A concept language extended with different kinds of transitive roles

Ulrike Sattler

RWTH Aachen
Lehr- und Forschungsgebiet Theoretische Informatik
Ahornstr. 55
52074 Aachen
Germany

email: uli@cantor.informatik.rwth-aachen.de

Phone: +49-241-8021130
+49-241-804566

Fax: +49-241-8888360

Fachbeitrag

A concept language extended with different kinds of transitive roles

Ulrike Sattler*

RWTH Aachen, uli@cantor.informatik.rwth-aachen.de

Abstract. Motivated by applications that demand for the adequate representation of part-whole relations, different possibilities of representing transitive relations in terminological knowledge representation systems are investigated. A well-known concept language, $\mathcal{ALC}$, is extended by three different kinds of transitive roles. It turns out that these extensions differ largely in expressiveness and computational complexity, hence this investigation gives insight into the diverse alternatives for the representation of transitive relations such as part-whole relations, family relations or partial orders in general.

1 Introduction

Terminological knowledge representation systems (TKR-systems) are powerful means to represent the unambiguous, well-defined terminological knowledge in technical and other domains. Mainly, TKR-systems consist of two parts: A knowledge base, which contains the explicit concept definitions given in a so-called concept language, and an inference engine which is able to infer implicit properties of the defined concepts such as satisfiability or subclass/superclass relations among these concepts. A concept language is characterized by a set of operators that can be used to define complex concepts (which are interpreted as subsets of an interpretation universe) and roles (which are interpreted as binary relations on an interpretation universe) from primitive concepts and roles.

Looking for a concept language with sufficient expressive power to be used for the representation of complex objects, one observes that part-whole relations are indispensable in the description of these objects. The following concept, for example, describes devices having at least one part that is a battery: `device` ⊓ (∃ `has_part.battery`).

Since part-whole relations have special properties used for reasoning about complex objects, they are subject to a great variety of investigations [ACG+94; Pri95; Sim87; Fra94]. A point of view supported by most of them is that there are different part-whole relations (such as component–aggregate and ingredient–object) with different properties, and that there is a general transitive part-whole relation. Hence part-whole relations deserve special attention and cannot be represented by simple binary relations.

Even if a concrete decomposition of a given object may seem object inherent and natural, it is rather arbitrary in most cases. This can be seen by comparing decompositions of the same object made by different persons or made with different intentions. Hence, given an object, it is rarely possible to associate an exact level of decomposition to each of its parts. In one decomposition, two parts may be found in the same level whereas in another decomposition, these parts are at different levels. Thus, if we want to address a part of an object, it might be necessary to address various levels of decomposition. Furthermore, it might be necessary to refer to *all* levels of decomposition or to *all* parts of an object. If the maximum depth of decomposition is known in advance, this can be achieved by using a concept language that allows for disjunction of concepts, as for example in the following concept that describes devices having a carcinogenic part at some level of decomposition:

```
device ⊓ ((∃has_part.carcinogenic) ⊔
          (∃has_part.(∃has_part.carcinogenic)) ⊔
          (∃has_part.(∃has_part.(∃has_part.carcinogenic))) ⊔ ...)
```

If this maximum depth is not known in advance, other, more expressive means have to be used to refer to these parts[2]. A first approach is to represent the part-whole relation by a transitive role `has_some_part` that is interpreted as a transitive relation. Using this role, we are now able to represent dangerous devices as given above by `device ⊓ (∃has_some_part.carcinogenic)`. On the other hand, there are cases where we want to distinguish between a *direct* part and a part of a part of . . . , for example, this might be the case if an object is decomposed into components and we want to distinguish between a device equipped with a battery and a device having—at some level of decomposition— a part that has a battery. A concept language having the expressive power to formulate this difference is one where, beside primitive roles, one is allowed to use the *transitive closure R^+* of a role R. The above examples can then be described by

$$device ⊓ (∃has\_part.battery) \quad \text{and}$$
$$device ⊓ (∃has\_part^+.(∃has\_part.battery)).$$

Unfortunately, extending the well-known concept language $\mathcal{ALC}$ [SS91] by the transitive closure of roles severely increases the computational complexity of the according inference problems such as subsumption or satisfiability. Even if this increase in complexity can be justified by the simultaneous growth in expressive power, one might not be willing to accept this complexity, and look for an alternative to the transitive closure of roles. Using results from Modal Logic [Lad77; HM92], it can be shown that the use of transitive roles is in general less expensive in terms of complexity than the use of the transitive closure of roles. In fact, it is the interaction between a role and its transitive closure which is responsible for the high computational complexity.

---

[2] As pointed out in [LB87], an important aspect of expressiveness is, however, "what can be left unsaid" in a representation.

The natural question arising here is whether there exists an alternative for the representation of transitive relations: One that is more expressive than transitive roles, and which has less dramatic effects on the computational complexity than the transitive closure of roles. Since the transitive closure of a relation $\mathcal{R}$ is the *smallest* transitive relation containing $\mathcal{R}$, a natural candidate for this alternative is *some* transitive relation containing $\mathcal{R}$.

In this paper, we present three extensions of the concept language $\mathcal{ALC}$ by different kinds of transitive roles:

– In $\mathcal{ALC}_+$, the operator $^+$ can be applied to role names. The role R^+ is then interpreted as the smallest transitive relation containing R.
– In $\mathcal{ALC}_{R^+}$, certain roles have to be interpreted as transitive roles without the possibility to relate them to a generating role as in the first extension.
– In $\mathcal{ALC}_\oplus$, the operator $^\oplus$ can be applied to role names. The role $R^\oplus$ is then interpreted as some (not necessarily the smallest) transitive relation containing R.

As a consequence of the results given in [FL79], the basic inference problems for $\mathcal{ALC}$ extended by the transitive closure of roles, $\mathcal{ALC}_+$, are EXPTIME-complete, whereas these problems are PSPACE-complete for $\mathcal{ALC}$. Using results from modal logic [Lad77; HM92], we show that these problems remain PSPACE-complete for $\mathcal{ALC}_{R^+}$. Finally, it turns out that for $\mathcal{ALC}_\oplus$, these problems are as hard as for $\mathcal{ALC}_+$, namely EXPTIME-complete.

2   Preliminaries

The concept language underlying this investigation is $\mathcal{ALC}$, a well-known concept language introduced by [SS91] and investigated, for example, in [HNS90; DLNN91; DLNN95].

Definition 1. Let N_C be a set of *concept names* and let N_R be a set of *role names*. The set of $\mathcal{ALC}$-*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 a role name, then $(C \sqcap D)$, $(C \sqcup D)$, $(\neg C)$, $(\forall R.C)$, $(\exists R.C)$ 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

$$(C \sqcap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I}$$
$$(C \sqcup D)^\mathcal{I} = C^\mathcal{I} \cup D^\mathcal{I}$$
$$\neg C^\mathcal{I} = \Delta^\mathcal{I} \setminus C^\mathcal{I}$$
$$(\exists R.C)^\mathcal{I} = \{d \in \Delta^\mathcal{I} \mid \text{There exists some } e \in \Delta^\mathcal{I} \text{ with } (d, e) \in R^\mathcal{I} \text{ and } e \in C^\mathcal{I}\}$$
$$(\forall R.C)^\mathcal{I} = \{d \in \Delta^\mathcal{I} \mid \text{For all } e \in \Delta^\mathcal{I}, \text{ if } (d, e) \in R^\mathcal{I}, \text{ then } e \in C^\mathcal{I}\}$$

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 $x \in C^{\mathcal{I}}$.

One can observe that extending $\mathcal{ALC}$ by composition or disjunction of roles does not change the expressive power of $\mathcal{ALC}$ because of the following equivalences:
$$\exists(R \circ S).C \equiv \exists R.(\exists S.C) \quad \text{and} \quad \exists(R \sqcup S).C \equiv (\exists R.C) \sqcup (\exists S.C).$$
In contrast, extending $\mathcal{ALC}$ by the transitive closure of roles really increases its expressive power (see [Baa91]).

3 $\mathcal{ALC}$ Extended by the Transitive Closure of Roles

Definition 2. $\mathcal{ALC}_+$ is the extension of $\mathcal{ALC}$ obtained by allowing, for each role $R \in N_R$, the use of its transitive closure R^+ inside concepts. Interpretations have to satisfy additionally:
$$(d, e) \in (R^+)^{\mathcal{I}}$$
$$\text{iff}$$
for $k \geq 1$ exists $d_0 = d, d_1, \ldots, d_k = e$ such that $(d_i, d_{i+1}) \in R^{\mathcal{I}}$ for all $i < k$.

On one hand, the effect of this extension on the expressive power can be seen by noting that $\mathcal{ALC}_+$ can no longer be viewed as a subclass of first order logic. This is due to the fact that the transitive closure of a relation cannot be expressed in first order logic, in contrast to transitivity.

On the other hand, $\mathcal{ALC}$ loses the *finite tree model property*[3] when extended by the transitive closure of roles. For example, the following concept describes instances of A having some R-successor in A and where each individual reachable over some R-path has itself some R-successor in A:

$$A \sqcap (\exists R.A) \sqcap (\forall R^+.(\exists R.A))$$

This concepts is satisfiable, but each of its models has either an infinite R-chain or it contains some R-cycle.

As a consequence of this fact, algorithms (like tableau-based algorithms) that try to construct a model of a concept containing the transitive closure of roles need special "cycle detection mechanisms" [Baa91]: They have to distinguish between cases where constraints on individuals propagated along some (possibly infinite) role chain are simply regenerated but satisfied and cases where their satisfaction is postponed in each step. This could happen for example while trying to construct a model of

$$\mathsf{A} \sqcap (\exists \mathsf{R}^+.\neg\mathsf{A}) \sqcap (\forall \mathsf{R}^+.\mathsf{A}).$$

---

[3] A concept language has the finite tree model property if each satisfiable concept has a finite tree model.

This cycle detection demands for the storage of a high amount of information and cannot be accomplished using polynomial space: As stated in Theorem 3.1. and Theorem 4.4. of [FL79], satisfiability of $\mathcal{ALC}_+$-concepts is Exptime-complete. A translation of this result from dynamic logic to the vocabulary of concept languages can be found in [Sch91].

4 $\mathcal{ALC}$ Extended by Transitive Roles

Recently, results from the field of modal logic gave new insight into problems concerning concept languages: It is well-known [Sch91] that $\mathcal{ALC}$ is a notational variant of propositional multi-modal logic $\mathbf{K}_n$. In the present work, results for the modal logic $\mathbf{K4}_n$, which is a multi modal logic with n so-called agents extending propositional logic, gave the impetus to look closer at transitive roles. If $\mathcal{ALC}$-interpretations are restricted to those where all role names are interpreted as transitive relations, then there is a $1-1$ correspondance between $\mathbf{K4}_n$-formulae and $\mathcal{ALC}$-concepts in such a way that a formula ϕ is satisfiable iff its translation is satisfiable with respect to the restricted semantics. Since it is shown in [HM92] that satisfiability of $\mathbf{K4}_n$ formulae is Pspace-complete, it is not surprising that we can even show that satisfiability of $\mathcal{ALC}$ extended by transitive roles (beside ordinary roles) is also Pspace-complete.

Definition 3. $\mathcal{ALC}_{R^+}$ is an extension of $\mathcal{ALC}$ obtained by allowing the use of transitive roles inside concepts. The set of role names N_R is a disjoint union of role names $N_P = \{P_1, P_2, \ldots\}$ and role names $N_+ = \{R_1, R_2, \ldots\}$. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ has to satisfy additionally

$$\text{if } (d, e) \in R_i{}^{\mathcal{I}} \text{ and } (e, f) \in R_i{}^{\mathcal{I}}, \text{ then } (d, f) \in R_i{}^{\mathcal{I}}$$

for each role $R_i \in N_+$.

In this section, a tableau based algorithm is presented that tests for the satisfiability of $\mathcal{ALC}_{R^+}$-concepts. The algorithm extends and combines those presented in [HM92] for multi modal logics in order to deal with the simultaneous use of both ordinary and transitive roles. It will be shown that this algorithm uses space polynomial in the length of the concept.

For simplicity, all concepts are supposed to be in *negation normal form*. This means that negation is applied to concept names only. A concept can be transformed into an equivalent one in negation normal form by pushing negation into concepts, for example $\neg(C \sqcup D) \equiv \neg C \sqcap \neg D$ and $\neg(\exists R.C) \equiv (\forall R.\neg C)$.

The tableau algorithm given below constructs a tree whose nodes represent individuals. Each node is labelled with a set of $\mathcal{ALC}_{R^+}$-concepts. When started with an $\mathcal{ALC}_{R^+}$-concept D in negation normal form, these sets can be restricted to subconcepts $\text{sub}(D)$ of D. It is easy to see that the number of subconcepts of D is linear in the length of D. Soundness and completeness of the tableau algorithm will be proved by showing that it creates a so-called tableau:

Definition 4. Let D be a $\mathcal{ALC}_{R^+}$-concept and let $\{P_1, \ldots P_n, R_1, \ldots R_m\}$ be the set of role names occuring in D. A *tableau* $T = (S, L, \mathcal{P}_1, \ldots, \mathcal{P}_n, \mathcal{R}_1, \ldots \mathcal{R}_m)$ for D is defined as follows: S is a set of individuals, $\mathcal{P}_i, \mathcal{R}_i \subseteq S \times S$, and $L : S \to 2^{\mathrm{sub}(D)}$ matches each individual to a set of subconcepts of D such that:

1. for some $s_0 \in S$ we have $D \in L(s_0)$, and for all $s \in S$ it holds that:
2. if $C \in L(s)$, then $\neg C \notin L(s)$,
3. if $C_1 \sqcap C_2 \in L(s)$, then $C_1 \in L(s)$ and $C_2 \in L(s)$,
4. if $C_1 \sqcup C_2 \in L(s)$, then $C_1 \in L(s)$ or $C_2 \in L(s)$,
5. if $(\forall P_i.C) \in L(s)$ and $(s, t) \in \mathcal{P}_i$, then $C \in L(t)$,
6. if $(\exists P_i.C) \in L(s)$, then there is some $t \in S$ with $(s, t) \in \mathcal{P}_i$ and $C \in L(t)$,
7. if $(\forall R_i.C) \in L(s)$ and $(s, t) \in \mathcal{R}_i$, then $C \in L(t)$ and $(\forall R_i.C) \in L(t)$,
8. if $(\exists R_i.C) \in L(s)$, then there is some $t \in S$ with $(s, t) \in \mathcal{R}_i$ and $C \in L(t)$.

Lemma 5. *An $\mathcal{ALC}_{R^+}$-concept D is consistent iff there exists a tableau for D.*

Sketch of the proof: Let $T = (S, L, \mathcal{P}_1, \ldots, \mathcal{P}_n, \mathcal{R}_1, \ldots \mathcal{R}_m)$ be a tableau for D, define $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ as follows:

$\Delta^\mathcal{I} := S$,
for all $C \in \mathrm{sub}(D)$ define $s \in C^\mathcal{I}$ iff $C \in L(s)$,
$P_i^\mathcal{I} := \mathcal{P}_i$,
$R_i^\mathcal{I} := \mathcal{R}_i{}^+$ where $\mathcal{R}_i{}^+$ denotes the transitive closure of $\mathcal{R}_i$.

By induction on the structure of concepts, it can be shown that $\mathcal{I}$ is well-defined and that $D^\mathcal{I} \neq \emptyset$. For concepts of the form $(C_1 \sqcap C_2), \neg C_1, (\exists P_i.C), (\forall P_i.C)$ and $(\exists R_i.C)$ it follows immediately that they are correctly interpreted. If we have $(\forall R_i.C) \in L(s)$, $(s, t) \in \mathcal{R}_i$ and $(t, u) \in \mathcal{R}_i$, then $(\forall R_i.C) \in L(t)$ and $C \in L(u)$. Hence $s \in (\forall R_i.C)^\mathcal{I}$ holds and concepts of the form $(\forall R_i.C)$ are also correctly interpreted.

For the converse, let $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ be a model of D. Define $T = (S, L, \mathcal{P}_1, \ldots, \mathcal{P}_n, \mathcal{R}_1, \ldots, \mathcal{R}_m)$ with:

$$S := \Delta^\mathcal{I},$$
$$\mathcal{P}_i := P_i^\mathcal{I} \text{ for } P_i \in N_P,$$
$$\mathcal{R}_i := R_i^\mathcal{I} \text{ for } R_i \in N_+,$$
$$L(s) := \{C \in \mathrm{sub}(D) \mid s \in C^\mathcal{I}\}.$$

It follows by construction that T is a tableau for D. ∎

Using Lemma 5, an algorithm which constructs a tableau for an $\mathcal{ALC}_{R^+}$-concept D can be used as a decision algorithm for satisfiability of D. The algorithm given here builds a tree starting with a single node and expanding it by either expanding labels of its leafs or by adding new nodes. Nodes of this tree are labelled with sets of subconcepts of D and are possibly marked "satisfiable".

Edges are either unlabelled or they are labelled with j or j^+ for role names P_j, R_j occuring in D (unlabelled edges are generated when testing whether an individual satisfies a disjunction because it satisfies the first or the second

R1 Construct a tree $\mathcal{T}$ consisting of a node x_0 labelled with $L(x_0) = \{D\}$.

R2 Repeat (a) to (d) and possibly expand $\mathcal{T}$ until none of them applies:

  (a) (Pre-tableau) If x_i is a leaf of $\mathcal{T}$, $L(x_i)$ is clash-free, $L(x_i)$ is not a pre-tableau and C is the least witness to this fact, then

$$\text{if } C = C_0 \sqcap C_1, \text{ then } L(x_i) := L(x_i) \cup \{C_0, C_1\}.$$
$$\text{if } C = C_0 \sqcup C_1, \text{ then create two successors } x_{i0}, x_{i1} \text{ of } x_i \text{ with}$$
$$L(x_{ij}) := L(x_i) \cup \{C_j\}.$$

  (b) (Successors) For x_i a leaf of $\mathcal{T}$, $L(x_i)$ a clash-free pre-tableau, do:

    For each $(\exists P_j.C) \in L(x_i)$ create a j-successor x_{ij} with
$L(x_{ij}) := \{C\} \cup L(x_i)/P_j$.

    For each $(\exists R_j.C) \in L(x_i)$, let
$\ell(x_i, (\exists R_j.C)) := \{C\} \cup L(x_i)/R_j \cup \{(\forall R_j.E) \mid (\forall R_j.E) \in L(x_i)\}$.
If for some ancestor w of x_i: $L(w) \supseteq \ell(x_i, (\exists R_j.C))$,
   then create a j^+-successor x_{ij^+} with $L(x_{ij^+}) := \emptyset$,
   else create a j^+-successor x_{ij^+} with $L(x_{ij^+}) := \ell(x_i, (\exists R_j.C))$.

  (c) Mark a node x "satisfiable" iff

    - $L(x)$ is not a pre-tableau and some successor of x is marked "satisfiable".
    - $L(x)$ is a clash-free pre-tableau which does not contain a concept of the form $(\exists R_j.C)$ or $(\exists P_j.C)$.
    - $L(x)$ is a pre-tableau, x has successors, and all of them are marked "satisfiable".

R3 If the root is marked "satisfiable", return "D is satisfiable" else "D is unsatisfiable".

Fig. 1. Tableau construction for a $\mathcal{ALC}_{R^+}$-concept D

disjunct). A node y which is a successor of a node x is called a j- (resp. j^+-) successor of x if the edge between them is labelled with j (resp. j^+). A node x' is called a pre-successor of x if there is an unlabelled path from x to x'. A node x is called an ancestor of a node y if there is a path from x to y regardless of the labels of its edges. Concerning the labels of the nodes, the following abbreviations are introduced.

Let M be a set of concepts. We call M a *pre-tableau* iff M satisfies conditions 2–4 of Definition 4 with M in place of $L(s)$. We say that M *contains a clash* iff there is a concept C with $\{C, \neg C\} \subseteq M$. For a role name $R \in N_P \cup N_+$, let $M/R := \{C \mid (\forall R.C) \in M\}$. The maximum role depth of M, $\mathrm{depth}(M)$, is the maximum of nested $(\exists R.C)$, $(\forall R.C)$ concepts of all concepts in M.

For the construction, we assume that concepts (in $\mathrm{sub}(D)$) are linearly ordered and that $\{P_1, \ldots, P_n, R_1, \ldots, R_m\}$ is the set of role names occuring in D. The algorithm is given in Figure 1, and two examples of the tableau construc-
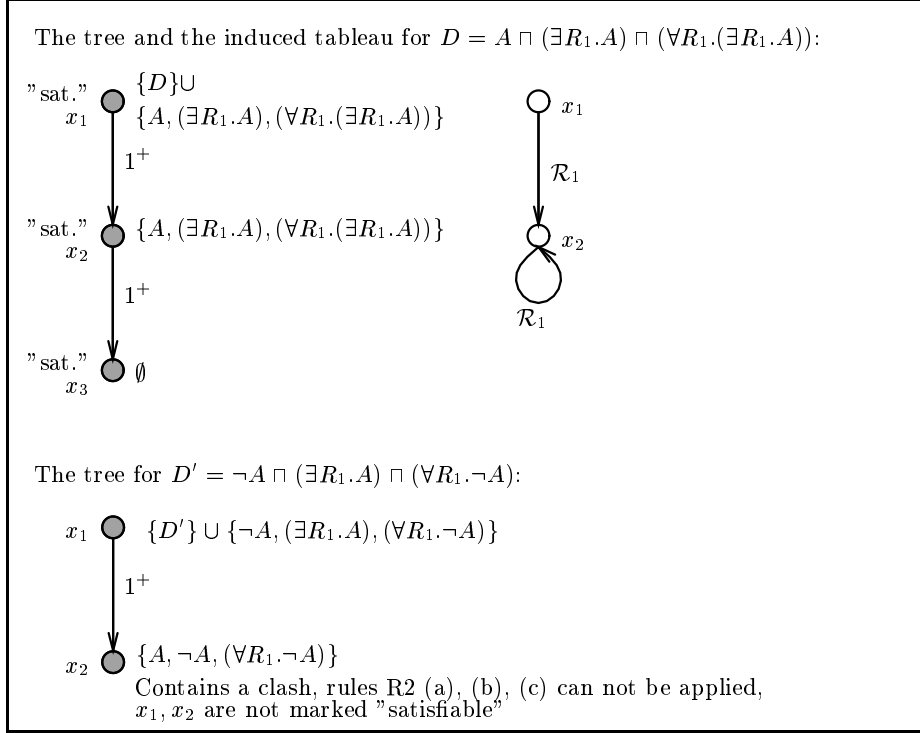
The tree and the induced tableau for $D = A \sqcap (\exists R_1.A) \sqcap (\forall R_1.(\exists R_1.A))$:

The tree for $D' = \neg A \sqcap (\exists R_1.A) \sqcap (\forall R_1.\neg A)$:

Fig. 2. Two examples for tableau construction

tions can be found in Figure 2. Please note that the empty set is a clash-free pre-tableau.

Lemma 6. *For each $\mathcal{ALC}_{R^+}$-concept D, the tableau construction terminates.*

Proof: Let $|\text{sub}(D)| = m$. We have $\text{depth}(L(x)) \leq m$ for all nodes x. Since nodes are labelled with subsets of $\text{sub}(D)$, $|L(x)| \leq m$ for all nodes x. Furthermore, if $C \in L(x)$, then $C \in L(x')$ for all pre-successors x' of x.

Besides showing termination, we want to give also an upper bound for the space needed by the algorithm, hence we investigate the depth of the tree constructed more closely.

Fact 1: Rule R2 (a) can be applied at most m times along an unlabelled path until it creates a node x such that $L(x)$ contains a clash.

Fact 2: If y is a j-successor of x, then $\text{depth}(L(y)) < \text{depth}(L(x))$. If y is a j^+-successor of x, $C \in L(y)$ and C is not of the form $(\forall R_j^+.C_1)$, then $\text{depth}(\{C\}) < \text{depth}(L(x))$.

Fact 3: If z is a j- or a j^+-successor of y, y is a k- or a k^+-successor of x and $j \neq k$, then $\text{depth}(L(z)) < \text{depth}(L(x))$.

Fact 4: The only way that the depth of the labels does not decrease is along some mixed j^+- and pre-path. Let $x_0, \ldots, x_k$ be nodes on such a path labelled with clash-free pre-tableaux $L(x_i)$ such that each x_i is a j^+-predecessor of $\tilde{x}_i$ and a pre-successor of $\tilde{x}_{i-1}$. Then each $L(\tilde{x}_i)$ can be divided into two parts $L1_i, L2_i$: The first consists of concepts of the form $\{(\forall R_j.E) \mid (\forall R_j.E) \in L(x_i)\} \cup L(x_i)/R_j$ and by construction we have $L1_i \subset L1_{i+1}$. The second is $L2_i = \{C\}$ where $(\exists R_j.C)$ led to the creation of $\tilde{x}_i$. Since for all ancestors w of $\tilde{x}$, we have $L(\tilde{x}_i) \nsubseteq L(w)$ or $L(\tilde{x}_i) = \emptyset$ by construction, there are at most m choices for $L2_i$ and at most m different choices for $L1_i$. Hence we have $k \le m^2$.
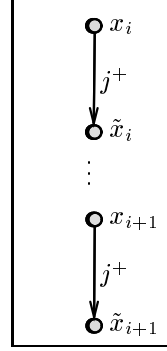
Collecting these facts, we have that the tree built by the tableau construction algorithm has depth at most m^4 and it is of bounded out-degree. Hence its construction terminates. ∎

Lemma 7. *A $\mathcal{ALC}_{R^+}$-concept D is satisfiable iff the tableau construction for D returns "D is satisfiable".*

Proof: Let $\mathcal{T}$ be the tree constructed by the tableau construction algorithm for D. Define a tableau $T = (S, L, \mathcal{P}_1, \ldots, \mathcal{P}_n, \mathcal{R}_1, \ldots \mathcal{R}_m)$ with

$$S = \{x \mid x \text{ is a node in } \mathcal{T}, x \text{ is marked satisfiable,}$$
$$\text{and } L(x) \text{ is a non-empty, clash-free pre-tableau}\},$$

$(x, y) \in \mathcal{P}_j$ iff y is a pre-successor of a j-successor of x,

$(x, y) \in \mathcal{R}_j$ iff y is a pre-successor of a j^+-successor of x and $L(y) \ne \emptyset$ or x has a j^+-successor z with $L(z) = \emptyset$, y is an ancestor of x and $L(y) \supseteq \ell(x, (\exists R_j.C))$

It is easy to see that T is a tableau for D: First, $D \in L(x)$ for all pre-successors x of the root x_0 of $\mathcal{T}$. Leafs of this subtree are either labelled with clash-free pre-tableaus or their labels contain a clash. If x_0 is marked "satisfiable", at least one of these leafs is marked "satisfiable", hence $D \in L(s)$ for some $s \in S$.

T satisfies properties 2–4 of Definition 4 because each $x \in S$ is labelled with a clash-free pre-tableau. R2 (c) creates for each $(\exists P_j.C) \in L(x_i)$ (resp. for those $(\exists R_j.C) \in L(x_i)$ where it is necessary) a j-successor (resp. j^+-successor) x_{ij} such that $C \in L(x_{ij})$, hence properties 6 and 8 are satisfied. Property 5 is satisfied because $L(x_i)/P_j \subseteq L(x_{ij})$ holds for all j-successors of x_i. Finally, property 7 holds because $L(x_i)/R_j \cup \{(\forall R_j.E) \mid (\forall R_j.E) \in L(x_i)\} \subseteq L(y)$ holds for all y with $(x_i, y) \in \mathcal{R}_j$.

For the converse, we show by induction on $h(x)$, the height of the subtree below x that, if x is not marked "satisfiable", then $X := \sqcap_{C \in L(x)} C$ is not satisfiable.

Let $h(x) = 0$, hence x is a leaf. If x is not marked satisfiable, it contains a clash and X is clearly unsatisfiable. Now let $h(x) = \ell + 1$. If $L(x)$ is not a pre-tableau and x is not marked satisfiable, then none of its successors is marked satisfiable. Hence we have $C_1 \sqcup C_2 \in L(x)$ and neither x_1 with $L(x_1) = L(x) \cup \{C_1\}$ nor x_2

with $L(x_2) = L(x) \cup \{C_2\}$ is marked satisfiable. It follows by induction that X is not satisfiable. If $L(x)$ is a pre-tableau and x is not marked satisfiable, then there is either some j- or j^+-successor of x which is not marked satisfiable or $L(x)$ does not contain any subconcept of the form $(\exists R.C)$ but contains a clash. In both cases, it follows by induction that X is not satisfiable. ∎

Theorem 8. *Satisfiability of $\mathcal{ALC}_{R^+}$-concepts can be decided using polynomial space.*

Proof: As stated in the proof of Lemma 6, the tree $\mathcal{T}$ constructed by the tableau construction algorithm for D is of depth at most m^4 where $|\mathrm{sub}(D)| \leq m$. Once this algorithm has marked a node satisfiable, it can forget about the subtree below this node and reuse the space where it was memorized. Since each $L(x)$ is a subset of $\mathrm{sub}(D)$, each $L(x)$ can be stored in m bits. Since there are less than m concepts of the form $(\exists R.C)$ in $\mathrm{sub}(D)$, there are less than m subtrees directly below a node x, and we can memorize which of them still have to be investigated in m bits.

Hence at each moment the algorithm is running, it has to store the following information for its actual node x at depth h: $L(x)$; which of the subtrees below x still have to be investigated, and these two pieces of information for each of its h ancestors. This can be stored in $m + m + h(m + m) = (1 + h)2m$ bits. Since $h \leq m^4$, the tableau construction algorithm needs at most $c + 2m + 2m^5$ bits of storage for some constant c. ∎

Theorem 9. *Satisfiability of $\mathcal{ALC}_{R^+}$-concepts is* PSPACE-*complete.*

As $\mathcal{ALC}$ is a sublanguage of $\mathcal{ALC}_{R^+}$, PSPACE-hardness of satisfiability of $\mathcal{ALC}_{R^+}$-concepts follows immediately from PSPACE-completeness of satisfiability of $\mathcal{ALC}$-concepts (see [SS91]). PSPACE-completeness is then implied by Theorem 8. ∎

As we have seen, worst-case complexity of $\mathcal{ALC}_{R^+}$ is lower than those of $\mathcal{ALC}_+$. The price in expressive power one has to pay for this lower complexity is illustrated by the following example: Let a queen be defined as a women whose children are princes or princesses and whose descendants are nobles:

$$\mathtt{queen} = \mathtt{women} \sqcap (\forall \mathtt{child}.(\mathtt{prince} \sqcup \mathtt{princess})) \sqcap (\forall \mathtt{child}^+.\mathtt{noble}). \quad (1)$$

This cannot be expressed if only a transitive role $\mathtt{descendants}$ can be used without the possibility to refer to successors of a subrole $\mathtt{child}$: In 1, we express that all individuals p for which the longest $\mathtt{child}$-path from an instance q of $\mathtt{queen}$ to p is of length 1 are instances of $(\mathtt{prince} \sqcup \mathtt{princess})$ and that all individuals reachable over *some* $\mathtt{child}$-path from q are instances of $\mathtt{noble}$. If only transitive roles are available as in $\mathcal{ALC}_{R^+}$, we can not distinguish between those "close" role successors and those reachable over some longer $\mathtt{child}$-path.

5 $\mathcal{ALC}$ Extended by Transitive Orbits of Roles

The investigation of $\mathcal{ALC}$ extended by transitive orbits, $\mathcal{ALC}_\oplus$, was motivated by the gap between $\mathcal{ALC}_{R+}$ and $\mathcal{ALC}_+$ in both computational complexity and expressive power. $\mathcal{ALC}_\oplus$ is the natural candidate for a compromise between $\mathcal{ALC}_{R+}$ and $\mathcal{ALC}_+$ because, on one hand, it allows to relate a relation with a transitive superrelation and, on the other hand, there is a chance that its handling could be algorithmically easier.

Definition 10. $\mathcal{ALC}_\oplus$ is an extension of $\mathcal{ALC}$ obtained by allowing the use of *transitive orbits* of roles inside concepts. The transitive orbit of a role R is denoted $R^\oplus$ and interpreted as a transitive role containing $R^\mathcal{I}$, i.e., we have

$$\text{if there exist } d = d_0, d_1, \dots, d_k = e \text{ with } (d_i, d_{i+1}) \in R^\mathcal{I} \text{ for all } i < k$$
$$\text{then } (d, e) \in (R^\oplus)^\mathcal{I}.$$

A small example is given to highlight the difference between $\mathcal{ALC}_+$ and $\mathcal{ALC}_\oplus$ concepts. Let $* \in \{+, \oplus\}$, and let

$$\texttt{device} \sqcap (\exists \texttt{has\_part}^*.\texttt{carcinogenic}) \tag{2}$$

Let $* = \oplus$, let $\mathcal{I}$ be an interpretation of 2 and let d be an instance of 2. Then $\mathcal{I}$ is a correct interpretation even if there is no $\texttt{has\_part}^\mathcal{I}$ chain from d to some $c \in \texttt{carcinogenic}^\mathcal{I}$: For d being an instance of 2, it is sufficient that there is some $c \in \texttt{carcinogenic}^\mathcal{I}$ with $(d, c) \in (\texttt{has\_part}^\oplus)^\mathcal{I}$.

In general, each model of an $\mathcal{ALC}_+$-concept D is also a model of its $\mathcal{ALC}_\oplus$-counterpart which is obtained by replacing each R^+ in D by $R^\oplus$. If $C' \sqsubseteq D'$ holds for two $\mathcal{ALC}_\oplus$ concepts C', D', then clearly $C \sqsubseteq D$ holds for their $\mathcal{ALC}_+$-counterparts C, D. The converse does not hold:

$$(\forall R.(A \sqcap \neg A)) \sqsubseteq (\forall R^*.(A \sqcap \neg A))$$

holds for $* = +$ (if x has no $R^\mathcal{I}$-successors, then it has clearly no $R^{+\mathcal{I}}$-successors), but it does not hold for $* = \oplus$ (x can have an $R^{\oplus\mathcal{I}}$-successor without having an $R^\mathcal{I}$-successor).

The tableau construction algorithm given in Section 4 can easily be modified to handle $\mathcal{ALC}_\oplus$-concepts: In rule R2 (b), roles $R_i^\oplus$ are handled in the same way as role names $R_i \in N_+$. For an ordinary role R_j, possible labels of j-successors of x_i are

$$\ell(x_i, (\exists R_j.C)) := \{C\} \cup L(x_i)/R_j \cup L(x_i)/R_j^\oplus \cup \{(\forall R_j^\oplus.E) \mid (\forall R_j^\oplus.E) \in L(x_i)\}$$

and the same test whether an ancestor of x_i is labelled by a superset of $\ell(x_i, (\exists R_j))$ has to be accomplished. In contrast to the trees constructed for $\mathcal{ALC}_{R+}$-concepts, the depth of trees constructed by this modified algorithm can no longer be bounded polynomially in the length of the concept. For example, if $\tilde{A}_i$ is defined as given below, each model of the concept

$$D = (\exists R.(\neg A_1 \sqcap \neg A_2 \sqcap \dots \neg A_n)) \sqcap$$
$$(\forall R^\oplus.((\exists R.\top) \sqcap (\tilde{A}_1 \sqcap \tilde{A}_2 \sqcap \dots \sqcap \tilde{A}_n)))$$

can have paths of length 2^n: It can be viewed as the representation of the binary encoding of the numbers 0 to $2^n - 1$. The concepts $\tilde{A}_i$ have to be defined in such a way that for the $k+1$-th $R^{\mathcal{I}}$-successor y of $x \in D^{\mathcal{I}}$ we have $y \in A_i^{\mathcal{I}}$ iff the i-th bit in the binary encoding of k is equal to 1. More precisely,

$$
\begin{aligned}
\tilde{A}_0 &= (A_0 \sqcap (\forall R.\neg A_0)) \sqcup (\neg A_0 \sqcap (\forall R.A_0)) \\
\tilde{A}_i &= (\sqcap_{0 \le j < i} A_j \sqcap ((A_i \sqcap \forall R.\neg A_i) \sqcup (\neg A_i \sqcap \forall R.A_i))) \sqcup \\
&\quad (\neg \sqcap_{0 \le j < i} A_j \sqcap ((A_i \sqcap \forall R.A_i) \sqcup (A_i \sqcap \forall R.\neg A_i))).
\end{aligned}
$$

The length of D is then quadratic in n whereas each model $\mathcal{I}$ of D has an $R^{\mathcal{I}}$-path of length in $O(2^n)$.

Theorem 11. *Satisfiability of $\mathcal{ALC}_{\oplus}$-concepts is* Exptime-*complete.*

Proof: Satisfiability of $\mathcal{ALC}_{\oplus}$-concepts is in Exptime because it can be decided by the modified tableau construction algorithm. It is easy to see that for an $\mathcal{ALC}_{\oplus}$-concept D, this modified algorithm creates a tree whose depth is exponentially bounded by the length of D because of the tests performed whether an ancestor is labelled by a superset of the label of new nodes.

To show that satisfiability of $\mathcal{ALC}_{\oplus}$-concepts is indeed Exptime-hard, we can modify the proof of Exptime-hardness for satisfiability in PDL given in [FL79]. The proof gives, for an alternating Turing Machine M, a PDL formula $f_M(x)$ such that $f_M(x)$ is satisfiable iff x is accepted by a simplified trace of M. The translation of this proof to $\mathcal{ALC}_{+}$-concepts is straightforward and ends with a concept D using a single role R and its transitive closure R^+. This concept is of the form

$$ D = C_1 \sqcap C_2 \sqcap (\forall R^+.C_2) $$

where R is the only role name occuring in C_1, C_2. Furthermore, R^+ occurs neither in C_1 nor in C_2. Because of its special form, D is satisfiable iff its $\mathcal{ALC}_{\oplus}$-counterpart $D' = C_1 \sqcap C_2 \sqcap (\forall R^{\oplus}.C_2)$ is satisfiable:

Each model of D is clearly a model of D'. Now let $\mathcal{I}'$ be a model of D' with $x \in D'^{\mathcal{I}'}$. Then $x \in C_1^{\mathcal{I}'} \sqcap C_2^{\mathcal{I}'}$ and for all y with $(x,y) \in R^{\oplus^{\mathcal{I}'}}$ it holds that $y \in C_2^{\mathcal{I}'}$. Let $\mathcal{I}$ be an interpretation of D which is equal to $\mathcal{I}'$ for concept and role names in C_1, C_2 and where $R^{+^{\mathcal{I}}}$ is the transitive closure of $R^{\mathcal{I}'}$. Hence we have that $(x,y) \in R^{+^{\mathcal{I}}}$ implies $(x,y) \in R^{\oplus^{\mathcal{I}'}}$ for all $x, y \in \Delta^{\mathcal{I}}$. It follows that $y \in C_2^{\mathcal{I}}$ for all $(x,y) \in R^{+^{\mathcal{I}}}$, and finally we have $x \in D^{\mathcal{I}}$. Hence $\mathcal{I}$ is a model of D. ∎

6 Conclusion

Transitive roles per se, without referring to an underlying subrole, are algorithmically easier to handle than the transitive closure of roles, whereas substituting the transitive closure of a role by some transitive superrole does not seem to make reasoning easier. Hence, when using a description logic based knowledge representation system, one should really think about whether the transitive closure of

roles is needed for this application or whether one can live with transitive roles. In the latter case, a terminological knowledge representation system based on $\mathcal{ALC}$ can be modified in such a way that it is able to handle transitive relations without severely increasing its computational complexity, but nevertheless increasing its expressive power: In the definition of concepts or description of individual objects, one can now refer to parts (ancestors, friends or relatives) at a level of decomposition (in a generation, at a degree of relationship) not known in advance. An interesting question arising from these observations is whether this holds for extensions of other concept languages as well.

References

[ACG⁺94] A. Artale, F. Cesarini, E. Grazzini, F. Pippolini, and G. Soda. Modelling composition in a terminological language environment. In *Workshop Notes of the ECAI Workshop on Parts and Wholes: Conceptual Part-Whole Relations and Formal Mereology*, pages 93–101, Amsterdam, 1994.

[Baa91] F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proc. of IJCAI-91*, 1991.

[DLNN91] F. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In *Proc. of KR-91*, Boston (USA), 1991.

[DLNN95] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. Technical Report RR-95-07, DFKI, Kaiserslautern, Deutschland, 1995.

[FL79] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *J. of Computer and System Science*, 18:194–211, 1979.

[Fra94] E. Franconi. A treatment of plurals and plural quantifications based on a theory of collections. *Minds and Machines*, 3(4):453–474, November 1994.

[HM92] 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.

[HNS90] B. Hollunder, W. Nutt, and M. Schmidt-Schauss. Subsumption algorithms for concept description languages. In *ECAI-90*, Pitman Publishing, London, 1990.

[Lad77] R.E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM J. of Computing*, 6(3):467–480, 1977.

[LB87] H. Levesque and R. J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93, 1987.

[Pri95] S. Pribbenow. Modeling physical objects: Reasoning about (different kinds of) parts. In *Time, Space, and Movement Workshop 95*, Bonas, France, 1995.

[Sch91] K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of IJCAI-91*, pages 466–471, Sydney, 1991.

[Sim87] P. M. Simons. *Parts. A study in Ontology*. Oxford: Clarendon, 1987.

[SS91] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.