithm. In particular, both *local simplification* (see Section 6.2) and *heuristic guided search* (see Section 6.4) can be used to try to minimise the size of the search tree (although it should be noted that both these techniques can also be adapted for use with syntactic branching search).

There are also some disadvantages to semantic branching search. Firstly, it is possible that performance could be degraded by adding the negated disjunct in the second branch of the search tree, for example if the disjunct is a very large or complex concept. However this does not seem to be a serious problem in practice, with semantic branching rarely exhibiting significantly worse performance than syntactic branching. Secondly, its effectiveness is problem dependent. It is most effective with randomly generated problems, particularly those that are over-constrained (likely to be unsatisfiable) [34]. It is also effective with some of the hand crafted problems from the Tableaux'98 benchmark suite [24; 6]. However it is of little benefit when classifying realistic KBs [33].

6.2 Local Simplification

Local simplification is another technique used to reduce the size of the search space resulting from the application of non-deterministic expansion rules. Before any non-deterministic expansion of a node label $\mathcal{L}(x)$ is performed, disjunctions in $\mathcal{L}(x)$ are examined, and if possible simplified. The simplification most commonly used is to deterministically expand disjunctions in $\mathcal{L}(x)$ that present only one expansion possibility and to detect a clash when a disjunction in $\mathcal{L}(x)$ has no expansion possibilities. This simplification has been called *boolean constraint propagation* (BCP) [20]. In effect, the inference rule

$$\frac{\neg C_1, \ldots, \neg C_n, C_1 \sqcup \ldots \sqcup C_n \sqcup D}{D}$$

is being used to simplify the conjunctive concept represented by $\mathcal{L}(x)$. For example, given a node x such that

$$\{(C \sqcup (D_1 \sqcap D_2)), (\neg D_1 \sqcup \neg D_2 \sqcup C), \neg C\} \subseteq \mathcal{L}(x),$$

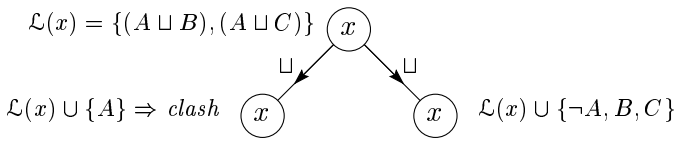


FIG. 7. Semantic branching search

BCP deterministically expands the disjunction $(C \sqcup (D_1 \sqcap D_2))$, adding $(D_1 \sqcap D_2)$ to $\mathcal{L}(x)$, because $\neg C \in \mathcal{L}(x)$. The deterministic expansion of $(D_1 \sqcap D_2)$ adds both D_1 and D_2 to $\mathcal{L}(x)$, allowing BCP to identify $(\neg D_1 \sqcup \neg D_2 \sqcup C)$ as a clash (without any branching having occurred), because $\{D_1, D_2, \neg C\} \subseteq \mathcal{L}(x)$.

BCP simplification is usually described as an integral part of SAT based algorithms [22], but it can also be used with syntactic branching. However, it is more effective with semantic branching as the negated concepts introduced by failed branches can result in additional simplifications. Taking the above example of $\{(A \sqcup B), (A \sqcup C)\} \subseteq \mathcal{L}(x)$, adding $\neg A$ to $\mathcal{L}(x)$ allows BCP to deterministically expand both of the disjunctions using the simplifications $(A \sqcup B)$ and $\neg A \rightarrow B$ and $(A \sqcup C)$ and $\neg A \rightarrow C$. The reduced search space resulting from the combination of semantic branching and BCP is shown in Figure 7.

Local simplification has the advantage that it can never increase the size of the search space and can thus only degrade performance to the extent of the overhead required to perform the simplification. Minimising this overhead does, however, require complex data structures [20], particularly in a modal/description logic setting.

As with semantic branching, effectiveness is problem dependent, the optimisation being most effective with over-constrained randomly generated problems [33].

6.3 Dependency Directed Backtracking

Inherent unsatisfiability concealed in sub-problems can lead to large amounts of unproductive backtracking search, sometimes called thrashing. For example, expanding a node x (using semantic branching), where

$$\mathcal{L}(x) = \{(C_1 \sqcup D_1), \ldots, (C_n \sqcup D_n), \exists R.(A \sqcap B), \forall R.\neg A\},$$

could lead to the fruitless exploration of 2^n possible R-successors of x before the inherent unsatisfiability is discovered. The search tree resulting from the tableaux expansion is illustrated in Figure 8.

This problem can be addressed by adapting a form of dependency directed backtracking called *backjumping*, which has been used in solving constraint satisfiability problems [5] (a similar technique was also used in the HARP theorem prover [39]). Backjumping works by labelling each concept in a node label with a dependency set indicating the branching points on which it depends. A concept $C \in \mathcal{L}(x)$ depends on a branching point if C was added to $\mathcal{L}(x)$ at the branching point or if $C \in \mathcal{L}(x)$ was generated by an expansion rule (including simplification) that depends on another concept $D \in \mathcal{L}(y)$, and $D \in \mathcal{L}(y)$ depends on the branching point. A concept $C \in \mathcal{L}(x)$ depends on a concept $D \in \mathcal{L}(y)$ when C was added to $\mathcal{L}(x)$ by a deterministic expansion that used $D \in \mathcal{L}(y)$. For example, if $A \in \mathcal{L}(x)$ was derived from the

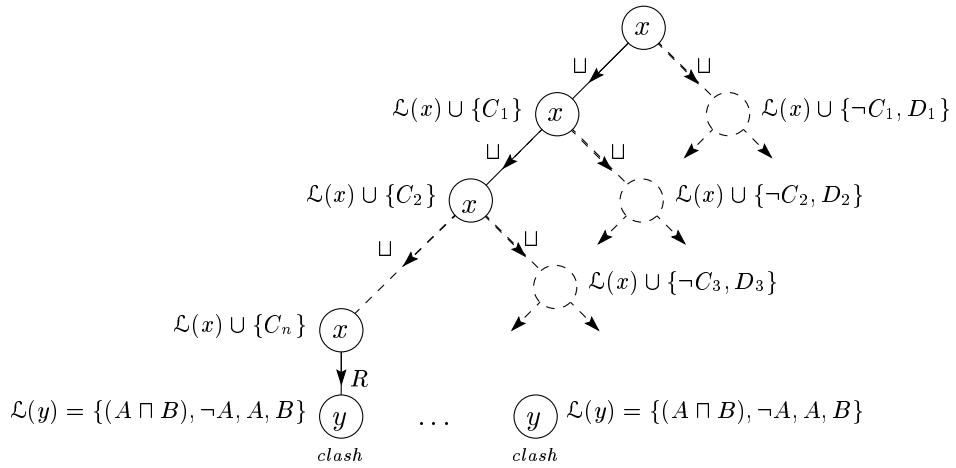


FIG. 8. Thrashing in backtracking search

expansion of $(A \sqcap B) \in \mathcal{L}(x)$, then $A \in \mathcal{L}(x)$ depends on $(A \sqcap B) \in \mathcal{L}(x)$.

When a clash is discovered, the dependency sets of the clashing concepts can be used to identify the most recent branching point where exploring the other branch might alleviate the cause of the clash. It is then possible to jump back over intervening branching points *without* exploring any alternative branches. Let us consider the earlier example and suppose that $\exists R.(A \sqcap B)$ has a dependency set $\mathbf{D}_i$ and $\forall R.\neg A$ has a dependency set $\mathbf{D}_j$. The search proceeds until $C_1 \ldots C_n$ have been added to $\mathcal{L}(x)$, when $\exists R.(A \sqcap B)$ and $\forall R.\neg A$ are deterministically expanded and a clash occurs in $\mathcal{L}(y)$ between the A derived from $\exists R.(A \sqcap B)$ and the $\neg A$ derived from $\forall R.\neg A$. As these derivations were both deterministic, the dependency sets will be $\mathbf{D}_i$ and $\mathbf{D}_j$ respectively, and so $\mathbf{D}_i \cup \mathbf{D}_j$ is returned. This set cannot include the branching points where $C_1 \ldots C_n$ were added to $\mathcal{L}(x)$ as $\mathbf{D}_i$ and $\mathbf{D}_j$ were defined before these branching points were reached. The algorithm can therefore backtrack through each of the preceding n branching points without exploring the second branches, and will continue to backtrack until it reaches the branching point equal to the maximum value in $\mathbf{D}_i \cup \mathbf{D}_j$ (if $\mathbf{D}_i = \mathbf{D}_j = \emptyset$, then the algorithm will backtrack through all branching points and return "unsatisfiable"). Figure 9 illustrates the pruned search tree, with the number of R-successors explored being reduced by an exponential number.

Backjumping can also be used with syntactic branching, but the procedure is slightly more complex as there may be more than two possible choices at a given branching point, and the dependency set of the disjunction being expanded must also be taken into account.

Like local simplification, backjumping can never increase the size of the search space. Moreover, it can lead to a dramatic reduction in the size of the search tree and thus a huge performance improvement. For example, when using either FaCT or DLP with backjumping disabled in order to classify a large (≈3,000 concept) KB derived from the European GALEN project [42], single satisfiability tests were encountered that could not be solved even after several weeks of CPU time. Classifying the same KB with backjumping enabled takes less than 100s of CPU time for either FaCT or

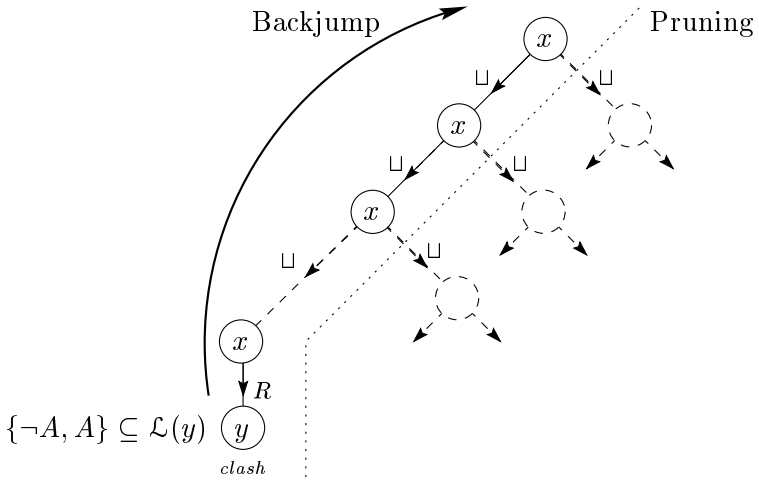


FIG. 9. Pruning the search using backjumping

DLP [34].

Backjumping's only disadvantage is the overhead of propagating and storing the dependency sets. This can be alleviated to some extent by using a pointer based implementation so that propagating a dependency set only requires the copying of a pointer.

6.4 Heuristic Guided Search

Heuristic techniques can be used to guide the search in a way that tries to minimise the size of the search tree. A method that is widely used in DPL SAT algorithms is to branch on the disjunct that has the *Maximum number of Occurrences in disjunctions of Minimum Size*—the well known MOMS heuristic [20]. By choosing a disjunct that occurs frequently in small disjunctions, the MOMS heuristic tries to maximise the effect of BCP. For example, if the label of a node x contains the unexpanded disjunctions $C \sqcup D_1, \ldots, C \sqcup D_n$, then branching on C leads to their deterministic expansion in a single step: when C is added to $\mathcal{L}(x)$, all of the disjunctions are fully expanded and when $\neg C$ is added to $\mathcal{L}(x)$, BCP will expand all of the disjunctions, causing $D_1, \ldots, D_n$ to be added to $\mathcal{L}(x)$. Branching first on any of $D_1, \ldots, D_n$, on the other hand, would only cause a single disjunction to be expanded.

The MOMS value for a candidate concept C is computed simply by counting the number of times C or its negation occur in minimally sized disjunctions. There are several variants of this heuristic, including the heuristic from Jeroslow and Wang [36]. The Jeroslow and Wang heuristic considers all occurrences of a disjunct, weighting them according to the size of the disjunction in which they occur. The heuristic then selects the disjunct with the highest overall weighting, again with the objective of maximising the effect of BCP and reducing the size of the search tree.

When a disjunct C has been selected from the disjunctions in $\mathcal{L}(x)$, a BCP maximising heuristic can also be used to determine the order in which the two possible

branches, $\mathcal{L}(x) \cup \{C\}$ and $\mathcal{L}(x) \cup \{\neg C\}$, are explored. This is done by separating the two components of the heuristic weighting contributed by occurrences of C and $\neg C$, trying $\mathcal{L}(x) \cup \{C\}$ first if C made the *smallest* contribution, and trying $\mathcal{L}(x) \cup \{\neg C\}$ first otherwise. The intention is to prune the search tree by maximising BCP in the first branch.

Unfortunately MOMS-style heuristics can interact adversely with the backjumping optimisation because they do not take dependency information into account. This was first discovered in the FaCT system, when it was noticed that using MOMS heuristic often led to much worse performance. The cause of this phenomenon turned out to be the fact that, without the heuristic, the data structures used in the implementation naturally led to "older" disjunctions (those dependent on earlier branching points) being expanded before "newer" ones, and this led to more effective pruning if a clash was discovered. Using the heuristic disturbed this ordering and reduced the effectiveness of backjumping [29].

Moreover, MOMS-style heuristics are of little value themselves in description logic systems because they rely for their effectiveness on finding the same disjuncts recurring in multiple unexpanded disjunctions: this is likely in hard propositional problems, where the disjuncts are propositional variables, and where the number of different variables is usually small compared to the number of disjunctive clauses (otherwise problems would, in general, be trivially satisfiable); it is unlikely in concept satisfiability problems, where the disjuncts are (possibly non-atomic) concepts, and where the number of different concepts is usually large compared to the number of disjunctive clauses. As a result, these heuristics will often discover that all disjuncts have similar or equal priorities, and the guidance they provide is not particularly useful.

An alternative strategy is to employ an *oldest-first* heuristic that tries to maximise the effectiveness of backjumping by using dependency sets to guide the expansion [34]. When choosing a disjunct on which to branch, the heuristic first selects those disjunctions that depend on the least recent branching points (i.e., those with minimal maximum values in their dependency sets), and then selects a disjunct from one of these disjunctions. This can be combined with the use of a BCP maximising heuristic, such as the Jeroslow and Wang heuristic, to select the disjunct from amongst the selected disjunctions.

The oldest-first heuristic can also be used to advantage when selecting the order in which existential role restrictions, and the labels of the R-successors which they generate, are expanded. One possible technique is to use the heuristic to select an unexpanded existential role restriction $\exists R.C$ from the label of a node x, apply the $\exists$-rule and the $\forall$-rule as necessary, and expand the label of the resulting R-successor. If the expansion results in a clash, then the algorithm will backtrack; if it does not, then continue selecting and expanding existential role restrictions from $\mathcal{L}(x)$ until it is fully expanded. A better technique is to first apply the $\exists$-rule and the $\forall$-rule exhaustively, creating a set of successor nodes. The order in which to expand these successors can then be based on the minimal maximum values in the dependency sets of all the concepts in their label, some of which may be due to universal role restrictions in $\mathcal{L}(x)$.

The main advantage of heuristics is that they can be used to complement other optimisations. The MOMS and Jeroslow and Wang heuristics, for example, are designed to increase the effectiveness of BCP while the oldest-first heuristic is designed

to increase the effectiveness of backjumping. They can also be selected and tuned to take advantage of the kinds of problem that are to be solved (if this is known). The BCP maximisation heuristics, for example, are generally quite effective with large randomly generated and hand crafted problems, whereas the oldest-first heuristic is more effective when classifying realistic KBs.

Unfortunately heuristics also have several disadvantages. They can add a significant overhead as the heuristic function may be expensive to evaluate and may need to be reevaluated at each branching point. Moreover, they may not improve performance, and may significantly degrade it, for example by interacting adversely with other optimisations, by increasing the frequency with which pathological worst cases can be expected to occur in generally easy problem sets.

6.5 Caching Satisfiability Status

During a satisfiability check there may be many successor nodes created. Some of these nodes can be very similar, particularly as the labels of the R-successors for a node x each contain the same concepts derived from the universal role restrictions in $\mathcal{L}(x)$. Systems such as DLP take advantage of this similarity by caching the satisfiability status of the sets of concepts with which node labels are initialised when they are created. The tableaux expansion of a node can then be avoided if the satisfiability status of its initial set of concepts is found in the cache.

However, this technique depends on the logic having the property that the satisfiability of a node is completely determined by its initial label set, and, due to the possible presence of inverse roles, $\mathcal{SI}$ logics do not have this property. For example, if the expansion of a node x generates an R-successor node y, with $\mathcal{L}(y) = \{\forall R^-.C\}$, then the satisfiability of y clearly also depends on the set of concepts in $\mathcal{L}(x)$. Similar problems could arise in the case where $\mathcal{L}(y)$ contains number restriction concepts.

If it is possible to solve these problems, then caching may be a very effective technique for $\mathcal{SI}$ logics, as it has been shown to be in the DLP system with a logic that does not support inverse roles. Caching is particularly useful in KB classification as cached values can be retained across multiple satisfiability tests. It can also be effective with both satisfiable and unsatisfiable problems, unlike many other optimisation techniques that are primarily aimed at speeding up the detection of unsatisfiability.

The main disadvantage with caching is the storage overhead incurred by retaining node labels (and perhaps additional information in the case of $\mathcal{SI}$ logics) and their satisfiability status throughout a satisfiability test (or longer, if the results are to be used in later satisfiability tests). An additional problem is that it interacts adversely with the backjumping optimisation as the dependency information required for backjumping cannot be effectively calculated for nodes that are found to be unsatisfiable as a result of a cache lookup. Although the set of concepts in the initial label of such a node is the same as that of the expanded node whose (un)satisfiability status has been cached, the dependency sets attached to the concepts that made up the two labels may not be the same. However, a weaker form of backjumping can still be performed by taking the dependency set of the unsatisfiable node to be the union of the dependency sets from the concepts in its label.

7 Discussion

A new DL system is being implemented based on the $\mathcal{SHIQ}$ algorithm we have developed from the $\mathcal{SHIF}$-algorithm described in Section 4.1 [28]. Pending the completion of this project, the existing FaCT system [31] has been modified to deal with inverse roles using the $\mathcal{SHIF}$ blocking strategy, the resulting system being referred to as iFaCT.

iFaCT has been used to conduct some initial experiments with a terminology representing (fragments of) database schemata and inter schema assertions from a data warehousing application [10] (a slightly simplified version of the proposed encoding was used to generate $\mathcal{SHIF}$ terminologies). iFaCT is able to classify this terminology, which contains 19 concepts and 42 axioms, in less than 0.1s of (266MHz Pentium) CPU time. In contrast, eliminating inverse roles using an embedding technique [11] gives an equisatisfiable FaCT terminology with an additional 84 axioms, but one which FaCT is unable to classify in 12 hours of CPU time. As discussed in Section 3, an extension of the embedding technique can be used to eliminate number restrictions [14], but requires a target logic which supports the transitive *closure* of roles, i.e., *converse*-PDL. The even larger number of axioms that this embedding would introduce makes it unlikely that tractable reasoning could be performed on the resulting terminology. Moreover, we are not aware of any algorithm for *converse*-PDL which does not employ a so-called *look behind analytical cut* [16], the application of which introduces considerable additional non-determinism. It seems inevitable that this would lead to a further degradation in empirical tractability.

The DL $\mathcal{SHIQ}$ will allow the above mentioned encoding of database schemata to be fully captured using qualified number restrictions. Future work will include completing the implementation of the $\mathcal{SHIQ}$ algorithm, testing its behaviour in this kind of application and investigating new techniques for improving its empirical tractability.

References

[1] F. Baader, H.-J. Bürckert, B. Nebel, W. Nutt, and G. Smolka. On the expressivity of feature logics with negation, functional uncertainty, and sort equations. *Journal of Logic, Language and Information*, 2:1–18, 1993.

[2] F. Baader, M. Buchheit, and B. Hollunder. Cardinality restrictions on concepts. *Artificial Intelligence*, 88(1–2):195–213, 1996.

[3] F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. Technical Report RR-90-13, DFKI, Kaiserslautern, Deutschland, 1990. An abridged version appeared in *Proc. of IJCAI-91*, pp. 446–451.

[4] F. Baader and U. Sattler. Number restrictions on complex roles in description logics. In L. C. Aiello, J. Doyle, and S. C. Shapiro, editors, *Proc. of KR'96*, pages 328–339. Morgan Kaufmann Publishers, San Francisco, California, November 1996.

[5] A. B. Baker. *Intelligent Backtracking on Constraint Satisfaction Problems: Experimental and Theoretical Results*. PhD thesis, University of Oregon, 1995.

[6] P. Balsiger and A. Heuerding. Comparison of theorem provers for modal logics — introduction and summary. vol. 1397 of *LNAI*, pages 25–26. Springer-Verlag, 1998.

[7] R. Berger. The undecidability of the dominoe problem. *Mem. Amer. Math. Soc.*, 66, 1966.

[8] M. Buchheit, F. M. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. *J. of Artificial Intelligence Research*, 1:109–138, 1993.

[9] D. Calvanese, M. Lenzerini, and D. Nardi. A unified framework for class based representation formalisms. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proc. of KR-94*, pages 109–120, Bonn, 1994. M. Kaufmann, Los Altos.

[10] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Source integration in data warehousing. In *Proc, of DEXA-98*, pages 192–197. IEEE Computer Society Press, 1998.

[11] D. Calvanese, G. De Giacomo, and R. Rosati. A note on encoding inverse roles and functional restrictions in $\mathcal{ALC}$ knowledge bases. In Franconi et al. [19].

[12] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

[13] G. De Giacomo and M. Lenzerini. Description logics with inverse roles, functional restrictions, and n-ary relations. vol. 838 of *LNAI*, pages 332–346. Springer-Verlag, 1994.

[14] G. De Giacomo and M. Lenzerini. What's in an aggregate: Foundations for description logics with tuples and sets. In *Proc. of IJCAI-95*, pages 801–807, 1995.

[15] G. De Giacomo and M. Lenzerini. Tbox and Abox reasoning in expressive description logics. In *Proc. of KR-96*, pages 316–327. M. Kaufmann, Los Altos, 1996.

[16] G. De Giacomo and F. Massacci. Combining deduction and model checking into tableaux and algorithms for Converse-PDL. *Information and Computation*, 1999. To appear.

[17] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In G. Brewka, editor, *Foundation of Knowledge Representation*. CSLI Publication, Cambridge University Press, 1996.

[18] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *J. of Computer and System Science*, 18:194–211, 1979.

[19] E. Franconi, G. De Giacomo, R. M. MacGregor, W. Nutt, C. A. Welty, and F. Sebastiani, editors. *Proc. of DL'98*. CEUR, May 1998.

[20] J. W. Freeman. *Improvements to propositional satisfiability search algorithms*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1995.

[21] J. W. Freeman. Hard random 3-SAT problems and the Davis-Putnam procedure. *AIJ*, 81:183–198, 1996.

[22] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K. In *Proc. of CADE-96*, LNAI, New Brunswick, NJ, USA, 1996.

[23] V. Haarslev and R. Möller. RACE system description. In Lambrix et al. [38], pages 130–132.

[24] A. Heuerding and S. Schwendimann. A benchmark method for the propositional modal logics K, KT, and S4. Technical report IAM-96-015, University of Bern, Switzerland, 1996.

[25] B. Hollunder, W. Nutt, and M. Schmidt-Schauß. Subsumption algorithms for concept description languages. In *Proc. of ECAI-90*, Pitman Publishing, London, 1990.

[26] B. Hollunder and F. Baader. Qualifying number restrictions in concept languages. In *Proc. of KR-91*, pages 335–346, Boston, MA, USA, 1991.

[27] I. Horrocks, U. Sattler, and S. Tobies. A PSPACE-algorithm for deciding $\mathcal{ALCI}_{R^+}$-satisfiability. Technical Report 98-08, LuFg Theoretical Computer Science, RWTH Aachen, 1998.

[28] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proc. of LPAR'99*, pages 161–180, 1999.

[29] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

[30] I. Horrocks. The FaCT system. In H. de Swart, editor, *Proc. of Tableaux'98*, number 1397 in LNAI, pages 307–312. Springer-Verlag, Berlin, May 1998.

[31] I. Horrocks. Using an expressive description logic: FaCT or fiction? In A. G. Cohn, L. Schubert, and S. C. Shapiro, editors, *Proc. of KR-98*, pages 636–647. Morgan Kaufmann Publishers, San Francisco, California, June 1998.

[32] I. Horrocks. FaCT and iFaCT. In Lambrix et al. [38], pages 133–135.

[33] I. Horrocks and P. F. Patel-Schneider. Comparing subsumption optimizations. In Franconi et al. [19], pages 90–94.

[34] I. Horrocks and P. F. Patel-Schneider. Optimising description logic subsumption. *J. of Logic and Computation*, 9(3):267–293, 1999.

[35] I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. *J. of Logic and Computation*, 9(3):385–410, 1999.

[36] R. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

[37] D. Kozen and J. Tiuryn. Logics of programs. In J. V. Leeuwen, editor, *Handbook of Theoretical Computer Science – Formal Models and Semantics*, pages 789–840. Elsevier Science Publishers (North-Holland), Amsterdam, 1990.

[38] P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, editors. *Proc. of DL'99*, 1999.

[39] F. Oppacher and E. Suen. HARP: A tableau-based theorem prover. *J. of Automated Reasoning*, 4:69–100, 1988.

[40] P. F. Patel-Schneider. DLP system description. In Franconi et al. [19], pages 87–89.

[41] V. R. Pratt. Models of program logic. *Proc. of FOCS-79*, pages 115–122, 1979.

[42] A. L. Rector, W. A. Nowlan, and A. Glowinski. Goals for concept representation in the GALEN project. In *Proc. of SCAMC'93*, pages 414–418, Washington DC, USA, 1993.

[43] U. Sattler. A concept language extended with different kinds of transitive roles. In *20. Deutsche Jahrestagung für KI*, vol. 1137 of *LNAI*. Springer-Verlag, 1996.

[44] K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of IJCAI-91*, pages 466–471, Sydney, 1991.

[45] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[46] E. Spaan. The complexity of propositional tense logics. In M. de Rijke, editor, *Diamonds and Defaults*, pages 287–307. Kluwer Academic Publishers, Dordrecht, 1993.

Received February 15, 2000