A PSPACE-algorithm for deciding $\mathcal{ALCNI}_{R^+}$-satisfiability

Ian Horrocks Ulrike Sattler Stephan Tobies

LTCS-Report 98-08

A PSPACE-algorithm for deciding $\mathcal{ALCNI}_{R^+}$-satisfiability[*]

Ian Horrocks[†] Ulrike Sattler[‡] Stephan Tobies[‡]

July 27, 2000

Abstract

$\mathcal{ALCNI}_{R^+}$—$\mathcal{ALCN}$ augmented with transitive and inverse roles—is an expressive Description Logic which is especially well-suited for the representation of complex, aggregated objects. Despite its expressiveness, it has been conjectured that concept satisfiability for this logic could be decided in a comparatively efficient way. In this paper we prove the correctness of this conjecture by presenting a PSPACE algorithm for deciding satisfiability and subsumption of $\mathcal{ALCNI}_{R^+}$-concepts. The space-efficiency of this tableau-based algorithm is due to a sophisticated guidance of the search for a solution. Moreover, this space-efficiency is not paid for with time-consumption; on the contrary, the guidance technique leads to very early refutation. This algorithm will be the basis for an efficient implementation.

[*]This work was partially supported by the Esprit Project 22469 – DWQ and the DFG.
[†]Department of Computer Science, University of Manchester.
[‡]Research Group for Theoretical Computer Science, University of Technology Aachen.

1 Syntax and Semantics of $\mathcal{ALCI}_{R^+}$

We start by introducing the Description Logic (DL) $\mathcal{ALCI}_{R^+}$, which is the extension of the well-known DL $\mathcal{ALC}$ [SSS91] with *transitively closed roles* and *inverse* (converse) roles. The set of transitive role names $\mathbf{R}_+$ is a subset of the set of role names $\mathbf{R}$. Interpretations map role names to binary relations on the interpretation domain, and transitive role names to transitive relations. In addition, for any role $R \in \mathbf{R}$, the role R^- is interpreted as the inverse of R.

Definition 1 Let N_C be a set of *concept names* and let $\mathbf{R}$ be a set of *role names* with transitive role names $\mathbf{R}_+ \subseteq \mathbf{R}$. The set of $\mathcal{ALCI}_{R^+}$*-roles* is $\mathbf{R} \cup \{R^- \mid R \in \mathbf{R}\}$. The set of $\mathcal{ALCI}_{R^+}$*-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 $\mathcal{ALCI}_{R^+}$-role, then $(C \sqcap D)$, $(C \sqcup D)$, $(\neg C)$, $(\forall R.C)$, and $(\exists R.C)$ are concepts. Both $\top$ and $\bot$ are also concepts.

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, the properties in Figure 1 are satisfied.

$$\begin{aligned}
\top^\mathcal{I} &= \Delta^\mathcal{I}\\
\bot^\mathcal{I} &= \emptyset\\
(C \sqcap D)^\mathcal{I} &= C^\mathcal{I} \cap D^\mathcal{I},\\
(C \sqcup D)^\mathcal{I} &= C^\mathcal{I} \cup D^\mathcal{I},\\
\neg C^\mathcal{I} &= \Delta^\mathcal{I} \setminus C^\mathcal{I},\\
(\exists S.C)^\mathcal{I} &= \{x \in \Delta^\mathcal{I} \mid \text{There exists } y \in \Delta^\mathcal{I} \text{ with } \langle x, y\rangle \in S^\mathcal{I} \text{ and } y \in C^\mathcal{I}\},\\
(\forall S.C)^\mathcal{I} &= \{x \in \Delta^\mathcal{I} \mid \text{For all } y \in \Delta^\mathcal{I}, \text{ if } \langle x, y\rangle \in S^\mathcal{I}, \text{ then } y \in C^\mathcal{I}\},\\
\text{For } S \in \mathbf{R}:&\ \ \langle x, y\rangle \in S^\mathcal{I} \text{ iff } \langle y, x\rangle \in S^{-\mathcal{I}}, \text{ and}\\
\text{For } R \in \mathbf{R}_+:&\ \ \text{if } \langle x, y\rangle \in R^\mathcal{I} \text{ and } \langle y, z\rangle \in R^\mathcal{I}, \text{ then } \langle x, z\rangle \in R^\mathcal{I}.
\end{aligned}$$

Figure 1: Semantics of $\mathcal{ALCI}_{R^+}$-concepts

A concept C is called *satisfiable* iff there is some interpretation $\mathcal{I}$ such that $C^\mathcal{I} \neq \emptyset$. Such an interpretation is called a *model of* C. A concept D *subsumes* a concept C (written $C \sqsubseteq D$) iff $C^\mathcal{I} \subseteq D^\mathcal{I}$ holds for each interpretation $\mathcal{I}$. For an interpretation $\mathcal{I}$, an individual $x \in \Delta^\mathcal{I}$ is called an *instance* of a concept C iff $x \in C^\mathcal{I}$.

In order to make the following considerations easier, we introduce two functions on roles:

1. The inverse relation on roles is symmetric, and to avoid considering roles such as R^{--}, we define a function Inv which returns the inverse of a role. More precisely, $\mathsf{Inv}(R) = R^-$ if R is a role name, and $\mathsf{Inv}(R) = S$ if $R = S^-$.

2. Obviously, a role R is transitive if and only if $\mathsf{Inv}(R)$ is transitive. However, this may be established by either R or $\mathsf{Inv}(R)$ being in $\mathbf{R}_+$. We therefore define a function Trans which returns true iff R is a transitive role—regardless of whether it is a role name or the inverse of a role name. More precisely, $\mathsf{Trans}(R) = \mathrm{true}$ iff $R \in \mathbf{R}_+$ or $\mathsf{Inv}(R) \in \mathbf{R}_+$.

2 Blocking

Before we introduce the new notion of blocking which allows for a space efficient implementation of the $\mathcal{ALCI}_{R^+}$-algorithm, let us recall the reason why we employ blocking at all, and the additional complexities introduced by inverse roles. We will also examine the weaknesses in the previous version [HS98a] of blocking for the $\mathcal{ALCI}_{R^+}$-algorithm which lead to an inefficient use of space.

The algorithm presented in [HS98a] for deciding satisfiability of $\mathcal{ALCI}_{R^+}$-concepts used the tableaux method [HN90], 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 must satisfy, and each edge is labelled with a role name.

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 are added as required in order to satisfy $\exists R.C$ 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 valid 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.

Termination is not, however, 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 [Sat96]. For example, given a leaf 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 an identical label to 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 [Baa90; BDS93]. For logics without inverse roles, the general procedure is to check the label of each new node y, and if it is a *subset* [BBH96] of the label of an existing 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.[1] The validity of the cyclical model is an easy consequence of the fact that the $\exists R.C$ concept which y must satisfy must also be satisfied by x, because x's label is a superset of y's. Termination is guaranteed by the fact that all concepts in node labels are ultimately derived from the decomposition of D, so all node labels must be a subset of the subconcepts of D, and a cycle must therefore occur within a finite number of expansion steps.

2.1 Dynamic Blocking

Blocking is, however, more problematical when inverse roles are added to the logic, and a key feature of the algorithms presented in [HS98a] 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. Taking the above example of a node labelled $\{C, \exists R.C, \forall R.(\exists R.C)\}$, if the successor of this node were blocked by a node whose label additionally included $\forall R^{-}.\neg C$, then the cyclical model would clearly be invalid.

In [HS98a] this problem was 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.

Another difficulty introduced by inverse roles is the fact that it is no longer possible to establish a block on a once and for all basis when a new node is added

[1]For logics with a transitive closure operator it is necessary to check the validity of the cyclical model created by blocking [Baa90], but for logics that only support transitive roles the cyclical model is always valid [Sat96].

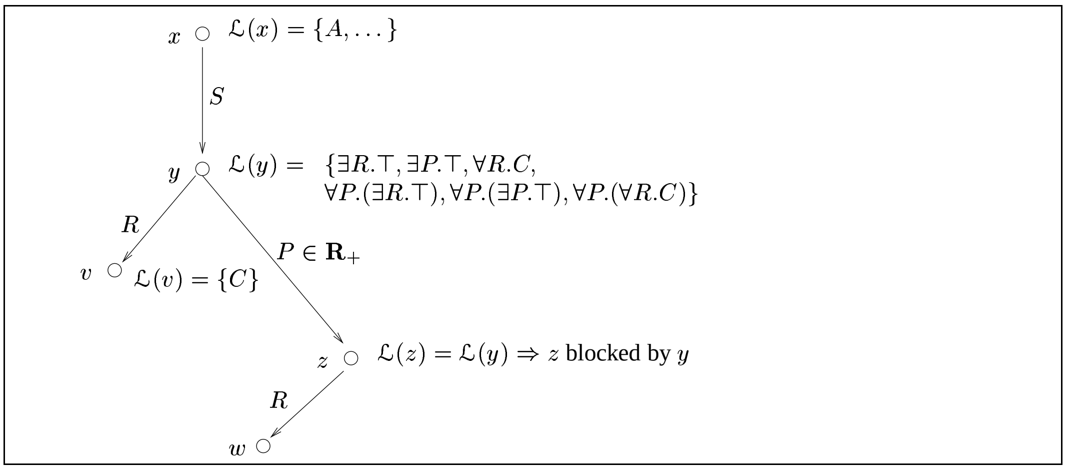
x
$\mathcal{L}(x) = \{A, \dots\}$
S
y
$\mathcal{L}(y) = \{\exists R.\top, \exists P.\top, \forall R.C, \forall P.(\exists R.\top), \forall P.(\exists P.\top), \forall P.(\forall R.C)\}$
R
$P \in \mathbf{R}_+$
v
$\mathcal{L}(v) = \{C\}$
z
$\mathcal{L}(z) = \mathcal{L}(y) \Rightarrow z$ blocked by y
R
w

Figure 2: A tableau where dynamic blocking is crucial

to the tree. This is because further expansion in other parts of the tree could lead to the labels of the blocking and/or blocked nodes being extended and the block being invalidated. Consider the example sketched in Figure 2, which shows parts of a tableau that was generated for the concept

$$A \sqcap \exists S.(\exists R.\top \sqcap \exists P.\top \sqcap \forall R.C \sqcap \forall P.(\exists R.\top) \sqcap \forall P.(\forall R.C) \sqcap \forall P.(\exists P.\top)),$$

where C represents the concept

$$\forall R^{-}.(\forall P^{-}.(\forall S^{-}.\neg A)).$$

This concept is clearly not satisfiable: w has to be an instance of C, which implies that x is an instance of $\neg A$—which is inconsistent with x being an instance of A.

As P is a transitive role, all universal value restrictions over P are propagated from y to z, hence $\mathcal{L}(y) = \mathcal{L}(z)$ and z is blocked by y. If the blocking of z were not subsequently broken when $\forall P^{-}.(\forall S^{-}.\neg A)$ is added to $\mathcal{L}(y)$ from $C \in \mathcal{L}(v)$, then $\neg A$ would never be added to $\mathcal{L}(x)$ and the inconsistency would not be detected.

As well as allowing blocks to be broken, it is also necessary to continue with some expansion of blocked nodes, because $\forall R.C$ concepts in their labels bcould affect other parts of the tree. Again, let us consider the example in Figure 2. After the blocking of z is broken and $\forall P^{-}.(\forall S^{-}.\neg A)$ is added to $\mathcal{L}(z)$ from $C \in \mathcal{L}(w)$, z is again blocked by y. However, the universal value restriction $\forall P^{-}.(\forall S^{-}.\neg A) \in \mathcal{L}(z)$ has to be expanded in order to detect the inconsistency.

These problems are overcome by using dynamic blocking: allowing blocks to be dynamically established and broken as the expansion progresses, and continuing to expand $\forall R.C$ concepts in the labels of blocked nodes.

2.2 Refined blocking

As mentioned before, in [HS98a] blocking of nodes was defined using label equality. This led to major problems when trying to establish a polynomial bound on the length of paths in the tree. If a node can only be blocked by an ancestor when the labels coincide, then there could be exponentially many ancestors in a path before blocking actually occurs. Due to the non-deterministic nature of the expansion rules, these subsets might actually be generated; the algorithm would then need to store the node labels of a path of exponential length, thus consuming exponential space.

This problem is already present when one tries to implement a tableau algorithm for the logic $\mathcal{ALC}_{R^+}$ [Sat96], where the non-deterministic nature of the expansion rules for disjunction might lead to the generation of a chain of exponential size before blocking occurs. Consider, for example, the concept

$$\begin{aligned} D &= \exists R.C \sqcap \forall R.(\exists R.C) \\ C &= (A_1 \sqcup B_1) \sqcap (A_2 \sqcup B_2) \sqcap \cdots \sqcap (A_n \sqcup B_n) \end{aligned}$$

where R is a transitive role. For $\mathcal{ALC}_{R^+}$, a polynomial bound on the path length is obtained by applying a simple strategy: a new successor is only generated when no other rule can be applied, and the $\sqcap$- and $\sqcup$-rules are only applied if neither the $\forall$-nor the $\forall_+$-rule can be applied. With this strategy, it is sufficient to test for (subset-)blocking before applying a $\sqcap$- or $\sqcup$-rule for the first time to a node x. Since no new concepts are propagated "upwards" in a completion tree, we can proceed in this way, effectively ignoringnon-deterministic choices due to propositional operators.

However, in the presence of inverse roles, this strategy is no longer possible. Indeed, the expansion rules for $\mathcal{ALCI}_{R^+}$ as they have been presented in [HS98b] based on set equality might already lead to a tableau with paths of exponential length—even if the tested concept does not contain any inverse roles. This is due to the fact that blocking is established on the basis of label equality and that disjunction is resolved in a non-deterministic way which could lead to the generation of 2^n different expansions of the concept C.

A possible solution to this problem is to generate new tableau nodes when expanding disjunction concepts [HM92; Sat96]. In this way, the labels of the nodes don't become cluttered with information which is not relevant for the correctness of blocking. However, in the presence of inverse roles, and when aiming for an efficient implementation, this solution is less then optimal: while in the $\mathcal{ALC}_{R^+}$-case it was possible to restrict the application of expansion rules to the leaves of the tree, this is no longer the case for $\mathcal{ALCI}_{R^+}$ since concepts of the form $\forall R.F$ may constrain both successor and predecessor nodes. If the concept F contains disjunction, this could necessitate the insertion of additional nodes inside the tree which would at least make the resulting tableaux algorithm less comprehensible.

In order to gain the benefits of the method sketched above without paying the conceptual overhead, we will keep the information which is relevant for blocking separated from the "irrelevant" information (due to propositional expansion) in a way which allows for a simple and comprehensible tableaux algorithm. In the following, we will explain this "separation" idea in detail, especially which rules propagate which kind of information.

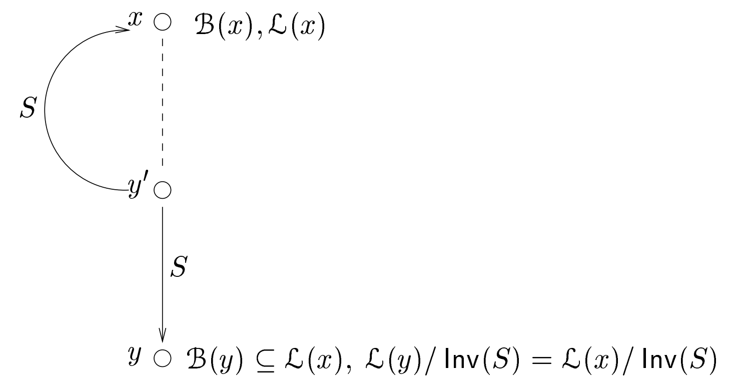
x
$\mathcal{B}(x), \mathcal{L}(x)$
S
y'
S
y
$\mathcal{B}(y) \subseteq \mathcal{L}(x),\ \mathcal{L}(y)/\,\mathsf{Inv}(S) = \mathcal{L}(x)/\,\mathsf{Inv}(S)$

Figure 3: A blocking situation

Figure 3 shows a blocking situation. Consider node y to be blocked by node x. When generating a model from this tree, the blocked node y will be omitted and y' will get x as an S-successor, which is indicated by the backwards arrow. On the one hand, this construction yields a new S-successor x of y', a situation which is taken care of by the subset blocking used in the normal $\mathcal{ALC}_{R^+}$ tableaux algorithms. On the other hand, x receives a new S^--successor y'. Now blocking has to make sure that, if a concept of the form $\forall S^-.C$ is in the label of x, then C (and $\forall S^-.C$ if S is a transitive role) occurs in the label of y'.

This was dealt with by equality blocking in [HS98a]. In the following algorithm it will be dealt with by extending the node labelling and changing the blocking condition into two conditions, one of which is a subset condition. In addition to the label $\mathcal{L}$, each node now has a second label $\mathcal{B}$, where the latter is always a subset of the former. The label $\mathcal{L}$ contains complete information, whereas $\mathcal{B}$ only contains information relevant to blocking. Thus, propositional consequences of concepts in $\mathcal{L}$ and concepts being propagated "upwards" in the tree are stored in $\mathcal{L}$ only. This allows the blocking conditions to be modified as follows:

- A first condition requires that $\mathcal{B}$ of the blocked node y be contained in the label $\mathcal{L}$ of the blocking node x. Due to the restriction to $\mathcal{B}$, this condition is less strict than equality blocking: expansions of disjunctions are only stored in $\mathcal{L}$ and thus cannot prevent a node from being blocked.

- A second condition takes care of the fact that the predecessor y' of the blocked node y becomes a new S^--successor of the blocking node x. It requires the S^--consequences of the blocked node y to be equal to the S^--consequences of the blocking node x, and hence deals with the extra problems which come with the introduction of inverse roles.

More precisely, the new definition of blocking reads as follows:
A node y is *blocked* if for some ancestor x, x is blocked or

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

where y' is the predecessor of y in the completion tree and $\mathcal{L}(\langle y', y\rangle) = S$. We define

$$\mathcal{L}(y)/\,\mathsf{Inv}(S) = \{\forall\,\mathsf{Inv}(S).C \in \mathcal{L}(y)\}.$$

Summing up, we build a completion tree in a way that, for all nodes x,

- we have $\mathcal{B}(x) \subseteq \mathcal{L}(x)$,
- $\mathcal{B}(x)$ contains only concepts which come *down* the tree,
- $\mathcal{L}(x)$ contains, additionally, all concepts which come *up* the tree, and
- expansion of disjunctions and conjunctions only affect $\mathcal{L}(x)$.

3 A Tableaux Algorithm for $\mathcal{ALCI}_{R^+}$

We now present a tableaux algorithm derived from the one presented in [HS98a]. We shape the rules in a way that allows for the separation of the concepts which are relevant for the two parts of the blocking condition. A slight modification of this algorithm will run using only polynomial space.

Like other tableaux algorithms, the $\mathcal{ALCI}_{R^+}$ algorithm tries to prove the satisfiability of a concept D by constructing a model of D. The model is represented by a so-called *completion tree*, a tree some of whose nodes correspond to individuals in the model, each node being labelled with two sets of $\mathcal{ALCI}_{R^+}$-concepts. When testing the satisfiability of an $\mathcal{ALCI}_{R^+}$-concept D, these sets are restricted to subsets of *sub*(D), where *sub*(D) is the set of subconcepts of D. Subconcepts are defined as follows:

$$\begin{array}{rcl}
\textit{sub}(A) & = & \{A\} \text{ for concept names } A \in N_C, \\
\textit{sub}(C \sqcap D) & = & \{C \sqcap D\} \cup \textit{sub}(C) \cup \textit{sub}(D), \\
\textit{sub}(C \sqcup D) & = & \{C \sqcup D\} \cup \textit{sub}(C) \cup \textit{sub}(D), \\
\textit{sub}(\forall R.C) & = & \{\forall R.C\} \cup \textit{sub}(C), \text{ and} \\
\textit{sub}(\exists R.C) & = & \{\exists R.C\} \cup \textit{sub}(C)
\end{array}$$

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 $\mathcal{ALCI}_{R^+}$-concept can easily be transformed to an equivalent one in NNF by pushing negations inwards [HN90].

The soundness and completeness of the algorithm will be proved by showing that it creates a *tableau* for D. We have chosen to take the (not so) long way round tableaux definition method for proving properties of tableaux algorithms because once tableaux are defined and Lemma 3 is proven the remaining proofs are considerably easier.

Definition 2 If D is an $\mathcal{ALCI}_{R^+}$-concept in NNF and $\mathbf{R}_D$ is the set of roles occurring in D, together with their inverses, a tableau T for D is defined to be a triple $(\mathbf{S}, \mathcal{L}, \mathcal{E})$ such that: $\mathbf{S}$ is a set of individuals, $\mathcal{L} : \mathbf{S} \to 2^{\mathit{sub}(D)}$ maps each individual to a set of concepts which is a subset of *sub*(D), $\mathcal{E} : \mathbf{R}_D \to 2^{\mathbf{S}\times\mathbf{S}}$ maps each role in $\mathbf{R}_D$ to a set of pairs of individuals, and there is some individual $s \in \mathbf{S}$ such that $D \in \mathcal{L}(s)$. For all $s, t \in \mathbf{S}$, $C, E \in sub(D)$, and $R \in \mathbf{R}_D$, it holds that:

1. $\bot \notin \mathcal{L}(s)$, and 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 $\forall R.C \in \mathcal{L}(s)$, $\langle s, t\rangle \in \mathcal{E}(R)$ and $\mathsf{Trans}(R)$, then $\forall R.C \in \mathcal{L}(t)$, and
7. $\langle s, t\rangle \in \mathcal{E}(R)$ iff $\langle t, s\rangle \in \mathcal{E}(\mathsf{Inv}(R))$.

Lemma 3 An $\mathcal{ALCI}_{R^+}$-concept D is satisfiable iff there exists a tableau for D.

Proof: For the *if* direction, if $T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ is a tableau for D with $D \in \mathcal{L}(s_0)$, a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of D can be defined as:

$$\begin{array}{rcl}
\Delta^{\mathcal{I}} & = & \mathbf{S} \\
A^{\mathcal{I}} & = & \{s \mid A \in \mathcal{L}(s)\} \quad \text{for all concept names A in } \mathit{sub}(D) \\
R^{\mathcal{I}} & = & \begin{cases} \mathcal{E}(R)^+ & \text{if } \mathsf{Trans}(R) \\ \mathcal{E}(R) & \text{otherwise} \end{cases}
\end{array}$$

where $\mathcal{E}(R)^+$ denotes the transitive closure of $\mathcal{E}(R)$. $D^{\mathcal{I}} \neq \emptyset$ because $s_0 \in D^{\mathcal{I}}$. Transitive roles are obviously interpreted as transitive relations. By induction on the structure of concepts, we show that, if $E \in \mathcal{L}(s)$, then $s \in E^{\mathcal{I}}$. Let $E \in \mathcal{L}(s)$.

1. If E is a concept name, then $s \in E^{\mathcal{I}}$ by definition.

2. If $E = \neg C$, then $C \notin \mathcal{L}(s)$ (due to Property 1 in Definition 2), so $s \in \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} = E^{\mathcal{I}}$.

3. If $E = (C_1 \sqcap C_2)$, then $C_1 \in \mathcal{L}(s)$ and $C_2 \in \mathcal{L}(s)$, so by induction $s \in C_1^{\mathcal{I}}$ and $s \in C_2^{\mathcal{I}}$. Hence $s \in (C_1 \sqcap C_2)^{\mathcal{I}}$.

4. The case $E = (C_1 \sqcup C_2)$ is analogous to 3.

5. If $E = (\exists S.C)$, then there is some $t \in \mathbf{S}$ such that $\langle s, t\rangle \in \mathcal{E}(S)$ and $C \in \mathcal{L}(t)$. By definition, $\langle s, t\rangle \in S^{\mathcal{I}}$ and by induction $t \in C^{\mathcal{I}}$. Hence $S \in (\exists S.C)^{\mathcal{I}}$.

6. If $E = (\forall S.C)$ and $\langle s, t\rangle \in S^{\mathcal{I}}$, then either

   (a) $\langle s, t\rangle \in \mathcal{E}(S)$ and $C \in \mathcal{L}(t)$, or

   (b) $\langle s, t\rangle \notin \mathcal{E}(S)$ and there exists a path of length $n \geq 1$ such that $\langle s, s_1\rangle$, $\langle s_1, s_2\rangle, \ldots, \langle s_n, t\rangle \in \mathcal{E}(S)$. Due to Property 6 in Definition 2, $\forall S.C \in \mathcal{L}(s_i)$ for all $1 \leqslant i \leqslant n$, and we have $C \in \mathcal{L}(t)$.

   In both cases, we have by induction $t \in C^{\mathcal{I}}$, and hence $s \in (\forall S.C)^{\mathcal{I}}$.

For the converse, if $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a model of D, then a tableau $T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ for D can be defined as:

$$\begin{array}{rcl} \mathbf{S} & = & \Delta^{\mathcal{I}} \\ \mathcal{E}(R) & = & R^{\mathcal{I}} \\ \mathcal{L}(s) & = & \{C \in \mathit{sub}(D) \mid s \in C^{\mathcal{I}}\} \end{array}$$

It only remains to demonstrate that T is a tableau for D:

1. T satisfies properties 1–5 in Definition 2 as a direct consequence of the semantics of $\mathcal{ALCI}_{R^+}$ concepts.

2. If $d \in (\forall R.C)^{\mathcal{I}}$, $\langle d, e\rangle \in R^{\mathcal{I}}$ and $\mathsf{Trans}(R)$, then $e \in (\forall R.C)^{\mathcal{I}}$ unless there is some f such that $\langle e, f\rangle \in R^{\mathcal{I}}$ and $f \notin C^{\mathcal{I}}$. However, if $\langle d, e\rangle \in R^{\mathcal{I}}$, $\langle e, f\rangle \in R^{\mathcal{I}}$ and $R \in \mathbf{R}_+$, then $\langle d, f\rangle \in R^{\mathcal{I}}$ and $d \notin (\forall R.C)^{\mathcal{I}}$. T therefore satisfies Property 6 in Definition 2.

3. T satisfies Property 7 in Definition 2 as a direct consequence of the semantics of inverse relations. ∎

3.1 Constructing an $\mathcal{ALCI}_{R^+}$ Tableau

From Lemma 3, an algorithm that constructs a tableau for an $\mathcal{ALCI}_{R^+}$-concept D can be used as a decision procedure for the satisfiability of D. Such an algorithm will now be described in detail.

The tableaux algorithm works on a *completion tree*. This is a tree in which each node x is labelled with two sets $\mathcal{L}(x)$ and $\mathcal{B}(x)$, where both sets are subsets of *sub*(D). Furthermore, each edge $\langle x, y\rangle$ of the tree is labelled $\mathcal{L}(\langle x, y\rangle) = R$ for some (possibly inverse) role R occurring in *sub*(D). Edges are added when expanding $\exists R.C$ and $\exists R^-.C$ terms; they correspond to relationships between pairs of individuals and are always directed from the root node to the leaf nodes. The algorithm expands the tree by extending $\mathcal{L}(x)$ (and possibly $\mathcal{B}(x)$) for some node x, or by adding new leaf nodes.

A completion tree $\mathbf{T}$ is said to contain a *clash* if, for a node x in $\mathbf{T}$, it holds that $\bot \in \mathcal{L}(x)$ or there is a concept C such that $\{C, \neg C\} \subseteq \mathcal{L}(x)$.

If nodes x and y are connected by an edge $\langle x, y\rangle$, then y is called a *successor* of x and x is called a *predecessor* of y. If $\mathcal{L}(\langle x, y\rangle) = R$, then y is called an R-*successor* of x and x is called an $\mathsf{Inv}(R)$-*predecessor* of y. *Ancestor* is the transitive closure of *predecessor* and *descendant* is the transitive closure of *successor*. A node y is called an R-*neighbour* of a node x if either y is an R-successor of x or y is an R-predecessor of x.

A node y is *blocked* if for some ancestor x, x is blocked or

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

where y' is the predecessor of y in the completion tree and $\mathcal{L}(\langle y', y\rangle) = S$. The set $\mathcal{L}(y)/\,\mathsf{Inv}(S)$ is defined by

$$\mathcal{L}(y)/\,\mathsf{Inv}(S) = \{\forall\,\mathsf{Inv}(S).C \in \mathcal{L}(y)\}.$$

The algorithm initialises a tree $\mathbf{T}$ to contain a single node x_0, called the *root* node, with $\mathcal{L}(x_0) = \mathcal{B}(x_0) = \{D\}$, where D is the concept to be tested for satisfiability. $\mathbf{T}$ is then expanded by repeatedly applying the rules from Figure 4.

The completion tree is *complete* when for some node x, $\mathcal{L}(x)$ contains a clash or when none of the rules is applicable. If, for an input concept D, the expansion rules can be applied in such a way that they yield a complete, clash-free completion tree, then the algorithm returns "D is *satisfiable*"; otherwise, the algorithm returns "D is *unsatisfiable*".

3.2 Soundness and Completeness

The soundness and completeness of the algorithm will be demonstrated by proving that, for an $\mathcal{ALCI}_{R^+}$-concept D, it always terminates and that it returns *satisfiable* if and only if D is satisfiable.

```
$\sqcap$-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 $C \in \mathcal{B}(y)$
                then create a new node $y$ with $\mathcal{L}(\langle x, y\rangle) = S$ and $\mathcal{L}(y) = \mathcal{B}(y) = \{C\}$
```

Figure 4: Tableaux expansion rules for $\mathcal{ALCI}_{R^+}$

Lemma 4 Let $\mathbf{T}$ be a completion tree obtained by applying the expansion rules to an $\mathcal{ALCI}_{R^+}$-concept D, then, for every node x in $\mathbf{T}$, $\mathcal{B}(x) \subseteq \mathcal{L}(x)$.

Proof: By a simple induction on the number of rule applications. ∎

Lemma 5 For each $\mathcal{ALCI}_{R^+}$-concept D, the tableaux algorithm terminates.

Proof: Let $m = |\mathit{sub}(D)|$. Obviously, m is linear in the length of D. Termination is a consequence of the following properties of the expansion rules:

1. The expansion rules never remove nodes from the tree or concepts from node labels.

2. Successors are only generated for concepts of the form $\exists R.C$, and for any node each of these concepts triggers the generation of at most one successor. Since $\mathit{sub}(D)$ contains at most m $\exists R.C$ concepts, the out-degree of the tree is bounded by m.

3. Nodes are labelled with nonempty subsets of $\mathit{sub}(D)$. If a path p is of length at least 2^{2m}, then there are 2 nodes x, y on p, with $\mathcal{L}(x) = \mathcal{L}(y)$ and $\mathcal{B}(x) = \mathcal{B}(y)$, and blocking occurs. Since a path on which nodes are blocked cannot become longer, paths are of length at most 2^{2m}. ■

Together with Lemma 3, the following lemma implies soundness of the tableaux algorithm.

Lemma 6 If the expansion rules can be applied to an $\mathcal{ALCCI}_{R^+}$-concept D such that they yield a complete and clash-free completion tree, then D has a tableau.

Proof: Let $\mathbf{T}$ be the complete and clash-free completion tree constructed by the tableaux algorithm for D. A tableau $T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ can be defined with:

$$\begin{aligned}
\mathbf{S} &= \{x \mid x \text{ is a node in } \mathbf{T}\text{, and } x \text{ is not blocked}\},\\
\mathcal{L} &= \text{the restriction of the labelling } \mathcal{L} \text{ in } \mathbf{T} \text{ to } \mathbf{S},\\
\mathcal{E}(R) &= \{\langle x, y\rangle \in \mathbf{S} \times \mathbf{S} \mid \begin{array}[t]{l} 1.\ y \text{ is an } R\text{-neighbour of } x \quad \textit{or}\\ 2.\ \mathcal{L}(\langle x, z\rangle) = R \text{ and } y \text{ blocks } z \quad \textit{or}\\ 3.\ \mathcal{L}(\langle y, z\rangle) = \mathsf{Inv}(R) \text{ and } x \text{ blocks } z\}, \end{array}
\end{aligned}$$

and it can be shown that T is a tableau for D:

1. $D \in \mathcal{L}(x_0)$ for the root x_0 of $\mathbf{T}$ and, as x_0 has no predecessors, it cannot be blocked. Hence $D \in \mathcal{L}(s)$ for some $s \in \mathbf{S}$.

2. Property 1 of Definition 2 is satisfied because $\mathbf{T}$ is clash-free.

3. Properties 2 and 3 of Definition 2 are satisfied because neither the $\sqcap$-rule nor the $\sqcup$-rule apply to any $x \in \mathbf{S}$.

4. Property 4 in Definition 2 is satisfied because for all $x \in \mathbf{S}$, if $\forall R.C \in \mathcal{L}(x)$ and $\langle x, y\rangle \in \mathcal{E}(R)$ we have three possible cases:

   (a) y is an R-neighbour of x. The $\forall$-rule guarantees $C \in \mathcal{L}(y)$.

   (b) $\mathcal{L}(\langle x, z\rangle) = R$, y blocks z. Then by the $\forall$-rule we have $C \in \mathcal{B}(z)$ and by the definition of blocking $\mathcal{B}(z) \subseteq \mathcal{L}(y)$. Hence $C \in \mathcal{L}(y)$.

   (c) $\mathcal{L}(\langle y, z\rangle) = \mathsf{Inv}(R)$, x blocks z. From the definition of blocking we have that $\mathcal{L}(z)/\,\mathsf{Inv}(\mathsf{Inv}(R)) = \mathcal{L}(x)/\,\mathsf{Inv}(\mathsf{Inv}(R))$. Hence $\forall R.C \in \mathcal{L}(z)$ and the $\forall$-rule guarantees $C \in \mathcal{L}(y)$.

5. Property 5 in Definition 2 is satisfied because for all $x \in \mathbf{S}$, if $\exists R.C \in \mathcal{L}(x)$, then the $\exists$-rule ensures that there is either:

(a) an R-predecessor y with $C \in \mathcal{B}(y) \subseteq \mathcal{L}(y)$ (see Lemma 4). Because y is a predecessor of x it cannot be blocked, so $y \in \mathbf{S}$ and $\langle y, x \rangle \in \mathcal{E}(R)$.

(b) an R-successor y with $C \in \mathcal{B}(y) \subseteq \mathcal{L}(y)$ (again, see Lemma 4). If y is not blocked, then $y \in \mathbf{S}$ and $\langle x, y \rangle \in \mathcal{E}(R)$. Otherwise, y is blocked by some z with $\mathcal{B}(y) \subseteq \mathcal{L}(z)$. Hence $C \in \mathcal{L}(z)$, $z \in \mathbf{S}$ and $\langle x, z \rangle \in \mathcal{E}(R)$.

6. Property 6 in Definition 2 is satisfied because for all $x \in \mathbf{S}$, if $\forall R.C \in \mathcal{L}(x)$ and $\langle x, y \rangle \in \mathcal{E}(R)$ and $\mathsf{Trans}(R)$ then we have three possible cases:

   (a) y is an R-neighbour of x. The $\forall_+$-rule guarantees $\forall R.C \in \mathcal{L}(y)$.

   (b) $\mathcal{L}(\langle x, z \rangle) = R$, y blocks z. Then by the $\forall_+$-rule we have $\forall R.C \in \mathcal{B}(z)$ and by the definition of blocking $\mathcal{B}(z) \subseteq \mathcal{L}(y)$. Hence $\forall R.C \in \mathcal{L}(y)$.

   (c) $\mathcal{L}(\langle y, z \rangle) = \mathsf{Inv}(R)$, x blocks z. From the definition of blocking we have that $\mathcal{L}(z)/R = \mathcal{L}(x)/R$. Hence $\forall R.C \in \mathcal{L}(z)$ and the $\forall_+$-rule guarantees $\forall R.C \in \mathcal{L}(y)$.

7. Property 7 in Definition 2 is satisfied because for each $\langle x, y \rangle \in \mathcal{E}(R)$, either:

   (a) x is an R-neighbour of y, so y is an $\mathsf{Inv}(R)$-neighbour of x and $\langle y, x \rangle \in \mathcal{E}(\mathsf{Inv}(R))$.

   (b) $\mathcal{L}(\langle x, z \rangle) = R$ and y blocks z, so $\mathcal{L}(\langle x, z \rangle) = \mathsf{Inv}(\mathsf{Inv}(R))$ and $\langle y, x \rangle \in \mathcal{E}(\mathsf{Inv}(R))$.

   (c) $\mathcal{L}(\langle y, z \rangle) = \mathsf{Inv}(R)$ and x blocks z, so $\langle y, x \rangle \in \mathcal{E}(\mathsf{Inv}(R))$. ∎

Lemma 7 If D has a tableau, then the expansion rules can be applied in such a way that the tableaux algorithm yields a complete and clash-free completion tree for D.

Proof: Let $T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ be a tableau for D. Using T, we trigger the application of the expansion rules such that they yield a completion tree $\mathbf{T}$ that is both complete and clash-free. We start with $\mathbf{T}$ consisting of a single node x_0, the root, with $\mathcal{B}(x_0) = \mathcal{L}(x_0) = \{D\}$.

T is a tableau, hence there is some $s_0 \in \mathbf{S}$ with $D \in \mathcal{L}(s_0)$. When applying the expansion rules to $\mathbf{T}$, the application of the non-deterministic $\sqcup$-rule is driven by the labelling in the tableau T. To this purpose, we define a mapping π which maps the nodes of $\mathbf{T}$ to elements of $\mathbf{S}$, and we steer the application of the $\sqcup$-rule such that $\mathcal{L}(x) \subseteq \mathcal{L}(\pi(x))$ holds for all nodes x of the completion tree.

More precisely, we define π inductively as follows:

- $\pi(x_0) = s_0$.

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

Figure 5: The $\sqcup'$-rule

- If $\pi(x_i) = s_i$ is already defined, and a successor y of x_i was generated for $\exists R.C \in \mathcal{L}(x_i)$, then $\pi(y) = t$ for some $t \in \mathbf{S}$ with $C \in \mathcal{L}(t)$ and $\langle s_i, t\rangle \in \mathcal{E}(R)$.

To make sure that we have $\mathcal{L}(x_i) \subseteq \mathcal{L}(\pi(x_i))$, we use the $\sqcup'$-rule given in Figure 5 instead of the $\sqcup$-rule. The expansion rules given in Figure 4 with the $\sqcup$-rule replaced by the $\sqcup'$-rule are called the *modified* expansion rules in the following.

It is easy to see that, if a tree $\mathbf{T}$ was generated using the modified expansion rules, then the expansion rules can be applied in such a way that they yield $\mathbf{T}$. Hence Lemma 6 and Lemma 5 still apply, and thus using the $\sqcup'$-rule instead of the $\sqcup$-rule preserves soundness and termination.

We will now show by induction that, if $\mathcal{L}(x) \subseteq \mathcal{L}(\pi(x))$ holds for all nodes x in $\mathbf{T}$, then the application of an expansion rule preserves this subset-relation. To start with, we clearly have $\{D\} = \mathcal{L}(x_0) \subseteq \mathcal{L}(s_0)$.

If the $\sqcap$-rule can be applied to x in $\mathbf{T}$ with $C = C_1 \sqcap C_2 \in \mathcal{L}(x)$, then C_1, C_2 are added to $\mathcal{L}(x)$. Since T is a tableau, $\{C_1, C_2\} \subseteq \mathcal{L}(\pi(x))$, and hence the $\sqcap$-rule preserves $\mathcal{L}(x) \subseteq \mathcal{L}(\pi(x))$.

If the $\sqcup'$-rule can be applied to x in $\mathbf{T}$ with $C = C_1 \sqcup C_2 \in \mathcal{L}(x)$, then $E \in \{C_1, C_2\}$ is in $\mathcal{L}(\pi(x))$, and E is added to $\mathcal{L}(x)$ by the $\sqcup'$-rule. Hence the $\sqcup'$-rule preserves $\mathcal{L}(x) \subseteq \mathcal{L}(\pi(x))$.

If the $\exists$-rule can be applied to x in $\mathbf{T}$ with $C = \exists R.C_1 \in \mathcal{L}(x)$, then $C \in \mathcal{L}(\pi(x))$ and there is some $t \in \mathbf{S}$ with $\langle \pi(x), t\rangle \in \mathcal{E}(R)$ and $C_1 \in \mathcal{L}(t)$. The $\exists$-rule creates a new successor y of x for which $\pi(y) = t$ for some t with $C_1 \in \mathcal{L}(t)$. Hence we have $\mathcal{L}(y) = \{C_1\} \subseteq \mathcal{L}(\pi(y))$.

If the $\forall$-rule can be applied to x in $\mathbf{T}$ with $C = \forall R.C_1 \in \mathcal{L}(x)$ and y is an R-neighbour of x, then $\langle \pi(x), \pi(y)\rangle \in \mathcal{E}(R)$, and thus $C_1 \in \mathcal{L}(\pi(y))$. The $\forall$-rule adds C_1 to $\mathcal{L}(y)$ and thus preserves $\mathcal{L}(x) \subseteq \mathcal{L}(\pi(x))$.

If the $\forall_+$-rule can be applied to x in $\mathbf{T}$ with $C = \forall R.C_1 \in \mathcal{L}(x)$, $\mathsf{Trans}(R)$, and y being an R-neighbour of x, then $\langle \pi(x), \pi(y)\rangle \in \mathcal{E}(R)$, and thus $\forall R.C_1 \in \mathcal{L}(\pi(y))$. The $\forall_+$-rule adds $\forall R.C_1$ to $\mathcal{L}(y)$ and thus preserves $\mathcal{L}(y) \subseteq \mathcal{L}(\pi(y))$.

Summing up, the tableau-construction triggered by T terminates with a complete tree, and since $\mathcal{L}(x) \subseteq \mathcal{L}(\pi(x))$ holds for all nodes x in $\mathbf{T}$, $\mathbf{T}$ is clash-free due to Property 1 of Definition 2. ∎

Theorem 8 The tableaux algorithm is a decision procedure for the satisfiability and subsumption of $\mathcal{ALCI}_{R^+}$-concepts.

Theorem 8 is an immediate consequence of Lemmata 3, 5, 6 and 7. Moreover, since $\mathcal{ALCI}_{R^+}$ is closed under negation, subsumption $C \sqsubseteq D$ can be reduced to the unsatisfiability of $C \sqcap \neg D$.

4 Complexity results

We will now turn our attention to the complexity of the tableaux algorithm in terms of memory consumption. We assume that the reader is familiar with the following complexity classes and the relationships between them:

Definition 9 $\text{DS}\textsc{pace}(f(m))$ is the class of all sets which are decidable by a *deterministic* Turing machine which needs no more then $O(f(m))$ space for an input of length m.

$\text{NS}\textsc{pace}(f(m))$ is the class of all sets which are decidable by a *non-deterministic* Turing machine which needs no more then $O(f(m))$ space for an input of length m.

$$\text{PS}\textsc{pace} = \bigcup_{i \in \mathbb{N}} \text{DS}\textsc{pace}(m^i)$$

$$\text{NPS}\textsc{pace} = \bigcup_{i \in \mathbb{N}} \text{NS}\textsc{pace}(m^i)$$

A set X is called *hard* for a complexity class $\mathcal{C}$, iff for all $Y \in \mathcal{C}$ there exists a reduction function χ which can be calculated in polynomial time such that

$$y \in Y \text{ iff } \chi(y) \in X.$$

Remark: When started with the input D, a non-deterministic Turing machine accepts D iff all runs terminate and there is at least one run that gives a positive answer. The input D is rejected iff all runs terminate and lead to a negative answer.

Obviously, the following inclusions hold:

$$\text{DS}\textsc{pace}(f(m)) \subseteq \text{NS}\textsc{pace}(f(m)) \quad \text{and} \quad \text{PS}\textsc{pace} \subseteq \text{NPS}\textsc{pace}$$

Furthermore, if a $\mathcal{C}$-hard set X can be reduced to a set X', then X' is also $\mathcal{C}$-hard.

Fact 10 ([SSS91]) $\text{SAT}(\mathcal{ALC}) = \{C \mid C \text{ is a satisfiable } \mathcal{ALC}\text{-concept}\}$ is PSPACE-complete.

In [Sat96], it was shown that $\mathcal{ALC}$ extended with transitive roles is still in PSPACE—in contrast to $\mathcal{ALC}$ extended with the *transitive closure of roles*, which yields EXPTIME-hardness. We claimed in [HS98c], and will prove in the following, that $\mathcal{ALC}$ with transitive and inverse roles is still in PSPACE. Moreover, this result is tightened in Section 5, where we show that number restrictions can be added to $\mathcal{ALCI}_{R^+}$ without leaving PSPACE.

For the moment, we are interested in the complexity class of the following set.

Definition 11 $\text{SAT}(\mathcal{ALCI}_{R^+}) = \{C \mid C \text{ is a satisfiable } \mathcal{ALCI}_{R^+}\text{-concept}\}$.

Since $\text{SAT}(\mathcal{ALC})$ is obviously reducible to $\text{SAT}(\mathcal{ALCI}_{R^+})$ we immediately get the following:

Theorem 12 $\text{SAT}(\mathcal{ALCI}_{R^+})$ is PSPACE-hard.

It remains to provide a PSPACE upper bound. Due to the following relationship it is sufficient to present a non-deterministic PSPACE algorithm in order to prove that a problem is in deterministic PSPACE.

Fact 13 (Savitch's Theorem, [Sav70]) PSPACE $=$ NPSPACE.

We will start by mentioning some basic facts which follow immediately by inspection of the tableau rules.

For each node x of the completion tree, $\mathcal{B}(x)$ only contains two kinds of concepts: the concept which triggered the generation of the node x, denoted by C_x, and concepts which were propagated *down* the completion tree by the first alternative of the $\forall$- and $\forall_+$-rules. Also $\mathcal{B}(x) \subseteq \mathcal{L}(x)$ holds for any node in the completion tree.

In Lemma 14 and 15, we establish a polynomial bound on the length of paths in the completion tree in a similar manner to that used for the modal logic $S4$ and $\mathcal{ALC}_{R^+}$ in [HM92; Sat96]. It then only remains to show that such a tree can be constructed using only polynomial space.

Lemma 14 Let $m = |\mathit{sub}(D)|$, $n \geq m^3$, and R be a role with $\mathsf{Trans}(R)$. Let $x_1, \ldots, x_n$ be successive nodes of a completion tree with $\mathcal{L}(\langle x_i, x_{i+1}\rangle) = R$ for $1 \leq i < n$. If the $\forall$- or the $\forall_+$-rule cannot be applied to these nodes, then there is a blocked x_i among them.

Proof: Firstly, consider the elements of $\mathcal{B}(x_i)$ for $i > 1$. Again, let C_{x_i} denote the concept that caused the generation of the node x_i. Then $\mathcal{B}(x_i) - \{C_{x_i}\}$ contains only concepts which have been inserted using the $\forall$-rule or the $\forall_+$-rule. Let $C \in \mathcal{B}(x_i) - \{C_{x_i}\}$. Then either $\forall R.C \in \mathcal{L}(x_{i-1})$ and the $\forall_+$-rule makes sure that $\forall R.C \in \mathcal{B}(x_i)$, or C is already of the from $\forall R.C'$ and has been inserted into $\mathcal{B}(x_i)$ by an application of the $\forall_+$-rule to x_{i-1}. In both cases it follows that the $\forall$- or the $\forall_+$-rule yield $C \in \mathcal{B}(x_{i+1})$. Hence we have

$$\mathcal{B}(x_i) - \{C_{x_i}\} \subseteq \mathcal{B}(x_{i+1}) \text{ for all } 1 \leq i < n,$$

which implies, since we have m choices for C_{x_i},

$$|\{\mathcal{B}(x_i) \mid 1 \leq i \leq n\}| \leq m^2.$$

Secondly, consider $\mathcal{L}(x_i)/\ \mathsf{Inv}(R)$. Again, the $\forall$- and the $\forall_+$-rule yield

$$\mathcal{L}(x_i)/\ \mathsf{Inv}(R) \subseteq \mathcal{L}(x_{i-1})/\ \mathsf{Inv}(R) \text{ for all } 1 < i \leq n,$$

which implies

$$|\{\mathcal{L}(x_i)/\ \mathsf{Inv}(R) \mid 1 \leq i \leq n\}| \leq m.$$

Summing up, within m^3 nodes there must be at least two nodes x_j, x_k which satisfy

$$\mathcal{B}(x_j) = \mathcal{B}(x_k) \quad \text{and} \quad \mathcal{L}(x_j)/\ \mathsf{Inv}(R) = \mathcal{L}(x_k)/\ \mathsf{Inv}(R).$$

This implies that one of these nodes is blocked by the other. ■

We will now use this lemma to give a polynomial bound on the length of paths in a completion tree generated by the tableaux rules.

Lemma 15 The paths of a completion tree for a concept D have a length of at most m^4 where $m = |\mathit{sub}(D)|$.

Proof: We define $\ell(x) = \max\{|C| \mid C \in \mathcal{L}(x)\}$, where $|C|$ denotes the length of the concept C. If x is an predecessor of y in the tree this implies $\ell(x) \geq \ell(y)$. If not $\mathsf{Trans}(R)$ and $\mathcal{L}(\langle x, y\rangle) = R$, then this implies $\ell(x) > \ell(y)$. Furthermore, for $R_1 \neq R_2$ (but possibly $R_1 = \mathsf{Inv}(R_2)$), $\mathcal{L}(\langle x, y\rangle) = R_1$ and $\mathcal{L}(\langle y, z\rangle) = R_2$ implies $\ell(x) > \ell(z)$.

The only way that the maximal length of concepts does not decrease is along a pure R-path with $\mathsf{Trans}(R)$. However, the $\forall$- and the $\forall_+$-rule must be applied before the $\exists$-rule may generate a new successor. Together with Lemma 14, this guarantees that these pure R-paths have a length of at most m^3.

Summing up, we can have a path of length at most m^3 before decreasing the maximal length of the concept in the node labels (or blocking occurs), which can happen at most m times and thus yields an upper bound of m^4 on the length of paths in a completion tree. ■

Note that the extra condition for the $\exists$-rule, which delays its application until no other rules are applicable, is necessary to prevent the generation of paths of exponential length. Consider the following example for some R with $\mathsf{Trans}(R)$:

$$\begin{aligned} D &= \exists R.C \sqcap \forall R.(\exists R.C) \sqcap \forall R^-.A_0 \\ C &= (\forall R^-.A_1 \sqcup \forall R^-.B_1) \sqcap \cdots \sqcap (\forall R^-.A_n \sqcup \forall R^-.B_n) \end{aligned}$$

When started with a root node x_0 labelled $\mathcal{B}(x_0) = \mathcal{L}(x_0) = \{D\}$, the tableaux algorithm generates a successor node x_1 with

$$\mathcal{B}(x_1) = \{C, \exists R.C, \forall R.(\exists R.C)\}$$

which, in turn, is capable of generating a further successor x_2 with $\mathcal{B}(x_2) = \mathcal{B}(x_1)$. This would lead to an infinite chain of nodes if it were not for blocking. Obviously, the first part of the blocking condition is satisfied since $\mathcal{B}(x_2) \subseteq \mathcal{B}(x_1)$. However, the second condition causes a problem since, in this example, we can generate 2^n different sets of R^--consequences for each node. If we can apply the $\exists$-rule freely, then the algorithm might generate all of these 2^n nodes to find out (after finally applying the $\forall_+$-rule) that x_2 is blocked by x_1.

4.1 The PSPACE algorithm

At this stage, it is possible to give a PSPACE decision procedure for SAT$(\mathcal{ALCI}_{R^+})$ by using a *cut rule* similar to the one in [GMar] for an EXPTIME-hard extension of $\mathcal{ALCI}_{R^+}$. However, for $\mathcal{ALCI}_{R^+}$, this technique is more non-deterministic than necessary and therefore only briefly sketched:

Whenever we process a new node x, we don't bother using the rules to calculate $\mathcal{L}(x)$ but just guess $\mathcal{L}(x)$ such that $\mathcal{B}(x) \subseteq \mathcal{L}(x) \subseteq \mathit{sub}(D)$. After that, we test if the $\sqcap$- or $\sqcup$-rules are applicable to x or if the $\forall$- and $\forall_+$-rules can be applied to x in a way that extends the labelling of its predecessor. If this is the case, we terminate the algorithm returning "D is unsatisfiable" (please recall the remark below Definition 9). If we find none of these rules to be applicable and $\mathcal{L}(x)$ to be clash-free, we start generating successors reusing space.

If there exists a complete and clash-free completion tree, then there is a run of this algorithm which will always guess correctly and hence find this clash-free completion tree. Since the length of the paths is limited by m^4 and we only need to keep one path in memory, this is a valid PSPACE-algorithm for deciding SAT($\mathcal{ALCI}_{R^+}$). However, an "efficient" implementation of this algorithm seems to be impossible due to its high degree of unguided non-determinism, which results from the guessing of arbitrary supersets of $\mathcal{B}(x)$ for $\mathcal{L}(x)$.

In this section, we will modify the expansion rules given in Figure 4 such that they yield a PSPACE algorithm, namely one which we believe will serve as a basis

```
$\forall'$-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 $\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{C\}$ and delete *all* descendants of $y$.

$\forall'_+$-rule: if 1. $\forall S.C \in \mathcal{L}(x)$
                     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 and $\forall S.C \notin \mathcal{L}(y)$
                  then $\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{\forall S.C\}$ and delete *all* descendants of $y$.
```

Figure 6: The $\forall'$- and $\forall'_+$-rules

for an "efficient" implementation. 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 bounded polynomially. 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. A simple trick (which essentially only uses the fact that we are dealing with a tree and that we never remove concepts from the labels) can overcome this problem:

Figure 6 shows the modified rules. Whenever a concept is added to the label of a predecessor, the rules delete the entire subtree below that predecessor—including the node from which this concept originated.

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

Lemma 16 For each $\mathcal{ALCI}_{R^+}$-concept D, the tableaux algorithm with the modified $\forall'$- and $\forall'_+$-rules terminates.

Proof: Let $m = |\mathit{sub}(D)|$. Again, the algorithm has the following properties:

1. Concepts are never removed from the labels of the nodes.

2. Successors are only generated for concepts of the form $\exists R.C$, and for any node each of these concepts triggers the generation of at most one successor. Since *sub*(D) contains at most m $\exists R.C$ concepts, the out-degree of the tree is bounded by m.

3. We have shown that a path in the completion tree will never become longer then m^4 nodes.

Let us assume the algorithm is non-terminating. Since the size of the completion tree is bounded, there have to be infinitely many deletions of subtrees in order to yield non-termination of the algorithm. Please note that each node is deleted at most once, but may trigger the deletion of its successors several times.

The root of the completion tree cannot be deleted because it has no predecessor. Hence there are nodes which are never deleted. Choose one of these nodes x with maximum distance from the root, i.e., which has a maximum number of predecessors. Suppose that x's successors are deleted only finitely many times. This cannot be the case because, after the last deletion of x's successors, the "new" successors were never deleted and thus x would not have maximum distance from the root. Hence x triggers the deletion of its successors infinitely many times. However, the $\forall$- and the $\forall_+$-rule are the only rules that lead to a deletion, and they simultaneously lead to an increase of $\mathcal{L}(x)$, namely by the missing concept which caused the deletion of x's successors. Since we never remove any concepts from the labels, this implies the existence of an infinitely increasing chain of subsets of *sub*(D), which is clearly impossible. ■

Theorem 17 SAT$(\mathcal{ALCI}_{R^+}) \in$ PSPACE.

Proof: Let $m = |\textit{sub}(D)|$. For each node x we can store the labels $\mathcal{L}(x)$ and $\mathcal{B}(x)$ using m bits for each set. We apply the expansion rules as given in Figure 4 and 6. If a clash is generated, we exit the algorithm and return "D is unsatisfiable" (please recall the remark below Definition 9). Otherwise, we can evaluate the completion in a depth-first way: we keep track of exactly one path of the completion tree by memorising, for each node x, which of the $\exists R.C$-concepts in $\mathcal{L}(x)$ successors have yet to be generated. This can be done using an additional m bits for each node. The "deletion" of all successors in the $\forall$- or the $\forall_+$-rule of a node x is then simply realised by setting all these additional bits to "has yet to be generated". There are three possible results of an investigation of a subtree below x:

- A clash is detected. This stops the algorithm with "D is unsatisfiable".
- The $\forall$- or the $\forall_+$-rule lead to an increase of $\mathcal{L}(x)$. We re-consider all subtrees below x, re-using the space used for former subtrees of x.
- Neither of these first two cases happen. We can then forget about this subtree and start the investigation of another subtree of x. If all subtrees have been investigated, we consider x's predecessor.

Proceeding like this, the algorithm can be implemented using $2m + m$ bits for each node, where the $2m$ bits are used to store the labels of the node, while m bits are used to keep track of the successors already generated. Since we reuse the memory for the successors, we only have to store one path of the completion tree at a time. From Lemma 15, the length of this path is bounded by m^4. Summing up, we can test for the existence of a completion tree using at most $\mathcal{O}(m^5)$ bits.

Unfortunately, due to the $\sqcup$-rule, we are dealing with a non-deterministic algorithm. However, Savitch's theorem tells us that there is a deterministic implementation of this algorithm using at most $\mathcal{O}(m^{10})$ bits, which is still a polynomial boundary. ∎

Theorem 12 and Theorem 17 imply:

Corollary 18 $\mathrm{SAT}(\mathcal{ALCI}_{R^+})$ is PSPACE-complete.

Unlike the algorithm using the cut rule, this algorithm seems to suggest an efficient implementation for an algorithm that decides the satisfiability of $\mathcal{ALCI}_{R^+}$ concepts. The only problems which have to be overcome are dealing with the non-determinism introduced by the $\sqcup$-rule and developing suitable optimisation techniques. Unlike other logics where this kind of non-determinism can be handled by an implementation using backtracking or back-jumping, in the presence of inverse roles things get more involved: not only do nodes in the completion tree influence nodes further down the tree (which are discarded during backtracking) but they also influence nodes further up the tree.

There is an immediate optimisation of the algorithm which has been omitted for the sake of the clarity of the presentation. We have only disallowed the application of the $\exists$-rule to a blocked node, which is sufficient to guarantee the termination of the algorithm. It is also possible to disallow the application of more rules to a blocked node without violating the soundness or the completeness of the algorithm, if the notion of blocking is slightly adapted. It then becomes necessary to distinguish directly and indirectly blocked nodes. More details can be found in [HS98b]. The technique presented there will stop the expansion of a blocked node earlier during the runtime of the algorithm and hence will save some work. The development of a suitable deterministic algorithm and suitable optimisation techniques will be part of future work, as will their implementation and evaluation.

It should be noted that, independently from the work presented in this paper, satisfiability of the tense modal logic $\mathbf{K4_t}$ has been shown to be PSPACE-complete [Spa93] using a technique similar to the refined blocking used here ($\mathbf{K4_t}$ is a syntactical variant of $\mathcal{ALCI}_{R^+}$ with only a single role name).

5 Number restrictions

A useful extension of $\mathcal{ALCI}_{R^+}$ is obtained by allowing, additionally, for number restrictions, a well-known means of restricting the number of role fillers for members of concepts. We now show that the above methods are also applicable to the resulting logic $\mathcal{ALCNI}_{R^+}$; that is, we show that reasoning in $\mathcal{ALCNI}_{R^+}$ is also in PSPACE.

5.1 Syntax and semantics of $\mathcal{ALCNI}_{R^+}$

The syntax of $\mathcal{ALCNI}_{R^+}$ is a simple extension of that of $\mathcal{ALCI}_{R^+}$.

Definition 19 The set of $\mathcal{ALCNI}_{R^+}$ concepts is obtained by adding the following rule to the Definition 1.

3. if $n \in \mathbb{N}$ and R is an $\mathcal{ALCI}_{R^+}$-role with $\neg\,\mathsf{Trans}(R)$ then $(\geq\ n\ R)$ and $(\leq\ n\ R)$ are concepts.

An $\mathcal{ALCNI}_{R^+}$ interpretation $\mathcal{I}$ must satisfy, in addition to Definition 1, the following equations:

$$(\geq\ n\ R)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid |\{y \in \Delta^{\mathcal{I}} \mid \langle x, y\rangle \in R^{\mathcal{I}}\}| \geq n\} \text{ and}$$
$$(\leq\ n\ R)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid |\{y \in \Delta^{\mathcal{I}} \mid \langle x, y\rangle \in R^{\mathcal{I}}\}| \leq n\}.$$

Due to the following equivalences, $\mathcal{ALCNI}_{R^+}$-concepts can easily be transformed into negation normal form:

$$\neg(\geq\ n\ R) \equiv \begin{cases} (\leq\ (n-1)\ R) & \text{if } n \geq 1, \\ \bot & \text{if } n = 0, \end{cases}$$
$$\neg(\leq\ n\ R) \equiv (\geq\ (n+1)\ R).$$

Moreover, we will not consider number restrictions of the form $(\geq\ 0\ R)$ or $(\leq\ 0\ R)$ since we suppose that they have been eliminated using the following equivalences:

$$(\leq\ 0\ R) \equiv \forall R.\bot$$
$$(\geq\ 0\ R) \equiv \top$$

The following definition extends the notion of a tableau to capture the semantics of number restrictions.

Definition 20 A tableau for an $\mathcal{ALCNI}_{R^+}$-concept D in NNF is defined in the same way as in Definition 2, with the additional properties:

8. if $(\geq\ n\ R) \in \mathcal{L}(s)$, then $|\{t \in \mathbf{S} \mid \mathcal{L}(\langle s, t\rangle) = R\}| \geq n$, and

9. if $(\leq\ n\ R) \in \mathcal{L}(s)$, then $|\{t \in \mathbf{S} \mid \mathcal{L}(\langle s, t\rangle) = R\}| \leq n$.

The proof of Lemma 3 can be adapted in a straightforward way to Lemma 20 and is therefore omitted here.

Lemma 21 An $\mathcal{ALCNI}_{R^+}$-concept D is satisfiable iff there exists a tableau for D.

5.2 Constructing an $\mathcal{ALCNI}_{R^+}$ Tableau

In the following, we modify the tableaux algorithm for $\mathcal{ALCI}_{R^+}$ in such a way that it yields a PSPACE decision procedure for $\mathcal{ALCNI}_{R^+}$. We start by investigating the blocking condition. Let us recall the definition of blocking for $\mathcal{ALCI}_{R^+}$:

A node y is *blocked* if for some ancestor x, x is blocked or

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

where y' is the predecessor of y in the completion tree and $\mathcal{L}(\langle y', y\rangle) = S$. We define

$$\mathcal{L}(y)/\,\mathsf{Inv}(S) := \{\forall\,\mathsf{Inv}(S).C \in \mathcal{L}(y)\}.$$

Unfortunately, this definition of blocking no longer works in the presence of number restrictions because a blocking node x may obtain an additional role successor—which might clash with an "at most" number restriction on x. Such a situation is shown in figure 7. Suppose y is blocked by x. When constructing the tableau from this completion tree, x becomes an S-neighbour of y'. Hence the tableau generated by this construction is not valid since x has two S^--neighbours.

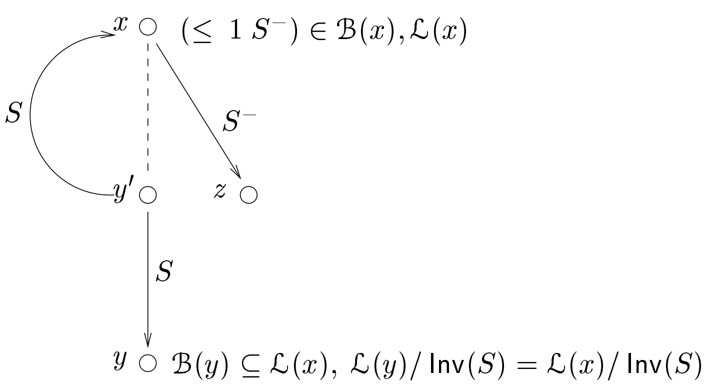
x
$(\leq\ 1\ S^-) \in \mathcal{B}(x), \mathcal{L}(x)$
S
S^-
y'
z
S
y
$\mathcal{B}(y) \subseteq \mathcal{L}(x),\ \mathcal{L}(y)/\,\mathsf{Inv}(S) = \mathcal{L}(x)/\,\mathsf{Inv}(S)$

Figure 7: A counterexample to refined blocking

There are at least two ways to overcome this problem. One is to employ a different technique for the construction of a tableau from a completion tree as has been done in [HS98b]. There, a valid tableau is constructed by "unraveling" the cycles introduced by blocking situations. Completion trees which contain blocked nodes will thus give rise to infinite tableaux. While this is necessary in the presence of role hierarchies as they have been studied in [HS98b] (since $\mathcal{ALCNI}_{R^+}$ augmented by role hierarchies lacks the finite model property), this is not necessary in the case of plain $\mathcal{ALCNI}_{R^+}$.

$\geq$-rule: if 1. $(\geq\ n\ S) \in \mathcal{L}(x)$ and
2. $n = 1$ and x has no S-neighbour, or
$n \geq 2$ and x has no S-successor, and
3. no other rule is applicable,
then create a new node y with $\mathcal{L}(\langle x, y\rangle) = S$ and $\mathcal{L}(y) = \mathcal{B}(y) = \{\}$

$\leq$-rule: if 1. $(\leq\ n\ S) \in \mathcal{L}(x)$ and
2. x has $m > n$ S-neighbours $y_1, \ldots, y_m$
then pick two nodes y_{i_1} and y_{i_2} such that y_{i_1} is not a predecessor of x,
set $\mathcal{L}(y_{i_2}) = \mathcal{L}(y_{i_2}) \cup \mathcal{L}(y_{i_1})$ and delete y_{i_1} from the tree.

Figure 8: Additional expansion rules for $\mathcal{ALCNI}_{R^+}$

We solve the problem by modifying the blocking condition. Problems can occur when a node x blocks a node y which is an S-successor of a node y' while $\mathcal{L}(x)$ contains a number restriction limiting the number of $\mathsf{Inv}(S)$-neighbours of x. We overcome this problem by disallowing the blocking of nodes which have an S-predecessor with $\neg\,\mathsf{Trans}(S)$. Since we don't allow transitive roles to appear in number restrictions, this solves the problem mentioned above.

The definition of a completion tree remains fundamentally the same, with only minor changes being incorporated. The new definition of blocking for the $\mathcal{ALCNI}_{R^+}$ algorithm reads as follows:

A node y is *blocked* if for some ancestor x, x is blocked or

- y is an S-successor of y',
- $\mathcal{B}(y) \subseteq \mathcal{L}(x)$ and $\mathcal{L}(y)/\,\mathsf{Inv}(S) = \mathcal{L}(x)/\,\mathsf{Inv}(S)$, and
- $\mathsf{Trans}(S)$.

A completion tree $\mathbf{T}$ is said to contain a *clash* if, for a node x in $\mathbf{T}$ and a concept C, $\{C, \neg C\} \subseteq \mathcal{L}(x)$, or for some $n, m \in \mathbb{N}$ with $n < m$:

$$\{(\leq\ n\ R), (\geq\ m\ R)\} \subseteq \mathcal{L}(x).$$

In addition to the expansion rules from figure 4, we introduce in Figure 8 two additional rules to cope with number restrictions .

5.2.1 Soundness and Completeness

The proof of soundness and completeness is quite similarly to the one for the $\mathcal{ALCI}_{R^+}$ tableaux algorithm. We start with Lemma 22 and 23 which imply termination.

Lemma 22 The paths of a completion tree for a $\mathcal{ALCNI}_{R^+}$-concept D have a length of at most m^4 where $m = |\textit{sub}(D)|$.

Proof: Lemma 14 still applies because it only talks about a chain of nodes connected by a transitive role for which the definition of blocking has not changed. Hence the proof of Lemma 15 is also still valid for the $\mathcal{ALCNI}_{R^+}$-case. ■

We also have to re-prove the termination of the algorithm. Since the $\leq$-rule in Figure 8 deletes nodes from the tree this is not as straightforward as in the $\mathcal{ALCI}_{R^+}$-case.

Lemma 23 For each $\mathcal{ALCNI}_{R^+}$-concept D, the tableaux algorithm terminates.

Proof: Let $m = |\textit{sub}(D)|$. Again, m is obviously linear in the length of D. Termination is a consequence of the following properties of the expansion rules:

1. The expansion rules never remove concepts from the labels.
2. The depth of the completion tree is limited by m^4.
3. Whenever the $\leq$-rule leads to the deletion of a node y, its label $\mathcal{L}(y)$ is added to a neighbour y' of its predecessor x. Assume that y had been generated by an application of the $\exists$-rule for a concept $\exists R.C_y \in \mathcal{L}(x)$. Then this rule can not be applied to x again for the same concept, since after the deletion $C_y \in \mathcal{L}(y')$. More generally, if x has an R-successor y which has been generated by an application of the $\exists$-rule to a concept $\exists R.C_y \in \mathcal{L}(x)$, then there will always be an R-neighbour y' of x such that $C_y \in \mathcal{L}(y')$. Hence the $\exists$-rule can be applied at most m times to a node in the completion tree. Also, the $\geq$-rule can be applied at most once to a node in the tree. ■

Unfortunately, the completion tree generated by the expansion rules cannot be transformed into a tableau as easily as in the $\mathcal{ALCI}_{R^+}$-case. This is due to the fact that, for constraints of the form $(\geq\ n\ R)$, the $\geq$-rule generates at most one successor. The next lemma shows that this suffices: If the constraints on the single successor do not yield a clash, then this successor (together with its successors) can be "copied" $n-1$ times to yield a tableau having "enough", namely n, R-successors.

Lemma 24 Let D be an $\mathcal{ALCNI}_{R^+}$-concept. Let $\mathbf{T}$ be a complete and clash-free completion tree for D. There exists a complete and clash free completion tree $\mathbf{T}'$ for D which satisfies: If x is a node of $\mathbf{T}'$ with $(\geq\ n\ R) \in \mathcal{L}(x)$, then

$$|\{y \mid y \text{ is } R\text{-neighbour of } x\}| \geq n. \qquad (\star)$$

Proof: Let k be the depth of the completion tree $\mathbf{T}$, i.e., the maximum length of a path in $\mathbf{T}$. We will prove this lemma by giving an algorithm which generates a sequence of completion trees $\mathbf{T}_0, \dots, \mathbf{T}_k$, with $\mathbf{T} = \mathbf{T}_0$ and $\mathbf{T}' = \mathbf{T}_k$, by adding "missing" neighbours. By induction we will show that the transformation preserves both completeness and clash-freeness.

$\mathbf{T}_{i+1}$ is obtained from $\mathbf{T}_i$ by applying the following transformation: For each $x \in \mathbf{T}_i$ such that

- x has a distance of $k - i$ to the root (that is, x has $k - 1$ ancestors),
- $\mathcal{L}(x)$ contains a number restriction $(\geq\ n\ R)$ with $n \geq 2$, and
- $|\{y \mid y \text{ is } R\text{-neighbour of } x\}| = \ell < n$

pick an arbitrary R-successor[2] y of x. Make $n - \ell$ disjoint copies of the subtree below and including y. Rename all nodes in the j-th of the copied subtrees by adding a subscript $_j$, thus obtaining a node z_j for each node z. The labellings of the edges and nodes in the j-th subtree are copied as well, that is, $\mathcal{L}(z_j) = \mathcal{L}(z)$, $\mathcal{B}(z_j) = \mathcal{B}(z)$, $\mathcal{L}(\langle z_j, w\rangle) = \mathcal{L}(\langle z, w\rangle)$, and $\mathcal{L}(\langle w, z_j\rangle) = \mathcal{L}(\langle w, z\rangle)$. Add these $n - \ell$ subtrees to $\mathbf{T}$.

Claim: This transformation preserves completeness and introduces no clashes.

Suppose on the contrary that there is an $x \in \mathbf{T}_i$ such that $\mathcal{L}(x)$ contains a clash and $\mathbf{T}_{i-1}$ is clash-free. Then either x was already a node in $\mathbf{T}_{i-1}$—which is a contradiction since then $\mathbf{T}_{i-1}$ would have contained a clash—or x is a copy of a node x' in $\mathbf{T}_{i-1}$. Since $\mathcal{L}(x) = \mathcal{L}(x')$, this would also imply that $\mathbf{T}_{i-1}$ is not clash-free, which again is a contradiction.

Now assume that $\mathbf{T}_i$ is not complete. If any of the $\{\sqcup, \sqcap, \forall, \forall_+, \geq\}$-rules were applicable, this would immediately lead to a contradiction. If the $\leq$-rule was applicable, then either it was already applicable in $\mathbf{T}_{i-1}$, leading to a contradiction, or the rule was not applicable before, which would imply that the generation of new successors would have enabled the rule. This cannot be the case since this would imply a clash of the form $\{(\geq\ n\ R), (\leq\ m\ R)\}$ with $n > m$ in $\mathbf{T}_{i-1}$.

By construction, $\mathbf{T}'$ satisfies the condition $(\star)$. ■

Lemma 25 If the expansion rules can be applied to an $\mathcal{ALCNI}_{R^+}$-concept D such that they yield a complete and clash-free completion tree, then D has a tableau.

Proof: Let $\mathbf{T}$ be such a completion tree for D. By Lemma 24 there exists a complete and clash-free completion tree $\mathbf{T}'$ for D which satisfies, additionally, $(\star)$. A tableau $T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ for D can be defined with

[2]Please note that such a successor must exist because, by induction, $\mathbf{T}_i$ is complete and hence the $\geq$-rule is not applicable.

$$\begin{aligned}
\mathbf{S} &= \{x \mid x \text{ is a node in } \mathbf{T}' \text{ and } x \text{ is not blocked}\},\\
\mathcal{L} &= \text{the restriction of the labelling } \mathcal{L} \text{ in } \mathbf{T}' \text{ to } \mathbf{S},\\
\mathcal{E}(R) &= \{\langle x,y\rangle \in \mathbf{S}\times\mathbf{S} \mid \begin{array}[t]{l} 1.\, y \text{ is an } R\text{-neighbour of } x \quad \textit{or}\\ 2.\, \mathcal{L}(\langle x,z\rangle) = R \text{ and } y \text{ blocks } z \quad \textit{or}\\ 3.\, \mathcal{L}(\langle y,z\rangle) = \mathsf{Inv}(R) \text{ and } x \text{ blocks } z\},\end{array}
\end{aligned}$$

It remains to show that T is indeed a tableau for D. The proof of Lemma 6 still applies. We only have to show that the additional properties involving number restrictions are satisfied.

- Property 8 of Definition 20 is satisfied because, for all $x \in \mathbf{S}$, its predecessor cannot be blocked (because then x would be blocked and hence x would not be in $\mathbf{S}$). If $(\geq\ n\ R) \in \mathcal{L}(x)$, its R-successors cannot be blocked because $\neg\,\mathsf{Trans}(R)$ (if R were transitive, then number restrictions on x's R-successors would not be allowed). Thus all R-neighbours of x in $\mathbf{T}'$ are in $\mathbf{S}$. This together with the fact that $\mathbf{T}'$ satisfies $(\star)$ implies that Property 8 is satisfied.

- Suppose Property 9 of Definition 20 were not satisfied; that is, for some $x \in \mathbf{S}$ with $(\leq\ n\ R) \in \mathcal{L}(x)$ we have $|\{y \in \mathbf{S} \mid \langle x,y\rangle \in \mathcal{E}(R)\}| > n$. Now, each R-successor of x is an R-neighbour of x in $\mathbf{T}'$. If, on the contrary, we assume that there exists a $y \in \mathbf{S}$ such that $\langle x,y\rangle \in \mathcal{E}(R)$ and y is not an R-neighbour of x in $\mathbf{T}'$, then this implies the existence of a z such that either:

  - $\mathcal{L}(\langle x,z\rangle) = R$ and y blocks z, which contradicts the definition of blocking because R would need to be transitive;
  - $\mathcal{L}(\langle y,z\rangle) = \mathsf{Inv}(R)$ and x blocks z, which yields the same contradiction of the definition of blocking.

  Hence, if Property 9 were not satisfied, then $\mathbf{T}'$ would not be complete, would not satisfy condition $(\star)$, or would contain a clash. As this would contradict the initial assumption we can infer that Property 9 is satisfied. ∎

Lemma 26 If D has a tableau, then the expansion rules can be applied in such a way that the tableaux algorithm yields a complete and clash-free completion tree for D.

Proof: While the expansion-rules for $\mathcal{ALCI}_{R^+}$ contained only a single non-deterministic rule, we now have two such rules, namely the $\sqcup$-rule and the $\leq$-rule. We have to guide the application of both rules. We use the same method as was used in the proof of Lemma 7.

Let $T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ be a tableau for D. Using T, we trigger the application of the expansion rules such that they yield a completion tree $\mathbf{T}$ that is both complete and clash-free. We start with $\mathbf{T}$ consisting of a single node x_0, the root, with $\mathcal{B}(x_0) = \mathcal{L}(x_0) = \{D\}$.

T is a tableau, hence there is some $s_0 \in \mathbf{S}$ with $D \in \mathcal{L}(s_0)$. When applying the expansion rules to $\mathbf{T}$, the application of the non-deterministic $\sqcup$- and $\leq$-rules is driven by the labelling of T. To this purpose we again define a mapping π which maps the nodes of $\mathbf{T}$ to elements of $\mathbf{S}$, and we steer the application of these rules such that $\mathcal{L}(x) \subseteq \mathcal{L}(\pi(x))$ holds for all nodes x of the completion tree.

Exactly as in the proof of Lemma 7, we define π inductively as follows:

- $\pi(x_0) = s_0$.
- If $\pi(x_i) = s_i$ is already defined and a successor y of x_i is generated for $\exists R.C \in \mathcal{L}(x_i)$, then $\pi(y) = t$ for some $t \in \mathbf{S}$ with $C \in \mathcal{L}(t)$ and $\langle s_i, t\rangle \in \mathcal{E}(R)$.

In addition to the $\sqcup'$-rule already shown in Figure 5, we also modify the $\leq$-rule as shown in Figure 9.

```
≤'-rule: if 1. $(\leq\ n\ S) \in \mathcal{L}(x)$ and
            2. $x$ has $m > n$ $S$-neighbours $y_1, \ldots y_m$
         then pick two nodes $y_{i_1}$ and $y_{i_2}$ such that $y_{i_1}$ is not a predecessor of $x$,
              and $\pi(y_{i_1}) = \pi(y_{i_2})$,
              set $\mathcal{L}(y_{i_2}) = \mathcal{L}(y_{i_2}) \cup \mathcal{L}(y_{i_1})$ and delete $y_{i_1}$ from the tree.
```

Figure 9: The $\leq'$-rule

Again, it is easy to see that, if a complete and clash-free completion tree $\mathbf{T}$ was generated using the modified rule, there is also a sequence of applications of the original rules which yield the same tree. This implies the soundness and termination of the modified algorithm.

The same arguments as in the proof of Lemma 7 suffice to show that $\mathcal{L}(x) \subseteq \mathcal{L}(\pi(x))$ is preserved by the application of the modified expansion rules. The only new case is the $\leq'$-rule (because the $\geq$-rule adds a node with an empty label, which is the trivial subset of all other labels):

Consider a node with $(\leq\ n\ R) \in \mathcal{L}(x)$, which implies by induction $(\leq\ n\ R) \in \mathcal{L}(\pi(x))$. Since $\pi(x)$ is a node in a tableau, it has at most n R-successors. Furthermore, for each R-neighbour y of x we have $\langle \pi(x), \pi(y)\rangle \in \mathcal{E}(R)$—hence, if the $\leq'$-rule is applicable, there are two nodes y_{i_1} and y_{i_2} such that $\pi(y_{i_1}) =$

$\pi(y_{i_2})$. By induction $\mathcal{L}(y_{i_1}) \subseteq \mathcal{L}(\pi(y_{i_1}))$ and $\mathcal{L}(y_{i_2}) \subseteq \mathcal{L}(\pi(y_{i_2}))$, and this implies $\mathcal{L}(y_{i_1}) \cup \mathcal{L}(y_{i_2}) \subseteq \mathcal{L}(\pi(y_{i_1}))$.

The tableau construction therefore finishes with a complete tree, and since we have $\mathcal{L}(x) \subseteq \mathcal{L}(\pi(x))$ for all nodes x in the tree, it is also clash-free. ∎

Theorem 27 The tableaux algorithm is a decision procedure for the satisfiability and subsumption of $\mathcal{ALCNI}_{R^+}$-concepts.

Proof: This is a direct consequence of Lemma 23, Lemma 25, and Lemma 26.

∎

5.3 A PSPACE-algorithm

Once again, the expansion rules do not immediately induce an algorithm which consumes only polynomial space. In particular, the $\leq$-rule requires several successors of a node to be present at the same time, and this cannot be combined with the constraint that we only want to have a single path of the completion tree in the memory at one time.

What we can do instead is to postpone the generation of successors by the $\exists$-rule until the moment when we know all number restrictions, and then to guess a "distribution" of the concepts E_i required by concepts of the form $\exists R.E_i$ to the maximum number of R-successors allowed by $(\leq\ n\ R)$ number restrictions.

More precisely, we:

- replace the $\exists$-rule by the two rules shown in Figure 10,
- remove the $\leq$-rule from the set of expansion rules, and
- use the $\forall'$- and the $\forall'_+$-rules from Figure 6.

Summing up, the so-called *modified $\mathcal{ALCNI}_{R^+}$-algorithm* uses the rules from the set $\{\sqcap, \sqcup, \forall', \forall'_+, \exists'_1, \exists'_2, \geq\}$.

To make sure that the two $\exists'$-rules do not lead to an incorrect algorithm, we delay the generation of R-successors of a node x until all number restrictions of the form $(\leq\ n\ R)$ which x has to satisfy are known. This can be achieved by postponing the generation of successors of x until no *non-generating* rules are applicable to x or any of its ancestors, where the $\exists'$-rules and the $\geq$-rule are the only generating rules (because they generate new nodes in **T**): all other rules are called non-generating rules. Note that this gives the $\exists'$-rules precedence over the $\geq$-rule since this rule is postponed until no other rule can be applied to a node or its ancestors.

An *n-partition of a set M* is a finite sequence of sets $M_1, \ldots, M_n$ such that $M_i \neq \emptyset$, $M_i \cap M_j = \emptyset$ for $i \neq j$, and $M = \bigcup_i^n M_i$.

```
$\exists'_1$-rule: if 1. $\exists R.C \in \mathcal{L}(x)$ and $x$ has no $R$-predecessor, and
                  2. there is no $R$-successor $y$ of $x$ such that $C \in \mathcal{L}(y)$, and
                  3. no non-generating rule is applicable to $x$ or any of its ancestors,
              then delete all $R$-successors of $x$;
                  let $n_{\text{min}}$ be the minimal $n$ such that $(\leq\ n\ R) \in \mathcal{L}(x)$;
                  let $M = \{E \mid \exists R.E \in \mathcal{L}(x)\ \}$
                  guess $m \in \{1, \dots, n_{\text{min}}\}$ and a partition $M_1, \dots, M_m$ of $M$
                  create $m$ successors $y_1, \dots, y_m$ and set $\mathcal{L}(y_i) = \mathcal{B}(y_i) = M_i$
                  set $\mathcal{L}(\langle x, y_i \rangle) = R)$

$\exists'_2$-rule: if 1. $\exists R.C \in \mathcal{L}(x)$ and $x$ has an $R$-predecessor $x'$, and
                  2. there is no $R$-neighbour $y$ of $x$ such that $C \in \mathcal{L}(y)$, and
                  3. no non-generating rule is applicable to $x$ or any of its ancestors,
              then delete all $R$-successors of $x$;
                  let $n_{\text{min}}$ be the minimal $n$ such that $(\leq\ n\ R) \in \mathcal{L}(x)$;
                  let $M = \{E \mid \exists R.E \in \mathcal{L}(x)\ \}$
                  guess an $m \in \{1, \dots, n_{\text{min}}\}$ and a partition $M_1, \dots, M_m$ of $M$
                  if $M_1 \not\subseteq \mathcal{L}(x')$ then
                        $\mathcal{L}(x') \longrightarrow \mathcal{L}(x') \cup M_1$ and delete all descendants of $x'$
                  else create $m - 1$ successors $y_2, \dots, y_m$
                        and set $\mathcal{L}(y_i) = \mathcal{B}(y_i) = M_i$, set $\mathcal{L}(\langle x, y_i \rangle) = R$.
```

Figure 10: The $\exists'$-rules

Guessing a distribution of concepts E_i for $(\exists R.E_i)$ concepts is an inevitable non-determinism which was already present, although perhaps less obvious, in the former $\leq$-rule. The extra complexity of guessing an $m \leq n_{\text{min}}$ makes the completeness proof easier for the modified algorithm—without guessing this m, in the case where the tableau which triggers the application of the rules has less than n_{min} R-successors we would have to argue that another tableau for D exists having n_{min} R-successors.

5.3.1 Soundness and completeness of the modified algorithm

Lemma 28 For each $\mathcal{ALCNI}_{R^+}$-concept D, the modified $\mathcal{ALCNI}_{R^+}$ algorithm terminates.

Proof: Let $m = |\mathit{sub}(D)|$. Again, the algorithm has the following properties:

1. Concepts are never removed from the labels of the nodes.
2. The depth of the tree is limited by m^4.
3. The out-degree of the tree is limited by m.
4. Whenever nodes are deleted, either the label of the predecessor of the deleted nodes grows, or the deleted nodes are replaced by "fresh" ones, in which case the rule which caused the deletion no longer applies. ∎

Lemma 29 For each $\mathcal{ALCNI}_{R^+}$-concept D, if the modified $\mathcal{ALCNI}_{R^+}$ algorithm generates a complete and clash-free tree, then D has a tableau.

Proof: This lemma is a (nearly) immediate consequence of Lemma 25. The deletion of nodes does not affect the soundness of the algorithm. Moreover, each transformation of the completion tree caused by an application of one of the $\exists'$-rules can be imitated by successive applications of the $\exists$- and the $\leq$-rules in the original algorithm. This is possible since none of the successors generated by the $\exists'$-rules has an empty label. Hence each complete and clash-free completion tree generated by the modified $\mathcal{ALCNI}_{R^+}$ algorithm could also be generated by the original algorithm, which implies the soundness of the modified $\mathcal{ALCNI}_{R^+}$ algorithm. ∎

Lemma 30 Let D be an $\mathcal{ALCNI}_{R^+}$-concept: If D has a tableau, then the expansion rules can be applied in such a way that the modified $\mathcal{ALCNI}_{R^+}$ algorithm yields a complete and clash-free completion tree for D.

Given a tableau for D, we can guide the application of the $\exists'$-rules in the same way as before: we can use the tableau to trigger the guessing of the "correct" disjuncts, the "correct" number of successors and the "correct" partitioning. The proof is quite similar to the one of Lemma 26 and is therefore omitted.

Theorem 31 The modified tableaux algorithm is a decision procedure for the satisfiability and subsumption of $\mathcal{ALCNI}_{R^+}$-concepts.

Proof: This theorem is an immediate consequence of Lemma 28, Lemma 29, and Lemma 30. ■

5.3.2 An efficient implementation of the modified algorithm.

Theorem 32 $\mathrm{SAT}(\mathcal{ALCNI}_{R^+}) \in \mathrm{PS}\textsc{pace}$.

Proof: Let D be an $\mathcal{ALCNI}_{R^+}$ concept and $m = |sub(D)|$. We know that a path in a completion tree for D has a size of at most m^4. All that remains to show is that we can check for the existence of a complete and clash-free tree for D in a depth-first manner, and that we only need polynomial storage for each node in a path.

The only new features in the $\mathcal{ALCNI}_{R^+}$ algorithm as compared to the $\mathcal{ALCI}_{R^+}$ algorithm which might cause trouble are the $\exists'$-rules. However, if we store the information about the number of successors already generated and the concepts which have already been distributed to these successors, then we can re-use the space allocated for the testing of the successors. Hence we need an additional m bits to store the concepts which still have to be distributed to successors, as well as $|R_D|$ counters to keep track of the numbers of successors already generated. The numbers to be stored in these counters will always be limited by the maximum number appearing in a number restriction in $sub(D)$. Hence the algorithm can be implemented using only polynomial space. Of course, this is again a non-deterministic algorithm, but due to Savitch's theorem it is enough to establish the upper complexity bound. ■

Corollary 33 SAT($\mathcal{ALCNI}_{R^+}$) is PSPACE-complete.

Proof: Just as for $\mathcal{ALCI}_{R^+}$, SAT($\mathcal{ALCNI}_{R^+}$) is also PSPACE-hard, and it is also in PSPACE as the previous theorem showed. ■

Unlike the $\mathcal{ALCI}_{R^+}$-algorithm, which probably can be implemented in an efficient manner, the modified $\mathcal{ALCNI}_{R^+}$ algorithm seems to forbid such an implementation. This is due to the large rôle of non-determinism in the application of the $\exists'_1$- and $\exists'_2$-rules. The number p of distinct m-partitions in the $\exists'_1$-rule is given by the formula:

$$p = \frac{m^{|M/R|}}{m!}.$$

Since $|M/R|$ is linear in the size of D and m might be small, this allows an exponential number of possibilities for the $\exists'_1$-rule. A similar formula holds for the $\exists'_2$-rule.

Acknowledgements

Thanks to Nicolette Bonnette for carefully reading an earlier version of this paper and pointing out a subtle error in the $\mathcal{ALCI}_{R^+}$-algorithm.

References

[Baa90] F. Baader. A formal definition of the expressive power of knowledge representation languages. Research Report RR-90-05, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI), April 1990.

[BBH96] F. Baader, M. Buchheit, and B. Hollunder. Cardinality restrictions on concepts. *Artificial Intelligence*, 88(1–2):195–213, 1996.

[BDS93] M. Buchheit, F. M. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993.

[GMar] Giuseppe De Giacomo and Fabio Massacci. Combining deduction and model checking into tableaux and algorithms for converse-pdl. *Information and Computation*, to appear.

[HM92] J. Y. Halpern and Y. Moses. A guide to completeness and complexity for model logics of knowledge and belief. *Artificial Intelligence*, 54(3):319–379, April 1992.

[HN90] B. Hollunder and W. Nutt. Subsumption algorithms for concept languages. In *Proceedings of the 9th European Conference on Artificial Intelligence (ECAI'90)*, pages 348–353. John Wiley & Sons Ltd., 1990.

[HS98a] I. Horrocks and U. Sattler. A description logic with transitive and converse roles and role hierarchies. Technical Report 98-05, LuFg Theoretical Computer Science, RWTH Aachen, 1998.

[HS98b] I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. 1998. Submitted for the JLC special issue on Description Logics.

[HS98c] I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. In E. Franconi, G. De Giacomo, R. M. MacGregor, W. Nutt, C. A. Welty, and F. Sebastiani, editors, *Collected Papers from the International Description Logics Workshop (DL'98)*, pages 72–81, 1998.

[Sat96] U. Sattler. A concept language extended with different kinds of transitive roles. In G. Görz and S. Hölldobler, editors, *20. Deutsche Jahrestagung für Künstliche Intelligenz*, number 1137 in Lecture Notes in Artificial Intelligence, pages 333–345. Springer Verlag, 1996.

[Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, April 1970.

[SSS91] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.

[Spa93] E. Spaan. The complexity of propositional tense logics. In M. de Rijke, editor, *Diamonds and Defaults*, pages 287–307. Kluwer Academic Publishers, Dordrecht, 1993.