A Description Logic with Transitive and Inverse Roles and Role Hierarchies

Ian Horrocks[*] and Ulrike Sattler[†]
Department of Computer Science, University of Manchester
LuFG Theoretical Computer Science, RWTH Aachen

Journal of Logic and Computation, 9(3):385-410, 1999.

[*] Part of this work was carried out while being a guest at IRST, Trento.
[†] This work was supported by the Esprit Project 22469 – DWQ.

Abstract

The combination of transitive and inverse roles is important in a range of applications, and is crucial for the adequate representation of aggregated objects, allowing the simultaneous description of parts by means of the whole to which they belong and of wholes by means of their constituent parts. In this paper we present tableaux algorithms for deciding concept satisfiability and subsumption in Description Logics that extend $\mathcal{ALC}$ with both transitive and inverse roles, a role hierarchy, and functional restrictions. In contrast to earlier algorithms for similar logics, those presented here are well-suited for implementation purposes: using transitive roles and role hierarchies in place of the transitive closure of roles enables sophisticated blocking techniques to be used in place of the cut rule, a rule whose high degree of non-determinism strongly discourages its use in an implementation. As well as promising superior computational behaviour, this new approach is shown to be sufficiently powerful to allow subsumption and satisfiability with respect to a (possibly cyclic) knowledge base to be reduced to concept subsumption and satisfiability, and to support reasoning in a Description Logic that no longer has the finite model property.

1 Motivation

As argued in [19, 26], transitive roles play an important rôle in the adequate representation of aggregated objects: they allow these objects to be described by referring to their parts without specifying a level of decomposition. In [17], the Description Logic (DL) $\mathcal{ALCH}_{R^+}$ is presented, which extends $\mathcal{ALC}$ [29] with transitive roles and a role hierarchy. It is argued in [27] that $\mathcal{ALCH}_{R^+}$ is well-suited to the representation of aggregated objects in applications that require various part-whole relations to be distinguished, some of which are transitive. For example,[1] a knowledge base describing nuclear reactors could contain the following entries

$$\begin{array}{ll} \texttt{is\_component\_of} & \sqsubseteq \texttt{is\_part\_of}, \\ \texttt{Control\_rod} & \sqsubseteq \texttt{Device} \sqcap \exists \texttt{is\_component\_of}.\texttt{Reactor\_core}, \\ \texttt{Reactor\_core} & \sqsubseteq \texttt{Device} \sqcap \exists \texttt{is\_component\_of}.\texttt{Nuclear\_reactor}, \end{array}$$

where $\sqsubseteq$ is a subsumption (implication) relationship and `is_part_of` is assumed to be a transitive role name. It can be inferred from this knowledge base that `Control_rod` is subsumed by $\exists \texttt{is\_part\_of}.\texttt{Nuclear\_reactor}$.

$\mathcal{ALCH}_{R^+}$ does not, however, allow the simultaneous description of components by means of the devices to which they belong, and devices by means of their constituent components: one or other is possible, but not both. To overcome this limitation we present $\mathcal{ALCHI}_{R^+}$, a DL that extends $\mathcal{ALCH}_{R^+}$ with inverse (converse) roles, allowing, for example, the use of `has_part` as well as `is_part_of`.[2] Using $\mathcal{ALCHI}_{R^+}$, we can describe a dangerous nuclear reactor by

$$\texttt{Nuclear\_reactor} \sqcap \exists \texttt{has\_part}.\texttt{Faulty} \sqsubseteq \texttt{Dangerous\_nuclear\_reactor},$$

and then recognise that

$$\texttt{Control\_rod} \sqcap \texttt{Faulty}$$

[1] This example is purely for didactic purposes and is not intended as a contribution to the philosophical debate on the semantics of part-whole relationships.

[2] Note that `has_part` is taken to be the inverse of `is_part_of`.

is subsumed by

$$\exists \texttt{is\_part\_of}.\texttt{Dangerous\_nuclear\_reactor}.$$

Moreover, $\mathcal{ALCHI}_{R^+}$ can be further extended with *functional restrictions* to give $\mathcal{ALCHFI}_{R^+}$. Functional restrictions are useful in general because they provide a weak form of number restrictions. For example, functional restrictions can be used in an axiom

$$\texttt{Nuclear\_reactor} \sqsubseteq \exists \texttt{controlled\_by}.\texttt{Control\_unit} \sqcap (\leqslant 1\ \texttt{controlled\_by})$$

in order to capture the knowledge that a nuclear reactor is controlled by at most one control unit. An interesting feature of $\mathcal{ALCHFI}_{R^+}$ is that it no longer has the finite model property: there can be $\mathcal{ALCHFI}_{R^+}$ concepts that are satisfiable but for which there exists no finite model (see Section 2.2).

Like $\mathcal{ALCH}_{R^+}$, $\mathcal{ALCHI}_{R^+}$ and $\mathcal{ALCHFI}_{R^+}$ also allow for the internalisation of *general concept inclusion axioms* using a *universal role*: a transitive role that subsumes all other roles in the terminology [17]. This technique allows subsumption and satisfiability with respect to (possibly cyclic) general concept inclusion axioms to be reduced to concept satisfiability and subsumption.

It could be argued that, instead of defining yet another DL, one could make use of the results presented in [9] and use $\mathcal{ALC}$ extended with role expressions which include transitive closure and inverse operators. The reason for not proceeding like this is the fact that transitive roles can be implemented more efficiently than the transitive closure of roles [17], although they lead to the same complexity class (ExpTime-complete) when added, together with role hierarchies, to $\mathcal{ALC}$. Using transitive roles and a role hierarchy instead of role expressions does lead to some loss of expressive power as it is no longer possible to completely capture the relationship between a role and its transitive closure. If, for example, `is_part_of` were taken to be the transitive closure of `is_component_of`, then we would have

$$\exists \texttt{is\_component\_of}.\top \doteq \exists \texttt{is\_part\_of}.\top,$$

where $\doteq$ is an equivalence (if and only if) relationship; when `is_part_of` is a transitive superrole of `is_component_of`, we only have

$$\exists \texttt{is\_component\_of}.\top \sqsubseteq \exists \texttt{is\_part\_of}.\top.$$

However, we believe that it is hard to find applications for which the *smallest* transitive superrole of a given role is crucial and that could not, instead, use a less specific transitive superrole.

Furthermore, it is still an open question whether the transitive closure of roles together with inverse roles necessitates the use of the *cut rule* [10], a rule that, due to its high degree of non-determinism, leads to an algorithm with very bad computational behaviour. This problem would be further exacerbated by embedding functional restrictions in such a logic (as described in [8]), because the embedding generates large numbers of general concept inclusion axioms—another source of non-determinism.

We will present algorithms for both $\mathcal{ALCHI}_{R^+}$ and $\mathcal{ALCHFI}_{R^+}$ that decide satisfiability and subsumption and that do not include the cut rule but instead employ new *blocking* techniques in order to guarantee both correctness and termination. Experiences with an implementation of $\mathcal{ALCH}_{R^+}$ in the FaCT system [18] suggest that these algorithms should behave well in practice.

2 Blocking

The algorithms that we will present use the tableaux method [16], 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, so easily 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 [26]. 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 [1, 4]. 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 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.[3] The validity of the cyclical model is an easy consequence of the fact that the concepts 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 here is the introduction of a *dynamic blocking* strategy. Besides using label equality instead of subset, this strategy allows blocks to be established, broken, and re-established.

[3] For logics with a transitive closure operator it is necessary to check the validity of the cyclical model created by blocking [1], but for logics that only support transitive roles the cyclical model is always valid [26].

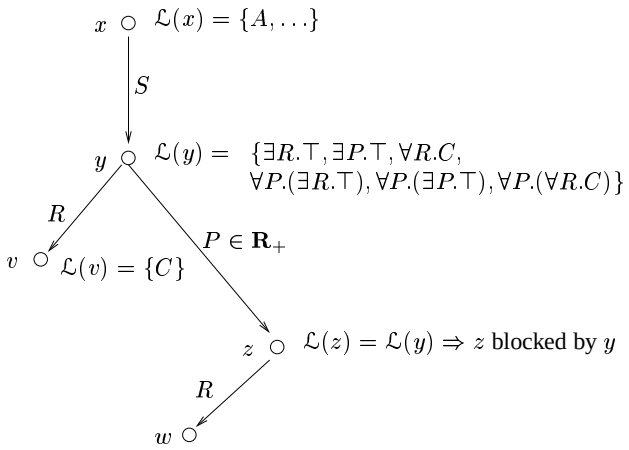

Figure 1: A tableau where dynamic blocking is crucial

With inverse roles the blocking condition must be equality of node labels 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.

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 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 1, 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 could effect other parts on the tree. Again, let us consider the example in Figure 1. 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 Pair-wise Blocking

Further extending the logic with 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) and a role hierarchy (subsumption relationships between roles) 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 superrole R. Attempting to terminate the sequence in a cycle causes the whole sequence to collapse into a single node due the functional restrictions $(\leqslant 1\ F)$, and this results in a contradiction as both C and $\neg C$ will be in the 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 is introduced, and soundness is proved by demonstrating that a blocked tree always has a corresponding infinite model.[4]

The new blocking strategy generates a tree where an infinite tableau is defined by recursively replacing the blocked node with a copy of the tree rooted at the blocking node. To be certain that this transplanted tree is still valid in its new location, blocks are established between pairs of nodes connected by the same role: a node y is blocked by a node x when their labels are equal, the labels of their predecessors y' and x' are equal, and the edges connecting x' to x and y' to y are labelled with the same role names. Note the similarity between this condition and that imposed by the combination of the blocking condition and cut rule in *converse*-PDL [11].

Figure 2 shows how pair-wise blocking is crucial in order to ensure that the algorithm discovers the unsatisfiability of the concept

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

where $F \sqsubseteq R$ and D represents the concept

$$C \sqcap (\leqslant 1\ F) \sqcap \exists F.\neg C.$$

Using dynamic blocking, z would be blocked by y. The resulting tree cannot represent a cyclical model in which y is related to itself by an F^{-} role as this would conflict with $(\leqslant 1\ F) \in \mathcal{L}(y)$. The tree must therefore represent the infinite model generated by recursively replacing each occurrence of z with a copy of the tree rooted at y. However, this is not a valid model as when z is substituted by a copy of y, $\exists F.\neg C \in \mathcal{L}(y)$, which was satisfied by $\neg C \in \mathcal{L}(x)$, is no longer satisfied in its new location.

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

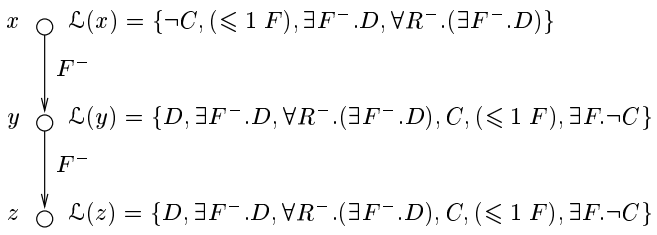

Figure 2: A tableau where pair-wise blocking is crucial

When pair-wise blocking is used, z is no longer blocked by y as the labels of their predecessors (y and x respectively) are not equal, and the algorithm continues to expand $\mathfrak{L}(z)$. The expansion of $\exists F.\neg C \in \mathfrak{L}(z)$ calls for the existence of a node whose label includes $\neg C$ and that is connected to z by an F labelled edge. Because of $(\leqslant 1\ F) \in \mathfrak{L}(z)$ this node must be y, and this results in a contradiction as both C and $\neg C$ will be in $\mathfrak{L}(y)$.

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

For ease of understanding, we start by introducing the Description Logic $\mathcal{ALCI}_{R^+}$, which is the extension of the well-known DL $\mathcal{ALC}$ [29] with *transitive 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.

In the next section, we describe a tableaux algorithm for testing the satisfiability of $\mathcal{ALCI}_{R^+}$ concepts and present a proof of its soundness and completeness. The extension of $\mathcal{ALCI}_{R^+}$ with role hierarchies, $\mathcal{ALCHI}_{R^+}$, together with the extended tableaux algorithm and corresponding proofs is then described in Section 5. Finally, the extension of $\mathcal{ALCHI}_{R^+}$ with functional restrictions together with the further extended tableaux algorithm and corresponding proofs is presented in Section 6.

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.

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 and roles R, S, the properties in Figure 3 are satisfied.

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

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

Figure 3: Semantics of $\mathcal{ALCI}_{R^+}$-concepts

$C \sqsubseteq D$) 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 $\mathsf{Inv}(R) \in \mathbf{R}_+$.

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

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 a set 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, where 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 [16].

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, $\mathcal{L} : \mathbf{S} \rightarrow 2^{sub(D)}$ maps each individual to a set of concepts which is a subset of $sub(D)$, $\mathcal{E} : \mathbf{R}_D \rightarrow 2^{\mathbf{S}\times\mathbf{S}}$ 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. 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:

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

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 $\langle s,s_1\rangle$, $\langle s_1,s_2\rangle$, $\ldots$, $\langle s_n,t\rangle \in \mathcal{E}(S)$ and $\mathsf{Trans}(S)$. 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 $t \in C^{\mathcal{I}}$, 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:

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

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. Assume that T does not satisfy Property 6 in Definition 2. There must then be some $d \in \mathbf{S}$ with $\langle d,e\rangle \in \mathcal{E}(R)$, $\mathsf{Trans}(R)$, $(\forall R.C) \in \mathcal{L}(d)$, and $(\forall R.C) \notin \mathcal{L}(e)$. The definition of T implies that $d \in (\forall R.C)^{\mathcal{I}}$, $\langle d,e\rangle \in R^{\mathcal{I}}$, $\mathsf{Trans}(R)$, and $e \notin (\forall R.C)^{\mathcal{I}}$. However, this can only be the case if there is some f such that $\langle e,f\rangle \in R^{\mathcal{I}}$ and $f \notin C^{\mathcal{I}}$. The transitivity of R implies that $\langle d,f\rangle \in R^{\mathcal{I}}$ and, because $d \in (\forall R.C)^{\mathcal{I}}$, we also have $f \in C$, contradicting $f \notin C$. Therefore T does satisfy Property 6 in Definition 2.

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

4.1 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 where each node x of the tree is labelled with a set $\mathcal{L}(x) \subseteq \mathit{sub}(D)$ and each edge $\langle x,y\rangle$ is labelled $\mathcal{L}(\langle x,y\rangle) = R$ for some (possibly inverse) role R occurring in $\mathit{sub}(D)$. 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 either by extending $\mathcal{L}(x)$ 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 $\mathbf{T}$ and a concept C, $\{C, \neg C\} \subseteq \mathcal{L}(x)$.

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; *ancestor* is the transitive closure of *predecessor*.

A node y is called an *R-neighbour* of a node x if either y is a successor of x and $\mathcal{L}(\langle x,y\rangle) = R$ or y is a predecessor of x and $\mathcal{L}(\langle y,x\rangle) = \mathsf{Inv}(R)$.

A node x is *blocked* if for some ancestor y, y is blocked or $\mathcal{L}(x) = \mathcal{L}(y)$. A blocked node x is *indirectly blocked* if its predecessor is blocked, otherwise it is *directly blocked*. If x is directly blocked, it has a unique ancestor y such that $\mathcal{L}(x) = \mathcal{L}(y)$: if there existed another

$\sqcap$-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 $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C\}$ for some $C \in \{C_1, C_2\}$

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

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

ancestor z such that $\mathcal{L}(x) = \mathcal{L}(z)$ then either y or z must be blocked. If x is directly blocked and y is the unique ancestor such that $\mathcal{L}(x) = \mathcal{L}(y)$, we will say that y *blocks* x.

The algorithm initialises a tree $\mathbf{T}$ to contain a single node x_0, called the *root* node, with $\mathcal{L}(x_0) = \{D\}$, 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*".

4.2 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.

Lemma 4 For each $\mathcal{ALCI}_{R^+}$-concept D, the tableaux algorithm terminates.

PROOF. Let $m = |sub(D)|$. 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 $sub(D)$ 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 $sub(D)$. If a path p is of length at least 2^m, then there are 2 nodes x, y on p, with $\mathcal{L}(x) = \mathcal{L}(y)$, and blocking occurs. Since

a path on which nodes are blocked cannot become longer, paths are of length at most 2^m. ∎

In [22], an optimised version of this tableaux algorithm is presented, namely one that generates completion trees whose depth is polymially bounded by the size of the input concept.

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

Lemma 5 (Soundness) If the expansion rules can be applied to an $\mathcal{ALCI}_{R^+}$-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:

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

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 $\langle x, y\rangle \in \mathcal{E}(R)$ then either:
   (a) y is an R-neighbour of x,
   (b) $\mathcal{L}(\langle x, z\rangle) = R$, y blocks z, and $\mathcal{L}(y) = \mathcal{L}(z)$, or
   (c) $\mathcal{L}(\langle y, z\rangle) = \mathsf{Inv}(R)$, x blocks z, and $\mathcal{L}(x) = \mathcal{L}(z)$.

   In all 3 cases, the $\forall$-rule ensures that $C \in \mathcal{L}(y)$.
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) a predecessor y such that $\mathcal{L}(\langle y, x\rangle) = \mathsf{Inv}(R)$ and $C \in \mathcal{L}(y)$. Because y is a predecessor of x it cannot be blocked, so $y \in \mathbf{S}$ and $\langle x, y\rangle \in \mathcal{E}(R)$.
   (b) a successor y such that $\mathcal{L}(\langle x, y\rangle) = R$ and $C \in \mathcal{L}(y)$. 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 $\mathcal{L}(z) = \mathcal{L}(y)$. 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 $\forall R.C \in \mathcal{L}(x)$, $\langle x, y\rangle \in \mathcal{E}(R)$, and $\mathsf{Trans}(R)$, then either:

$\sqcup'$-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 $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C\}$ for some $C \in \{C_1, C_2\} \cap \mathcal{L}(\pi(x))$

Figure 5: The $\sqcup'$-rule

(a) y is an R-neighbour of x,

(b) $\mathcal{L}(\langle x, z\rangle) = R$, y blocks z, and $\mathcal{L}(y) = \mathcal{L}(z)$, or

(c) $\mathcal{L}(\langle y, z\rangle) = \mathsf{Inv}(R)$, x blocks z, hence $\mathcal{L}(x) = \mathcal{L}(z)$ and $\forall R.C \in \mathcal{L}(z)$.

In all 3 cases, the $\forall_+$-rule ensures that $\forall R.C \in \mathcal{L}(y)$.

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.

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

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

In all 3 cases, $\langle y, x\rangle \in \mathcal{E}(\mathsf{Inv}(R))$. ∎

Lemma 6 (Completeness) 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 $\mathcal{L}(x_0) = \{D\}$.

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:

- $\pi(x_0) = s_0$.
- 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 *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 5 and Lemma 4 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 $C_1 \sqcap C_2 \in \mathcal{L}(x)$, 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 $C_1 \sqcup C_2 \in \mathcal{L}(x)$, then $C \in \{C_1, C_2\}$ is in $\mathcal{L}(\pi(x))$, and C 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 $\mathcal{L}(y) \subseteq \mathcal{L}(\pi(y))$.

If the $\forall_+$-rule can be applied to x in $\mathbf{T}$ with $C = \forall R.C_1 \in \mathcal{L}(x)$, $\mathsf{Trans}(R)$, 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 7 The tableaux algorithm is a decision procedure for the satisfiability and subsumption of $\mathcal{ALCI}_{R^+}$-concepts.

Theorem 7 is an immediate consequence of the Lemmata 3, 4, 5 and 6. Moreover, since $\mathcal{ALCI}_{R^+}$ is closed under negation, subsumption $C \sqsubseteq D$ can be reduced to the unsatisfiability of $C \sqcap \neg D$.

5 $\mathcal{ALCI}_{R^+}$ Extended by Role Hierarchies

We will now extend the tableaux algorithm presented in Section 4.1 to deal with *role hierarchies* in a similar way to the algorithm for $\mathcal{ALCH}_{R^+}$ presented in [17]. $\mathcal{ALCHI}_{R^+}$ extends $\mathcal{ALCI}_{R^+}$ by allowing, additionally, for inclusion axioms on roles. These axioms can involve transitive as well as non-transitive roles, and inverse roles as well as role names. For example, to express that a role R is symmetric, we add the two axioms $R \sqsubseteq R^-$ and $R^- \sqsubseteq R$.

Definition 8 A *role inclusion axiom* is of the form

$$R \sqsubseteq S,$$

for two (possibly inverse) roles R and S. For a set of role inclusion axioms $\mathcal{R}$,

$$\mathcal{R}^+ := (\mathcal{R} \cup \{\mathsf{Inv}(R) \sqsubseteq \mathsf{Inv}(S) \mid R \sqsubseteq S \in \mathcal{R}\}, \sqsubseteq^*)$$

is called a *role hierarchy*, where $\sqsubseteq^*$ is the transitive-reflexive closure of $\sqsubseteq$ over $\mathcal{R} \cup \{\mathsf{Inv}(R) \sqsubseteq \mathsf{Inv}(S) \mid R \sqsubseteq S \in \mathcal{R}\}$.

$\mathcal{ALCHI}_{R^+}$ is the extension of $\mathcal{ALCI}_{R^+}$ obtained by allowing, additionally, for a role hierarchy $\mathcal{R}^+$.

As well as being correct for $\mathcal{ALCI}_{R^+}$ concepts, an $\mathcal{ALCHI}_{R^+}$ interpretation has to satisfy, for all roles R, S with $R \sqsubseteq^* S$, the additional condition

$$\langle x, y \rangle \in R^{\mathcal{I}} \text{ implies } \langle x, y \rangle \in S^{\mathcal{I}}.$$

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

Figure 6: The new $\forall_+$-rule for $\mathcal{ALCHI}_{R^+}$

The tableaux algorithm given in the preceding section can easily be modified to decide satisfiability of $\mathcal{ALCHI}_{R^+}$-concepts by extending the definitions of both R-neighbours and the $\forall_+$-rule to include the notion of role hierarchies. To prove the soundness and completeness of the extended algorithm, the definition of a tableau is also extended.

Definition 9 As well as satisfying Definition 2 (i.e., being a valid $\mathcal{ALCI}_{R^+}$ tableau), a tableau $T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ for an $\mathcal{ALCHI}_{R^+}$-concept D must also satisfy:

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 $\forall R.C \in \mathcal{L}(t)$,

8. if $\langle x, y\rangle \in \mathcal{E}(R)$ and $R \sqsubseteq^* S$, then $\langle x, y\rangle \in \mathcal{E}(S)$,

where Property 6′ extends and supersedes Property 6 from Definition 2.

5.1 Constructing an $\mathcal{ALCHI}_{R^+}$ Tableau

For the $\mathcal{ALCHI}_{R^+}$ algorithm, the $\forall_+$-rule is replaced with the $\forall'_+$-rule (see Figure 6) and the definition of R-neighbours is extended as follows:

Definition 10 Given a completion tree, a node y is called an *S-neighbour* of a node x if, for some R with $R \sqsubseteq^* S$, either y is a successor of x and $\mathcal{L}(\langle x, y\rangle) = R$ or y is a predecessor of x and $\mathcal{L}(\langle y, x\rangle) = \mathsf{Inv}(R)$.

Due to this definition and the reflexivity of $\sqsubseteq^*$, the $\forall'_+$-rule extends the $\forall_+$-rule. In the following, the tableaux algorithm resulting from these modifications will be called the *modified tableaux algorithm*.

5.2 Soundness and Completeness

To prove that the modified tableaux algorithm is indeed a decision procedure for the satisfiability of $\mathcal{ALCHI}_{R^+}$-concepts, all 4 technical lemmata used in Section 4.2 to prove this fact for the $\mathcal{ALCI}_{R^+}$ tableaux algorithm have to be re-proven for $\mathcal{ALCHI}_{R^+}$. In the following, we will restrict our attention to cases that differ from those already considered for $\mathcal{ALCI}_{R^+}$.

Lemma 11 An $\mathcal{ALCHI}_{R^+}$-concept D is satisfiable iff there exists a tableau for D.

PROOF. For the *if* direction, the construction of a model of D from a tableau for D is similar to the one presented in the proof of Lemma 3. 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 follows:

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

The interpretation of non-transitive roles is recursive in order to correctly interpret those non-transitive roles that have a transitive sub-role. From the definition of $R^{\mathcal{I}}$ and Property 8 of a tableau it follows that if $\langle x, y\rangle \in S^{\mathcal{I}}$, then either $\langle x, y\rangle \in \mathcal{E}(S)$ or there exists a path $\langle x, x_1\rangle, \langle x_1, x_2\rangle, \ldots, \langle x_n, y\rangle \in \mathcal{E}(R)$ for some R with $\mathsf{Trans}(R)$ and $R \sqsubseteq^* S$.

Property 8 of a tableau ensures that $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ holds for all roles with $R \sqsubseteq^* S$, including those cases where R is a transitive role. Again, it can be shown by induction on the structure of concepts that $\mathcal{I}$ is a correct interpretation. We restrict our attention to the only case that is different from the ones in the proof of Lemma 3. Let $E \in \mathit{sub}(D)$ with $E \in \mathcal{L}(s)$.

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

$$\langle s, s_1\rangle, \langle s_1, s_2\rangle, \ldots, \langle s_n, t\rangle \in \mathcal{E}(R)$$

for some R with $\mathsf{Trans}(R)$ and $R \sqsubseteq^* S$. Due to Property 6′ in Definition 9, $\forall R.C \in \mathcal{L}(s_i)$ for all $1 \leqslant i \leqslant n$, and we have $C \in \mathcal{L}(t)$.

In both cases, we have $t \in C^{\mathcal{I}}$.

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 is defined like the one defined in the proof of Lemma 3. 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 $\mathcal{ALCHI}_{R^+}$-concepts.
2. If $d \in (\forall S.C)^{\mathcal{I}}$ and $\langle d, e\rangle \in R^{\mathcal{I}}$ for some R with $\mathsf{Trans}(R)$ and $R \sqsubseteq^* S$, 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 $\langle d, e\rangle \in R^{\mathcal{I}}$, $\langle e, f\rangle \in R^{\mathcal{I}}$, $\mathsf{Trans}(R)$, and $R \sqsubseteq^* S$, then $\langle d, f\rangle \in R^{\mathcal{I}}$, $\langle d, f\rangle \in S^{\mathcal{I}}$, and $d \notin (\forall S.C)^{\mathcal{I}}$. T therefore satisfies Property 6′ in Definition 9.
3. Since $\mathcal{I}$ is a model of D, $\langle x, y\rangle \in R^{\mathcal{I}}$ implies $\langle x, y\rangle \in S^{\mathcal{I}}$ for all roles R, S with $R \sqsubseteq^* S$. Hence T satisfies Property 8 in Definition 9. ∎

Lemma 12 For each $\mathcal{ALCHI}_{R^+}$-concept D, the modified tableaux algorithm terminates.

The proof is identical to the one given for Lemma 4.

Lemma 13 (Soundness) If the expansion rules can be applied to an $\mathcal{ALCHI}_{R^+}$-concept D such that they yield a complete and clash-free completion tree, then D has a tableau.

PROOF. The definition of a tableau from a complete and clash-free completion tree $\mathbf{T}$, as presented in the proof of Lemma 5, has to be slightly modified. A tableau $T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ is now defined with:

$$\begin{aligned}
\mathbf{S} &= \{x \mid x \text{ is a node in } \mathbf{T}, \text{ and } x \text{ is not blocked}\}\\
\mathcal{L} &= \text{the restriction of the labelling } \mathcal{L} \text{ in } \mathbf{T} \text{ to } \mathbf{S},\\
\mathcal{E}(S) &= \{\langle x, y\rangle \in \mathbf{S} \times \mathbf{S} \mid 1.\ y \text{ is an } S\text{-neighbour of } x \quad \textit{or}\\
&\qquad\qquad 2.\ \text{There exists a role } R \text{ with } R \sqsubseteq^* S \text{ and}\\
&\qquad\qquad\quad a.\ \mathcal{L}(\langle x, z\rangle) = R \text{ and } y \text{ blocks } z \quad \textit{or}\\
&\qquad\qquad\quad b.\ \mathcal{L}(\langle y, z\rangle) = \mathsf{Inv}(R) \text{ and } x \text{ blocks } z\}
\end{aligned}$$

and, again, 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. Due to the enhanced "neighbour" relation in Definition 10, proofs of the satisfaction of Properties 1–3, 5 and 7 in Definition 2 are identical to those in the proof of Lemma 5.

3. Property 4 in Definition 2 is satisfied because, for all $x \in \mathbf{S}$, if $\forall S.C \in \mathcal{L}(x)$ and $\langle x, y\rangle \in \mathcal{E}(S)$ then either:

   (a) y is an S-neighbour of x, or

   (b) for some role R with $R \sqsubseteq^{*} S$, either

      i. $\mathcal{L}(\langle x, z\rangle) = R$, y blocks z, and $\mathcal{L}(y) = \mathcal{L}(z)$, or

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

   In all cases, the $\forall$-rule ensures $C \in \mathcal{L}(y)$.

4. Property 6' in Definition 9 is satisfied because, for all $x \in \mathbf{S}$, if $\forall S.C \in \mathcal{L}(x)$ and $\langle x, y\rangle \in \mathcal{E}(R)$ for some R with $\mathsf{Trans}(R)$ and $R \sqsubseteq^{*} S$, then either:

   (a) y is an R-neighbour of x, or

   (b) for some role R' with $R' \sqsubseteq^{*} R$, either

      i. $\mathcal{L}(\langle x, z\rangle) = R'$, y blocks z and $\mathcal{L}(y) = \mathcal{L}(z)$, or

      ii. $\mathcal{L}(\langle y, z\rangle) = \mathsf{Inv}(R')$, x blocks z and $\mathcal{L}(x) = \mathcal{L}(z)$.

   In all cases, the $\forall'_{+}$-rule ensures that $\forall R.C \in \mathcal{L}(y)$.

5. Property 8 in Definition 9 follows immediately from the definition of $\mathcal{E}$. ∎

Lemma 14 (Completeness) If an $\mathcal{ALCHI}_{R^+}$-concept 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.

The proof of Lemma 14 is identical to the one presented for Lemma 6. Again, summing up, we have the following theorem.

Theorem 15 The modified tableaux algorithm is a decision procedure for the satisfiability and subsumption of $\mathcal{ALCHI}_{R^+}$-concepts.

5.3 General Concept Inclusion Axioms

In [1, 28, 3], the *internalisation* of terminological axioms is introduced. This technique is used to reduce reasoning with respect to a (possibly cyclic) *terminology* to satisfiability of concepts. In [17], we saw how role hierarchies can be used to reduce satisfiability and subsumption with respect to a terminology to concept satisfiability and subsumption. In the presence of inverse roles, this reduction must be slightly modified.

Definition 16 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 $\mathcal{ALCHI}_{R^+}$-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 $C_i \sqsubseteq D_i \in \mathcal{T}$. 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 $\mathcal{T}$ ($C \sqsubseteq_{\mathcal{T}} D$) 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. This role U is a transitive super-role of all relevant roles and their respective inverses. Hence, for each interpretation $\mathcal{I}$, each individual t reachable via some role path from another individual s is an $U^{\mathcal{I}}$-successor of s. All general concept inclusion axioms $C_i \sqsubseteq D_i$ in $\mathcal{T}$ are propagated along all role paths using the value restriction $\forall U.\neg C \sqcup D$.

Lemma 17 Let $\mathcal{T}$ be terminology and C, D be $\mathcal{ALCHI}_{R^+}$-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 with $R \sqsubseteq U$, $\mathsf{Inv}(R) \sqsubseteq U$ for each role R that occurs in $\mathcal{T}, C$, or D. Then C is satisfiable with respect to $\mathcal{T}$ iff

$$C \sqcap C_{\mathcal{T}} \sqcap \forall U.C_{\mathcal{T}}$$

is satisfiable. D subsumes C with respect to $\mathcal{T}$ ($C \sqsubseteq_{\mathcal{T}} D$) iff

$$C \sqcap \neg D \sqcap C_{\mathcal{T}} \sqcap \forall U.C_{\mathcal{T}}$$

is unsatisfiable.

Remark: Instead of defining U as a transitive super-role of all roles and their respective inverses, one could have defined U as a transitive super-role of all roles and, additionally, a symmetric role by adding $U \sqsubseteq U^-$ and $U^- \sqsubseteq U$.

The proof of Lemma 17 is similar to the ones that can be found in [28, 1]. One point to show is that, if an $\mathcal{ALCHI}_{R^+}$-concept C is satisfiable with respect to a terminology $\mathcal{T}$, then $C_{\mathcal{T}}$ has a *connected* model, namely one whose individuals are all related to each other by some role path. This follows from the definition of the semantics of $\mathcal{ALCHI}_{R^+}$-concepts. The other point to prove is that, if y is reachable from x via a role path (possibly involving inverse roles), then $\langle x, y\rangle \in U^{\mathcal{I}}$. This is an easy consequence of the definition of U.

Decidability of satisfiability and subsumption with respect to a terminology is an immediate consequence of Lemma 17 and Theorem 15.

Theorem 18 The modified tableaux algorithm is a decision procedure for satisfiability and subsumption of $\mathcal{ALCHI}_{R^+}$-concepts with respect to terminologies.

6 Extending $\mathcal{ALCHI}_{R^+}$ by Functional Restrictions

In this section, we will present the extension of $\mathcal{ALCHI}_{R^+}$ with functional restrictions to give $\mathcal{ALCHFI}_{R^+}$. The most general way to do this is to allow, for (possibly inverse) roles R, concepts of the form $(\leqslant 1\ R)$. These concepts express local functionality, and can be used to express global functionality, by using the general concept inclusion axiom $\top \sqsubseteq (\leqslant 1\ R)$. As the logic supports general negation, it is also necessary to allow for negated functional restrictions $\neg(\leqslant 1\ R)$; in negation normal form these become restrictions of the form $(\geqslant 2\ R)$ [16].

In $\mathcal{ALCHFI}_{R^+}$, the roles that can appear in functional restrictions are limited to *simple* roles, where a role is simple if it is neither transitive nor has transitive sub-roles. Without this limitation the extension of the $\mathcal{ALCHI}_{R^+}$ tableau construction algorithm would be more difficult due to the possibility of having to collapse a chain of successors into a single node. This would be necessary if, for example, $(\leqslant 1\ S)$ is added to the label of a node x where $R \in \mathbf{R}_+$, x already has a chain of R-successors, and $R \sqsubseteq^* S$.

Definition 19 $\mathcal{ALCHFI}_{R^+}$ is the extension of $\mathcal{ALCHI}_{R^+}$ obtained by allowing, additionally, for functional restrictions: for a simple role R, $(\leqslant 1\ R)$ is also an $\mathcal{ALCHFI}_{R^+}$-concept. A role R is a *simple* role iff $R \notin \mathbf{R}_+$ and, for any $S \sqsubseteq^* R$, S is also a simple role.

An $\mathcal{ALCHFI}_{R^+}$-interpretation is an $\mathcal{ALCHI}_{R^+}$-interpretation that satisfies, additionally,

$$(\leqslant 1\ R)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \text{For all } y, z\text{: if } \langle x, y\rangle \in R^{\mathcal{I}} \text{ and } \langle x, z\rangle \in R^{\mathcal{I}}, \text{ then } y = z\},$$
$$(\geqslant 2\ R)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \text{There exist } y, z\text{: } \langle x, y\rangle \in R^{\mathcal{I}}, \langle x, z\rangle \in R^{\mathcal{I}}, \text{ and } y \neq z\}.$$

Definition 20 If D is an $\mathcal{ALCHFI}_{R^+}$-concept in NNF, then a tableau T for D is defined like in Definition 9, with the additional properties:

9. if $(\leqslant 1\ R) \in \mathcal{L}(s)$ and $\langle s, t\rangle \in \mathcal{E}(R)$ and $\langle s, t'\rangle \in \mathcal{E}(R)$, then $t = t'$, and
10. if $(\geqslant 2\ R) \in \mathcal{L}(s)$, then there are some $t, t' \in \mathbf{S}$ such that $\langle s, t\rangle \in \mathcal{E}(R)$, $\langle s, t'\rangle \in \mathcal{E}(R)$, and $t \neq t'$.

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

The reader will recall from earlier sections that $\mathcal{ALCHFI}_{R^+}$ no longer has the finite model property. In the algorithm presented here, this will be dealt with by generating (finite) completion trees and showing how they can be interpreted as infinite tableaux.

The proof is similar to the proof of Lemma 11, with the additional observations that

1. In the *if* direction, Properties 9 and 10 in Definition 20 ensure that functional restrictions are interpreted correctly. This depends on the fact that only simple roles can appear in functional restrictions, as for a simple role R, $R^{\mathcal{I}} = \mathcal{E}(R)$.
2. In the *only if* direction, the semantics of functional restrictions ensure that Properties 9 and 10 in Definition 20 are satisfied.

6.1 Constructing an $\mathcal{ALCHFI}_{R^+}$ Tableau

In this section, we show how the tableaux algorithm for $\mathcal{ALCHI}_{R^+}$ can be extended to deal with $\mathcal{ALCHFI}_{R^+}$-concepts. The following is a list of modifications that are necessary to deal with functional roles. The resulting definitions are then given in Definition 22.

1. If a node x has more R-neighbours than allowed by a functional restriction $(\leqslant 1\ R)$, we will merge these R-neighbours into a single one. Since these R-neighbours can also be neighbours with respect to some roles S, S' which are not comparable by $\sqsubseteq^*$, the merged R-neighbour is also an S- and an S'-neighbour of x. To capture this, edges will be labelled with *sets* of roles.

2. Due to the new, set-valued edge labelling, the definitions of neighbours and successors have to be adjusted; as described in Definition 22.

3. The blocking strategy from Section 4.1 is extended by using pair-wise blocking as described in Section 2.2.

4. Tableau expansion rules must be added for functional restriction concepts, and the $\exists$-rule must be amended in order to deal with set valued edge labels. The complete set of $\mathcal{ALCHF}\mathcal{I}_{R^+}$ expansion rules is given in Figure 7. For the proof of the soundness of these rules, namely the proof of Lemma 24, if $(\geqslant 2\ R) \in \mathcal{L}(x)$, then we always introduce two R-successors which can never be merged. For this purpose, we use a concept name A that does not occur in the input concept D and thus does not interfere with the other constraints. For implementation purposes, this rule could clearly be simplified,[5] but its current design facilitates the proofs.

5. The definition of a *clash* is extended to include those cases where there are conflicting functional restrictions. Given the $\geqslant$-rule as described, this is not strictly necessary, but it would be required if the $\geqslant$-rule did not create two logically disjoint successors.

Definition 22 In contrast to completion trees introduced in Section 4.1, in the following, each edge of completion trees is labelled with a *set* of roles.

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

A node x is *directly blocked* if 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 is *indirectly blocked* if its predecessor is blocked, and in order to avoid wasted expansion after an application of the $\leqslant$-rule, a node y will also be taken to be indirectly blocked if it is a successor of a node x and $\mathcal{L}(\langle x, y\rangle) = \emptyset$.

For a node x, $\mathcal{L}(x)$ is said to contain a *clash* if it contains an $\mathcal{ALCHI}_{R^+}$-clash, or, for roles R and S, $\{(\leqslant 1\ R), (\geqslant 2\ S)\} \subseteq \mathcal{L}(x)$ and $S \sqsubseteq^* R$.

[5] It is intuitively obvious that if $(\geqslant 2\ R) \in \mathcal{L}(x)$, and there is no conflicting functional restriction in $\mathcal{L}(x)$, then the sub-tree rooted in a single R-successor of x could be duplicated in order to satisfy $(\geqslant 2\ R)$.

$\sqcap$-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, $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C\}$ for some $C \in \{C_1, C_2\}$

$\exists$-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. there is an R-neighbour y of x 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 $\mathcal{L}(\langle x, y_1\rangle) = \{R\}$,
$\mathcal{L}(\langle x, y_2\rangle) = \{R\}$, $\mathcal{L}(y_1) = \{A\}$ 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. $\mathcal{L}(\langle x, y\rangle) \longrightarrow \emptyset$

Figure 7: The complete tableaux expansion rules for $\mathcal{ALCHFI}_{R^+}$

6.2 Soundness and Completeness

The soundness and completeness proof follows the same pattern as those for the other logics, but the tableaux construction proof is more complex as it must be able to create an infinite tableau.

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

PROOF. Very similar to the proof of Lemma 4, it only being necessary to show that there are a finite number of different node-relation-node triples. Let $m = |sub(D)|$ and $n = |\mathbf{R}_D|$. 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. 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.

2. Successors are only generated for concepts of the form $\exists R.C$ and $(\geqslant 2\ R)$. For a node x, each of these concepts triggers the generation of at most two successors y: note that if 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 enhanced definition of a clash, implies that the rule application which led to the generation of y will not be repeated. Since *sub*(D) contains a total of at most m $\exists R.C$ and $(\geqslant 2\ R)$ concepts, the out-degree of the tree is bounded by $2m$.

3. Nodes are labelled with nonempty subsets of *sub*$(D) \cup \{A, \neg A\}$ and edges with subsets of R_D, so there are at most 2^{2mn} different possible labellings for a pair of nodes and an edge. Therefore, if a path p is of length at least 2^{2mn}, then from the pair-wise blocking condition defined in Section 6.1 there must be 2 nodes x, y on p 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}. ∎

Lemma 24 (Soundness) If the expansion rules can be applied to an $\mathcal{ALCHFI}_{R^+}$-concept D such that they yield a complete and clash-free completion tree, then D has a tableau.

PROOF. Intuitively, the definition of a tableau $T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ from a complete and clash-free completion tree $\mathbf{T}$ works as follows: An individual in $\mathbf{S}$ corresponds to a *path* in $\mathbf{T}$ from the root node to some node that is not blocked. To obtain infinite tableaux, these paths may be *cyclic*. Instead of going to a directly blocked node, these paths go "back" to the blocking node—and this an infinite number of times. Thus, if blocking occurred while constructing a tableaux, we obtain an infinite tableau. [6]

More precisely, let $\mathbf{T}$ be a complete and clash-free completion tree. We will use the mapping $\mathsf{Tail}(p)$ to return the last element in a path p: given a path $p = [x_0, \ldots, x_n]$, where the x_i are nodes in $\mathbf{T}$, $\mathsf{Tail}(p) = x_n$. Paths in $\mathbf{T}$ are defined inductively as follows:

1. For the root node x_0 in $\mathbf{T}$, $[x_0]$ is a path in $\mathbf{T}$;
2. For a path p and a node x_i in $\mathbf{T}$, $[p, x_i]$ is a path in $\mathbf{T}$ iff
   (a) x_i is a successor of $\mathsf{Tail}(p)$ and x_i is not blocked, *or*
   (b) for some node y in $\mathbf{T}$, y is a successor of $\mathsf{Tail}(p)$ and x_i blocks y.

Now we can define a tableau $T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ with:

$$\begin{aligned}
\mathbf{S} &= \{x_p \mid p \text{ is a path in } \mathbf{T}\} \\
\mathcal{L}(x_p) &= \mathcal{L}(\mathsf{Tail}(p)) \\
\mathcal{E}(R) &= \{\langle x_p, x_q\rangle \in \mathbf{S} \times \mathbf{S} \mid \text{Either } q = [p, \mathsf{Tail}(q)] \text{ and} \\
&\qquad 1.\ \mathsf{Tail}(q) \text{ is an } R\text{-successor of } \mathsf{Tail}(p), \textit{ or} \\
&\qquad 2.\ \text{for some node } y \text{ in } \mathbf{T},\ y \text{ is an } R\text{-successor} \\
&\qquad\quad \text{of } \mathsf{Tail}(p) \text{ and } \mathsf{Tail}(q) \text{ blocks } y \\
&\qquad \text{or } p = [q, \mathsf{Tail}(p)] \text{ and} \\
&\qquad 1.\ \mathsf{Tail}(p) \text{ is an } \mathsf{Inv}(R)\text{-successor of } \mathsf{Tail}(q), \textit{ or} \\
&\qquad 2.\ \text{for some node } y \text{ in } \mathbf{T},\ y \text{ is an } \mathsf{Inv}(R)\text{-successor} \\
&\qquad\quad \text{of } \mathsf{Tail}(q) \text{ and } \mathsf{Tail}(p) \text{ blocks } y\}
\end{aligned}$$

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

[6] If a simplified $\geqslant$-rule were employed, as outlined in Section 6.1, then a more elaborate construction would be required, one that created duplicate paths as necessary in order to satisfy $(\geqslant 2\ R)$ concepts.

1. $D \in \mathcal{L}(x_0)$ for the root x_0 of $\mathbf{T}$ and $[x_0]$ is a path in $\mathbf{T}$. Hence $D \in \mathcal{L}(x_{[x_0]})$ for $x_{[x_0]} \in \mathbf{S}$.

2. The proof that Properties 1–3 of Definition 2 are satisfied is identical to the proof of Lemma 5.

3. The proof that Properties 4–6 of Definition 2 are satisfied is similar to that given in the proof of Lemma 5, with the additional observations that

   (a) The new Definition 22 of R-neighbours must be taken into account.

   (b) For all individuals $x_p \in \mathbf{S}$, the "immediate environment" of x_p is identical to that of the node $\mathsf{Tail}(p)$ in $\mathbf{T}$. To be more precise, for all nodes x in $\mathbf{T}$, if y is an R-neighbour of x, then for every individual $x_p \in \mathbf{S}$ with $\mathsf{Tail}(p) = x$ there is an individual $x_q \in \mathbf{S}$ such that $\langle x_p, x_q \rangle \in \mathcal{E}(R)$ and $\mathcal{L}(x_q) = \mathcal{L}(y)$. This is straightforward in the case where y is an R-successor of x: either $\mathsf{Tail}(q) = y$, or $\mathsf{Tail}(q) = z$ for some z that blocks y and $\mathcal{L}(z) = \mathcal{L}(y)$.

   However, in the case where x is an $\mathsf{Inv}(R)$-successor of y (so y is an R-neighbour of x), and x blocks some node z, the maintenance of this property crucially depends on the definition of pair-wise blocking: let w be the predecessor of z, q be a path with $\mathsf{Tail}(q) = w$ and p be the path $[q, x]$ resulting from the block. By definition $x_p \in \mathbf{S}$ with $\mathsf{Tail}(p) = x$. From pair-wise blocking we have that w is an R-neighbour of z, and $\mathcal{L}(w) = \mathcal{L}(y)$, so for $x_q \in \mathbf{S}$, $\langle x_p, x_q \rangle \in \mathcal{E}(R)$ and $\mathcal{L}(x_q) = \mathcal{L}(y)$.

4. Property 7 holds because of the symmetric definition of the mapping $\mathcal{E}$.

5. The proof that Properties 6′ and 8 of Definition 9 are satisfied is identical to the proof of Lemma 13.

6. Suppose Property 9 of Definition 20 were not satisfied. Let $x_p, x_q, x_{q'}$ be individuals in $\mathbf{S}$ with $(\leqslant 1\ R) \in \mathcal{L}(x_p)$, $\{\langle x_p, x_q \rangle, \langle x_p, x_{q'} \rangle\} \subseteq \mathcal{E}(R)$ and $q \neq q'$. This means that either

   (a) $\mathsf{Tail}(q)$ and $\mathsf{Tail}(q')$ are both R-neighbours of $\mathsf{Tail}(p)$, *or*

   (b) one of them, say $\mathsf{Tail}(q)$, is an R-neighbour of $\mathsf{Tail}(p)$ and $\mathsf{Tail}(q')$ blocks an R-neighbour y of $\mathsf{Tail}(p)$, but then both y and $\mathsf{Tail}(q)$ are R-neighbours of $\mathsf{Tail}(p)$, and $y \neq \mathsf{Tail}(q)$ because y is blocked while $\mathsf{Tail}(q)$ is not, *or*

   (c) $\mathsf{Tail}(q)$ and $\mathsf{Tail}(q')$ block R-neighbours y and z of $\mathsf{Tail}(p)$, but then both y and z are R-neighbours of $\mathsf{Tail}(p)$, with $y \neq z$ because $q \neq q'$ and a blocked node has a unique blocking node.

   In all three cases $\mathsf{Tail}(p)$ has two R-neighbours, the $\leqslant$-rule would be applicable, and $\mathbf{T}$ cannot be complete.

7. Property 10 of Definition 20 follows immediately from the $\geqslant$-rule and the definition of the tableau T: since $\mathbf{T}$ is clash-free, if $(\geqslant 2\ R) \in \mathcal{L}(x_p)$, then $\mathsf{Tail}(p)$ in $\mathbf{T}$ has two R-successors that cannot be blocked by the same node, hence $\langle x_p, x_q \rangle \in \mathcal{E}(R)$ and $\langle x_p, x_{q'} \rangle \in \mathcal{E}(R)$, with $q \neq q'$. ∎

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

Again, this proof is similar to the one for Lemma 6 and $\mathcal{ALCI}_{R^+}$-concepts. This is due to the fact that we did not introduce new non-deterministic rules. Given a tableau, we can trigger the application of the expansion rules such that they yield a complete and clash-free completion tree, with Properties 9 and 10 of Definition 20 ensuring that it is also complete and clash-free with respect to functional restrictions.

Theorem 26 The tableaux algorithm is a decision procedure for the satisfiability and subsumption of $\mathcal{ALCHFI}_{R^+}$-concepts with respect to terminologies.

Using the same techniques and arguments as in Section 5.3, general concept inclusion axioms can be internalised. Hence the tableaux algorithm is a decision procedure for satisfiability and subsumption of $\mathcal{ALCFI}_{R^+}$-concepts with respect to terminologies.

7 Summary and Related Work

The combination of transitive and inverse roles is important for the adequate representation of aggregated objects, allowing the simultaneous description of parts by means of the whole to which they belong and of wholes by means of their constituent parts. Using a new dynamic blocking technique we have been able to develop a cut-free tableaux algorithm for deciding satisfiability and subsumption in $\mathcal{ALCHI}_{R^+}$, a DL that extends $\mathcal{ALC}$ with both transitive and inverse roles, as well as a role hierarchy.

Moreover, by adding pair-wise blocking it has been possible to extend this algorithm to deal with $\mathcal{ALCHFI}_{R^+}$, a DL that additionally supports functional restrictions; this is in spite of the fact that $\mathcal{ALCHFI}_{R^+}$ no longer has the finite model property. Support for functional restrictions is important in several application domains. In particular, $\mathcal{ALCHFI}_{R^+}$ could be used to model E/R schemata (a formalism introduced by [7], and widely used for the conceptual modeling of relational databases). In the (common) case where the E/R-schema is of the kind where minimum cardinality restrictions are either 0 or 1 and maximum cardinality restrictions are either 1 or ∞, the technique presented in [6, 5] translates the schema into an $\mathcal{ALCFI}$ terminology (which contains general concept inclusion axioms); the method for reasoning with such terminologies using $\mathcal{ALCHFI}_{R^+}$ has been described in Section 5.3. Extending $\mathcal{ALCHFI}_{R^+}$ with general number restrictions would allow the same method to be used with E/R-schemata containing general cardinality restrictions; this will be part of future work.

Although $\mathcal{ALCHI}_{R^+}$ and $\mathcal{ALCHFI}_{R^+}$ are in the same complexity class as $\mathcal{ALC}$ augmented with the transitive closure and inverse role forming operators[7] (ExpTime-complete), there are good reasons to believe that they will have better computational properties.

Firstly, transitive closure is inherently harder than transitive roles: the extension of $\mathcal{ALC}$ with transitive closure ($\mathcal{ALC}_+$) is already ExpTime-hard (which is an easy consequence of results presented in [13, 25]), whereas the extension of $\mathcal{ALC}$ with transitive roles is still in PSpace-complete (which is a not so easy consequence [26] of results presented in [23, 15]).

[7] Which is a notational variant of *converse*-PDL if the set of program constructors is restricted to transitive closure and inverse.

Secondly, even with respect to $\mathcal{ALCH}_{R^+}$, which is in the same complexity class as $\mathcal{ALC}_+$, transitive closure appears to be considerably harder than transitive roles. Expanding $\exists R^+.C$-concepts introduces additional non-determinism because one must guess the length of an R-chain leading to an instance of C. Moreover, blocking is more involved because, in $\mathcal{ALC}_+$, a block may represent a contradiction since it might only indicate a postponement of the satisfaction of an $\exists R^+.C$-concept in an inherently unsatisfiable context, a situation known as a *bad cycle* [1].

Thirdly, another difficulty must be dealt with if inverse roles are present. To our knowledge, the only technique for distinguishing the above mentioned bad cycles uses the so-called *cut rule* (an algorithm for *converse*-PDL employing a cut rule is presented in [10]); the application of this rule leads to considerable additional non-determinism. Intuitively, the cut rule guesses, for each subconcept C of the input concept and for each blocking node x, whether C or $\neg C$ holds at x. Moreover, the $\mathcal{ALCHFI}_{R^+}$ algorithm deals directly with functional restrictions, rather than using an embedding [12] which is yet another source of non-determinism.

Finally, a large fragment of $\mathcal{ALCHFI}_{R^+}$, namely $\mathcal{ALCFI}_{R^+}$, is still in Pspace [22]. A corresponding relationship holds between $\mathcal{ALCH}_{R^+}$ and $\mathcal{ALC}_{R^+}$, and implementations of $\mathcal{ALCH}_{R^+}$ in the FaCT and DLP systems [17, 24] have been shown to behave well in realistic applications [20]. A possible explanation for this phenomenon is that the constructor which makes these logics ExpTime-hard, namely the role hierarchy, is mainly used for the encoding of axioms. According to our experiences, realistic knowledge bases contain few axioms that are not amenable to the absorption optimisation technique described in [18].

Moreover, the $\mathcal{ALCH}_{R^+}$ algorithm has been shown to be amenable to a range of other optimisation techniques [21]; we believe that both its good behaviour and the optimisation techniques will carry over into both $\mathcal{ALCHI}_{R^+}$ and $\mathcal{ALCHFI}_{R^+}$. To verify this belief, these new algorithms are to be implemented in a descendant of the FaCT system.

Acknowledgements

We would like to thank the anonymous referees for their valuable comments and suggestions.

References

[1] F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 446–451, 1991.

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

[3] F. Baader, H.-J. Burckert, 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.

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

[5] D. Calvanese. *Unrestricted and Finite Model Reasoning in Class-Based Representation Formalisms*. PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1996.

[6] D. Calvanese, M. Lenzerini, and D. Nardi. A unified framework for class based representation formalisms. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR-94)*, pages 109–120, Bonn, 1994. Morgan Kaufmann, Los Altos.

[7] P. P. Chen. The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[8] G. De Giacomo. *Decidability of Class-Based Knowledge Representation Formalisms*. PhD thesis, Dip. di Inf. e Sist., Univ. di Roma "La Sapienza", 1995.

[9] G. De Giacomo and M. Lenzerini. Tbox and Abox reasoning in expressive description logics. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning (KR-96)*, pages 316–327. Morgan Kaufmann, Los Altos, 1996.

[10] G. De Giacomo and F. Massacci. Combining deduction and model checking into tableaux and algorithms for converse-pdl. *Information and Computation*, 1998. To appear.

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

[12] G. De Giacomo and M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics (extended abstract). In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, 1994.

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

[14] 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)*. CEUR, May 1998.

[15] J. Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logic of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.

[16] 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.

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

[18] I. Horrocks. Using an expressive description logic: FaCT or fiction? In A. G. Cohn, L. Schubert, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)*, pages 636–647. Morgan Kaufmann Publishers, San Francisco, CA, June 1998.

[19] I. Horrocks and G. Gough. Description logics with transitive roles. In M.-C. Rousset, R. Brachmann, F. Donini, E. Franconi, I. Horrocks, and A. Levy, editors, *Proceedings of the International Workshop on Description Logics*, pages 25–28, Gif sur Yvette, France, 1997. Université Paris-Sud.

[20] I. Horrocks and P. F. Patel-Schneider. Comparing subsumption optimizations. In Franconi et al. [14], pages 90–94.

[21] I. Horrocks and P. F. Patel-Schneider. Optimising propositional modal satisfiability for description logic subsumption. In J. Calmet and J. Plaza, editors, *Artificial Intelligence and Symbolic Computation: International Conference AISC'98*, number 1476 in Lecture Notes in Artificial Intelligence, pages 234–246. Springer-Verlag, September 1998.

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

[23] R. E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal of Computing*, 6(3):467–480, 1977.

[24] P. F. Patel-Schneider. DLP system description. In Franconi et al. [14], pages 87–89.

[25] V. R. Pratt. Models of program logics. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, San Juan, Puerto Rico, 1979.

[26] 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.

[27] U. Sattler. *Terminological knowledge representation systems in a process engineering application*. PhD thesis, RWTH Aachen, 1998.

[28] K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 466–471, Sydney, 1991.

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