Augmenting Concept Languages by Transitive Closure of Roles
An Alternative to Terminological Cycles

Franz Baader*
German Research Center for Artificial Intelligence
Projektgruppe WINO, Postfach 2080
W-6750 Kaiserslautem, Germany
baader@dfki.uni-kl.de

Abstract

In Baader (1990,1990a), we have considered different types of semantics for terminologicial cycles in the concept language T*L*Q which allows only conjunction of concepts and value-restrictions. It turned out that greatest fixed-point semantics (gfp-semantics) seems to be most appropriate for cycles in this language. In the present paper we shall show that the concept defining facilities of FL_O with cyclic definitions and gfp-semantics can also be obtained in a different way. One may replace cycles by role definitions involving union, composition, and transitive closure of roles.

This proposes a way of retaining, in an extended language, the pleasant features of gfp-semantics for FLQ with cyclic definitions without running into the troubles caused by cycles in larger languages. Starting with the language ALC of Schmidt-SchauB&Smolka (1988)—which allows negation, conjunction and disjunction of concepts as well as value-restrictions and exists-in-restrictions—we shall disallow cyclic concept definitions, but instead shall add the possibility of role definitions involving union, composition, and transitive closure of roles. In contrast to other terminological KR-systems which incorporate the transitive closure operator for roles, we shall be able to give a sound and complete algorithm for concept subsumption.

1 Introduction

In knowledge representation (KR) languages based on KL-ONE (Brachman&Schmolze (1985)), one starts with atomic concepts and roles, and can use the language formalism to define new concepts and roles. Concepts can be considered as unary predicates which are interpreted as sets of individuals whereas roles are binary predicates which are interpreted as binary relations between individuals. The languages (e.g., F*L* and F*L*- of Levesque&Brachman (1987), *TF* and NTF of Nebel (1990)) differ in what kind of constructs are allowed for the definition of concepts and roles. Their common feature—besides the use of concepts and roles—is that the meaning of the constructs is defined with the help of a model-theoretic semantics. Most of these languages do not go beyond the scope of first-order predicate logic, and they usually have very restricted formalisms for defining roles.

This work was supported by the German "Bundesministerium fur Forschung und Technologic" under Grant ITW 8903 0.

However, for many applications it would be very useful to have means for expressing things like transitive closure of roles. For example, if we have a role child (resp. is-direct-part-of) we might want to use its transitive closure offspring (resp. is-part-of) in order to define concepts like "man who has only male offsprings" (resp, "car which has only functioning parts"). Obviously, we cannot just introduce a new role offspring without enforcing the appropriate relationship between offspring and child. Since the transitive closure of binary relations cannot be expressed in first-order predicate logic (see Aho&Ullman (1979)), the concept languages mentioned above cannot be used for that purpose.

There are two possibilities to overcome this problem. On the one hand, one may introduce a new role-forming operator trans, and define its semantics such that, for any role R, Irans(R) is interpreted as the transitive closure of R. This operator is e.g. contained in the terminological representation language LOOM (MacGregor&Bates (1987)). However, LOOM does not have a complete algorithm to determine subsumption relationships between concepts.

On the other hand, cyclic concept definitions together with an appropriate fixed-point semantics can be used to express value-restrictions with respect to the transitive closure of roles (see Baader (1990,1990a)). However, cyclic definitions are prohibited in most terminological knowledge representation languages because, from a theoretical point of view, their semantics is not clear and, from a practical point of view, existing inference algorithms cannot handle cycles.

The first thorough investigation of cycles in terminological knowledge representation languages can be found in Nebel (1990,1990a). Nebel has introduced three different kinds of semantics—namely, least fixed-point semantics (lfp-semantics), greatest fixed-point semantics (gfp-semantics), and what he called descriptive semantics—for cyclic definitions in his language NTF, Baader (1990,1990a) considers terminological cycles in a small KL-ONE-based language which allows only concept conjunctions and value-restrictions. For this language, which will be called FL_O in the following, the effect of the three above mentioned types of semantics can be completely described with the help of finite automata. As a consequence, subsumption determination for each type of semantics can be reduced to a well-known decision problem for finite automata. For the language FL_o. the gfp-semantics comes off best. The characterization of this

semantics is easy and has an obvious intuitive interpretation. It also shows that gfp-semantics is the appropriate semantics for expressing value-restrictions with respect to the transitive closure of roles.

However, the results described in Baader (1990a) have two major drawbacks which we intend to overcome in the present paper. First, the language FL_0 is too small to be sufficient for practical purposes. As shown in Baader (1990), the results can be extended to the language FI *of* Levesque&Brachman (1987), and it seems to be relatively easy to include number-restrictions. However, as soon as we also consider disjunction of concepts and exists-in-restrictions (see Def. 2.1), the unpleasant features which lfp-semanucs had for FL_0 (see Baader (1990,1990a)) also occur for gfp-semantics in this larger language. If we should like to have general negation of concepts, least or greatest fixed-points may not even exist, thus rendering fixed-point semantics impossible.

Second, the characterization of gfp-semantics for *fLo*—though relatively easy and intuitive—still involves notions from formal language theory such as regular languages and finite automata. In the present paper we shall show that the concept defining facilities of FI_0 with cyclic definitions and gfp-semantics can also be obtained in a different way. One may prohibit cycles and instead allow role definitions using union, composition, and transitive closure of roles. The regular languages which occur in the characterization of gfp-semantics for FL_0 can directly be translated into role definitions in this new language.

This proposes a way of retaining, in an extended language, the pleasant features of gfp-semantics for FL_0 with cyclic definitions without running into the troubles caused by cycles in larger languages. Starting with the language *ALC* of Schmidt-SchauB&Smolka (1988)-which allows negation, conjunction and disjunction of concepts as well as value-restrictions and exists-in-restrictions—we shall disallow cyclic concept definitions, but instead shall add the possibility of role definitions involving union, composition, and transitive closure of roles. In contrast to other terminological KR-systems which incorporate the transitive closure operator for roles, we shall be able to give a sound and complete algorithm for concept subsumption.

Because of the space limitations it is not possible to include a complete formal description of this algorithm into the present paper. Instead we shall first recall by an example how the subsumption algorithm for *ALC* works. It will then be explained how the ideas underlying this algorithm can be generalized to the extended language. Two more examples will be used to demonstrate what kind of new problems may occur. A complete description of the algorithm, together with the proof of its correctness can be found in Baader (1990b).

2 The Languages A*LC* and FL_0

The language which we shall use as a starting point for the extension described in Section 4 is called "attributive concept description language with unions and complements", for short A*LC* (Schmidt-SchauB&Smolka (1988)). The reason for choosing ALC was that it is large enough to exhibit most of the problems connected with such an extension. Taking a larger language (e.g., including number restrictions) would only mean more work without bringing new insights.

Definition 2.1. (concept terms and terminologies)
Let C be a set of concept names and R be a set of role names. The set of *concept terms* of $\mathcal{ALC}$ is inductively defined. As a starting point of the induction, any element of C is a concept term (*atomic terms*). Now let C and D be concept terms already defined, and let R be a role name. Then $C \sqcap D$ (*conjunction*), $C \sqcup D$ (*disjunction*), $\neg C$ (*negation*), $\forall R{:}C$ (*value-restriction*), and $\exists R{:}C$ (*exists-in-restriction*) are concept terms.
Let A be a concept name and let D be a concept term. Then $A = D$ is a terminological axiom. A *terminology* (T-box) is a finite set of terminological axioms with the additional restriction that no concept name may appear more than once as a left hand side of a definition. ❑

The sublanguage $\mathcal{FL}_0$ of $\mathcal{ALC}$ is defined as follows: the concept defining operators are restricted to concept conjunction and value-restriction.

A T-box T contains two different kinds of concept names. *Defined concepts* occur on the left hand side of a terminological axiom of T. The other concepts are called *primitive concepts*. Cycles in terminologies are defined as follows. Let A, B be concept names and let T be a T-box. We say that A *directly uses* B in T iff B appears on the right hand side of the definition of A. Let *uses* denote the transitive closure of the relation *directly uses*. Then T contains a *terminological cycle* iff there exists a concept name A in T such that A uses A

For example, assume that we want to describe all "aliens having only melmacian ancestors on the mother's side", using the primitive roles father, mother, and the primitive concepts Alien, Melmacian. In $\mathcal{FL}_0$, this concept (for short Mam) is defined by the cyclic terminology $Mam = Alien \sqcap \forall mother{:}Ma$, $Ma = Melmacian \sqcap \forall mother{:}Ma \sqcap \forall father{:}Ma$, which introduces the auxiliary concept Ma for "aliens having only melmacian ancestors".

The next definition gives a model-theoretic semantics for the language introduced in Definition 2.1.

Definition 2.2. (interpretations and models)
An *interpretation* I consists of a set dom(I), the domain of the interpretation, and an interpretation function which associates with each concept name A a subset A^I of dom(I), and with each role name R a binary relation R^I on dom(I), i.e., a subset of dom(I) × dom(I).
The interpretation function—which gives an interpretation for atomic terms—can be extended to arbitrary terms as follows: Let C, D be concept terms and R be a role name. Assume that C^I and D^I are already defined. Then $(C \sqcap D)^I := C^I \cap D^I$, $(C \sqcup D)^I := C^I \cup D^I$, $(\neg C)^I := dom(I) \setminus C^I$, $(\forall R{:}C)^I := \{x \in dom(I);$ for all y such that $(x,y) \in R^I$ we have $y \in C^I\}$, and $(\exists R{:}C)^I := \{x \in dom(I);$ there exists y such that $(x,y) \in R^I$ and $y \in C^I\}$.
An interpretation I is a *model* of the T-box T iff it satisfies $A^I = D^I$ for all terminological axioms $A = D$ in T. ❑

An important service terminological representation systems provide is computing the *subsumption hierarchy*.

Let T be a terminology and let A, B be concept names. Then we say that B *subsumes* A with respect to T (symbolically $A \sqsubseteq_T B$) iff $A^I \subseteq B^I$ for all models I of T

3 Extensions of $\mathcal{FL}_0$

The semantics we have given in Definition 2.2 is not restricted to non-cyclic terminologies. But for cyclic terminologies this kind of semantics, which will be called *descriptive semantics* in the following, turns out to be unsatisfactory (see Baader (1990a), Example 2.3). For this reason, alternative types of semantics for terminological cycles have been considered, namely *greatest fixed-point semantics* (gfp-semantics) and *least fixed-point semantics* (lfp-semantics). Roughly speaking, gfp-semantics (lfp-semantics) means that, with respect to a given interpretation of the primitive concepts and roles, the defined concepts are interpreted as large (small) as possible in gfp-models (lfp-models) of the terminology (see Nebel (1990,1990a) or Baader (1990,1990a) for details).

In Baader (1990,1990a) it was shown that a finite automaton $\mathcal{A}_T$ can be associated to each terminology T of $\mathcal{FL}_0$. The alphabet of $\mathcal{A}_T$ is the set $\mathbf{R_T}$ of all role names occurring in T, the states of $\mathcal{A}_T$ are the concept names occurring in T, and the transitions of $\mathcal{A}_T$ are defined by the terminological axioms of T (see Baader (1990,1990a) for details). This automaton was used to describe the effect of the three above mentioned types of semantics with the help of well-known notions from automata theory. The characterization of gfp-semantics is easy, and it involves only regular languages over the alphabet of role names. *More precisely, the automaton $\mathcal{A}_T$ defines a regular* languages L(A,B) for each pair of concepts A, B occurring in T. For an interpretation I and a word $W = R_1R_2...R_n$ over $\mathbf{R_T}$, W^I denotes the composition $R_1^I \circ R_2^I \circ ... \circ R_n^I$ of the binary relations $R_1^I, R_2^I, ..., R_n^I$. For the empty word ε, ε^I denotes the identity relation, i.e., $\varepsilon^I = \{(d,d); d \in dom(I)\}$.

Theorem 3.1. Let T be a terminology of $\mathcal{FL}_0$, and let $\mathcal{A}_T$ be the corresponding automaton. Let I be a gfp-model of T, and let A be a concept name occurring in T.
For any $d \in dom(I)$ we have $d \in A^I$ iff for all primitive concepts P, all words $W \in L(A,P)$, and all individuals $e \in dom(I)$, $(d,e) \in W^I$ implies $e \in P^I$. ❑

In our example which is about aliens from Melmac, we have L(Mam,Melmacian) = (mother(father∪mother)*)

The characterization in Theorem 3.1 means that, if we start with an individual d = ALF and consider first his mother, and then arbitrarily mothers or fathers of the individuals already obtained, then ALF is an alien having only melmacian ancestors on the mother's side if and only if all the individuals reached that way come from Melmac.

Theorem 3.1 motivates the definition of regular value-restrictions in the following "regular extension" of $\mathcal{FL}_0$.

Definition 3.2. (1) Let L be a regular language over the set of role names, and let C be a concept term already defined. Then ∀L:C is a *regular value-restriction*. Its semantics is defined as $(\forall L:C)^I := \{d \in dom(I)$; for all words $W \in L$ and all individuals $e \in dom(I)$, $(d,e) \in W^I$ implies $e \in C^I\}$.
(2) In the *regular extension* $\mathcal{FL}_{reg}$ of $\mathcal{FL}_0$ we allow to use regular value-restrictions and concept conjunction as concept forming operators. ❑

Theorem 3.1 implies that, with respect to gfp-semantics, cyclic terminologies of $\mathcal{FL}_0$ can be expressed by acyclic terminologies of $\mathcal{FL}_{reg}$. On the other hand, it can be shown that any acyclic terminology of $\mathcal{FL}_{reg}$ can be expressed by a possibly cyclic terminology of $\mathcal{FL}_0$. Moreover, any cyclic terminology of $\mathcal{FL}_{reg}$ (considered with gfp-semantics) can be transformed into an equivalent acyclic terminology of $\mathcal{FL}_{reg}$ (Baader (1990b)). In our example, we can describe Mam by the following concept term of $\mathcal{FL}_{reg}$:

$$\text{Alien} \sqcap \forall(\text{mother}(\text{father} \cup \text{mother})^*):\text{Melmacian}$$

Proposition 3.3. Possibly cyclic terminologies of $\mathcal{FL}_0$ considered with gfp-semantics, acyclic terminologies of $\mathcal{FL}_{reg}$, and possibly cyclic terminologies of $\mathcal{FL}_{reg}$ considered with gfp-semantics have the same expressive power. ❑

In place of cyclic definitions or regular languages in value-restrictions, we shall now allow role terms involving union, composition and transitive closure of roles in the following "transitive extension" of $\mathcal{FL}_0$. All these role forming operators have already been considered in KL-ONE based systems, though not necessarily together. It will turn out that we get a language having exactly the same expressiveness as $\mathcal{FL}_{reg}$.

Definition 3.4. (1) Let R be a set of role names. The set of *role terms* is inductively defined as follows. As a starting point of the induction, any role name is a role term (*atomic role*), and the symbol Ø is a role term (*empty role*). Now assume that R and S are role terms already defined. Then $R \sqcup S$ (*union*), $R \circ S$ (*composition*), and trans(R) (*transitive closure*) are role terms. The semantics of the role forming operators is defined in the obvious way: $\emptyset^I := \emptyset$, $(R \sqcup S)^I := R^I \cup S^I$, $(R \circ S)^I := R^I \circ S^I$, and $(trans(R))^I := \bigcup_{n \geq 1}(R^I)^n$, i.e., $(trans(R))^I$ is the transitive closure of R^I.
(2) In the *transitive extension* $\mathcal{FL}_{trans}$ of $\mathcal{FL}_0$ we allow to use role terms instead of simply roles in value-restrictions. ❑

It is now easy to see that there is a direct correspondence between the regular languages in value-restrictions of $\mathcal{FL}_{reg}$ and the role terms in value-restrictions of $\mathcal{FL}_{trans}$. Consequently, acyclic (resp. cyclic) terminologies of $\mathcal{FL}_{trans}$ have the same expressive power as acyclic (resp. cyclic) terminologies of $\mathcal{FL}_{reg}$, and thus as possibly cyclic terminologies of $\mathcal{FL}_0$ considered with gfp-semantics.

In $\mathcal{FL}_{trans}$ we can use the following term to define the concept Mam of our example:

$$\text{Alien} \sqcap \forall\,\text{mother:Melmacian} \sqcap \forall(\text{mother} \circ \text{trans}(\text{father} \sqcup \text{mother})):\text{Melmacian}.$$

4 The Extension of $\mathcal{ALC}$

In the previous section we have seen that the expressiveness of possibly cyclic terminologies of $\mathcal{FL}_0$ considered with gfp-semantics can also be obtained without involving cyclic definitions; we just have to include the appropriate role

forming operators into the language. These role forming operators can now be integrated into the larger language $\mathcal{ALC}$ without causing any of the troubles we should have with cyclic definitions in $\mathcal{ALC}$.

Definition 4.1. (1) In the *transitive extension* $\mathcal{ALC}_{trans}$ of $\mathcal{ALC}$ we allow to use role terms (as defined in part (1) of Definition 3.4) instead of simply roles in value-restrictions and exists-in-restrictions. The semantics of $\mathcal{ALC}_{trans}$ is defined as in Definition 2.2 and 3.4.
(2) In the *regular extension* $\mathcal{ALC}_{reg}$ of $\mathcal{ALC}$ we allow to use regular value-restrictions and regular exists-in-restrictions in place of the usual restrictions of $\mathcal{ALC}$. The semantics of the regular value-restrictions is defined as in part (1) of Definition 3.2. The semantics of the regular exists-in-restrictions will be defined in a way such that $\neg(\exists L{:}C)$ is equivalent to $\forall L{:}(\neg C)$. That means that we define $(\exists L{:}C)^I := \{d \in dom(I)$; there exists a word $W \in L$ and an individual $e \in dom(I)$ such that $(d,e) \in W^I$ and $e \in C^I\}$. ❑

Please note that the regular exists-in-restriction is very similar to what is called "Functional Uncertainty" in computational linguistic (see Kaplan&Maxwell (1988)).

As for $\mathcal{FL}_{reg}$ and $\mathcal{FL}_{trans}$, acyclic terminologies of $\mathcal{ALC}_{trans}$ and acyclic terminologies of $\mathcal{ALC}_{reg}$ have the same expressive power. This shows that we may restrict our attention to one of these two languages. The definition of $\mathcal{ALC}_{trans}$ is more intuitive, and thus $\mathcal{ALC}_{trans}$ may be more appropriate if we want to apply the language to actual representation problems. But $\mathcal{ALC}_{reg}$ turns out to be more convenient for describing the subsumption algorithm. We can now state the main result of the paper.

Theorem 4.2. There exists a sound and complete algorithm for testing subsumption relationships w.r.t. acyclic terminologies of $\mathcal{ALC}_{reg}$ (or equivalently $\mathcal{ALC}_{trans}$). ❑

All the existing system which incorporate transitive closure of roles have only sound but incomplete algorithms, i.e., these algorithms may sometimes fail to detect subsumption relationships.

Since we only allow acyclic terminologies of $\mathcal{ALC}_{reg}$, subsumption with respect to terminologies can be reduced to subsumption of concept terms by expanding concept definitions (see e.g., Nebel (1990)). For two concept terms C, D we say that C is subsumed by D (symbolically $C \sqsubseteq D$) iff $C^I \subseteq D^I$ *for all interpretations* I. As for $\mathcal{ALC}$, the subsumption problem for concept terms can further be reduced to the satisfiability problem, where a concept term C is called satisfiable iff there exists an interpretation I such that $C^I \neq \emptyset$. In fact, for concept terms C, D and an interpretation I, we have $C^I \subseteq D^I$ iff $C^I \setminus D^I = \emptyset$, i.e., iff $(C \sqcap \neg D)^I = \emptyset$. This shows that C is subsumed by D iff $C \sqcap \neg D$ is unsatisfiable. Thus it is sufficient to have an algorithm which decides satisfiability of concept terms.

This algorithm will use the idea of constraint propagation, as proposed by Schmidt-Schauß&Smolka (1988) for $\mathcal{ALC}$, and successfully used by Hollunder et al. (1990) and Hollunder&Nutt (1990) for various other languages. However, an algorithm for $\mathcal{ALC}_{reg}$ has to treat regular restrictions of the form $\exists L{:}C$, $\forall L{:}C$ instead of simple restrictions $\exists R{:}C$, $\forall R{:}C$.

In order to clarify this difference, let us first recall by an example how satisfiability can be checked for concept terms of $\mathcal{ALC}$ (see Schmidt-Schauß&Smolka (1988), and Hollunder&Nutt (1990) for details).

4.1 The Satisfiability Test for $\mathcal{ALC}$

Assume that C is a concept term of $\mathcal{ALC}$ which has to be checked for satisfiability. In a first step we can push all negations as far as possible into the term using the fact that the terms $\neg\neg D$ and D, $\neg(D \sqcap E)$ and $\neg D \sqcup \neg E$, $\neg(D \sqcup E)$ and $\neg D \sqcap \neg E$, $\neg(\exists R{:}D)$ and $\forall R{:}(\neg D)$, as well as $\neg(\forall R{:}D)$ and $\exists R{:}(\neg D)$ are equivalent. We end up with a term C' in *negation normal form where negation is only applied to* concept names.

Example 4.3. Assume that we want to know whether the term $\exists R{:}A \sqcap \exists R{:}B$ is subsumed by $\exists R{:}(A \sqcap B)$. That means that we have to check whether the term $C := \exists R{:}A \sqcap \exists R{:}B \sqcap \neg(\exists R{:}(A \sqcap B))$ is unsatisfiable. The negation normal form of C is the term $C' := \exists R{:}A \sqcap \exists R{:}B \sqcap \forall R{:}(\neg A \sqcup \neg B)$.

In a second step, we try to construct a finite interpretation I such that $C'^I \neq \emptyset$. That means that there has to exist an individual in dom(I) which is an element of C'^I. Thus the algorithm generates such an individual b and imposes the constraint $b \in C'^I$ on it. In the example, this means that b has to satisfy the following constraints: $b \in (\exists R{:}A)^I$, $b \in (\exists R{:}B)^I$, and $b \in (\forall R{:}(\neg A \sqcup \neg B))^I$.

From $b \in (\exists R{:}A)^I$ we can deduce that there has to exist an individual c such that $(b,c) \in R^I$ and $c \in A^I$. Analogously, $b \in (\exists R{:}B)^I$ implies the existence of an individual d with $(b,d) \in R^I$ and $d \in B^I$. We should not assume that c = d since this would possibly impose too many constraints on the individuals newly introduced to satisfy the exists-in-restrictions on b. Thus the algorithm introduces for any exists-in-restriction a new individual as role-successor, and this individual has to satisfy the constraints expressed by the restriction.

Since b also has to satisfy the value-restriction $\forall R{:}(\neg A \sqcup \neg B)$, and c, d were introduced as R^I-successors of b, we also get the constraints $c \in (\neg A \sqcup \neg B)^I$, and $d \in (\neg A \sqcup \neg B)^I$. Now c has to satisfy the constraints $c \in A^I$ and $c \in (\neg A \sqcup \neg B)^I$ whereas d has to satisfy the constraints $d \in B^I$ and $d \in (\neg A \sqcup \neg B)^I$. Thus the algorithm uses value-restrictions in interaction with already defined role-relationships to impose *new constraints on individuals.*

Now $c \in (\neg A \sqcup \neg B)^I$ means that $c \in (\neg A)^I$ or $c \in (\neg B)^I$, and we have to choose one of these possibilities. If we assume $c \in (\neg A)^I$, this clashes with the other constraint $c \in A^I$. Thus we have to choose $c \in (\neg B)^I$. Analogously, we have to choose $d \in (\neg A)^I$ in order to satisfy the constraint $d \in (\neg A \sqcup \neg B)^I$ without creating a contradiction to $d \in B^I$. Thus, for disjunctive constraints, the algorithm tries both possibilities in successive attempts. It has to backtrack, if it reaches a contradiction, i.e., if the same individual has to satisfy complementary constraints.

In the example, we have now satisfied all the constraints without getting a contradiction. This shows that C' is satisfiable, and thus $\exists R{:}A \sqcap \exists R{:}B$ is not subsumed by

$\exists R:(A \sqcap B)$. We have generated an interpretation I as witness for this fact: $dom(I) = \{b, c, d\}$; $R^I = \{(b,c), (b,d)\}$; $A^I = \{c\}$ and $B^I = \{d\}$. For this interpretation, $b \in C'^I$. That means that $b \in (\exists R:A \sqcap \exists R:B)^I$, but $b \notin (\exists R:(A \sqcap B))^I$.

Termination of the algorithm is ensured by the fact that the newly introduced constraints are always smaller than the constraints which enforced their introduction.

4.2 The Generalization to $\mathcal{ALC}_{reg}$

A satisfiability algorithm for $\mathcal{ALC}_{reg}$ has to treat regular restrictions of the form $\exists L:C$ and $\forall L:C$ instead of simple restrictions $\exists R:C$ and $\forall R:C$. In order to satisfy a constraint of the form $b \in (\exists R:C)^I$, the algorithm described above introduces a new individual c which has to satisfy bR^Ic and $c \in C^I$. This is not so easy if we have to satisfy a regular constraint of the form $b \in (\exists L:C)^I$. All we know is that there has to exist some word $W \in L$ and an individual c such that bW^Ic and $c \in C^I$. But we do not know which W does the job, and if L is infinite, there are infinitely many canditates. Thus trying them one after another will not do.

Obviously, the concept terms $\exists L:C$ and $C \sqcup \exists(L \setminus \{\varepsilon\}):C$ are equivalent. For that reason we may without loss of generality assume that L does not contain the empty word. Thus the correct word $W \in L$ has some role symbol R as its first symbol. That means that there exists a word U such that $W = RU$. The alphabet of role symbols over which L is built is finite, and thus there are only finitely many possibilities for choosing a symbol R. Once we have chosen R, we still do not know which word U does the job. All we know about U is that it is an element of the set $R^{-1}L := \{V; RV \in L\}$.

Definition 4.4. Let L be a language and let W be a word. The left quotient $W^{-1}L$ of L with respect to W is defined as $W^{-1}L := \{ V; WV \in L \}$. ❑

For a regular language L, the language $W^{-1}L$ is also regular (see Eilenberg (1974), p. 37), and obviously, this is also true for $W^{-1}L \setminus \{\varepsilon\}$. For words V, W we have $(VW)^{-1}L = W^{-1}(V^{-1}L)$. For example, let L be the regular language $(RS)^+$. Then $R^{-1}L = S(RS)^*$, $S^{-1}L = \emptyset$, and $(RS)^{-1}L = S^{-1}(R^{-1}L) = (RS)^*$.

In the satisfiability test, we can now choose between two possibilities: $U \in R^{-1}L$ can be the empty word (provided that $R \in L$) or U can be nonempty (provided that $R^{-1}L \setminus \{\varepsilon\} \neq \emptyset$). If we assume $U = \varepsilon$, then the new individual c has to satisfy bR^Ic and $c \in C^I$, and the exists-in-restriction is worked off. If we assume $U \neq \varepsilon$, then $b(RU)^Ic$ ensures the existence of an individual d such that bR^Id, dU^Ic, and $c \in C^I$. We still do not know the appropriate U, but the existence of such a word U and an individual c with dU^Ic, and $c \in C^I$ can be expressed by the constraint $d \in (\exists(R^{-1}L \setminus \{\varepsilon\}):C)^I$.

Thus we have seen how the treatment of exists-in-restrictions in the satisfiability algorithm for $\mathcal{ALC}$ can be generalized to $\mathcal{ALC}_{reg}$. We shall now turn to value-restrictions.

Assume that we have a constraint $b \in (\forall L:C)^I$, and—to satisfy an exists-in-constraint on b—we have introduced an individual c such that bR^Ic. Obviously, if $R \in L$, we have to add the constraint $c \in C^I$; but this is not sufficient for the following reason. Assume that U is an element of $R^{-1}L \setminus \{\varepsilon\}$, i.e., U is a nonempty word such that $RU \in L$. If, in some step of the algorithm, an individual d is introduced such that cU^Id holds, then d has to satisfy the constraint $d \in C^I$. This is so because $b(RU)^Id$, $RU \in L$, and b has to satisfy $b \in (\forall L:C)^I$. We can keep track of this possibility by imposing the constraint $c \in (\forall(R^{-1}L \setminus \{\varepsilon\}):C)^I$ on c.

Unlike the situation for $\mathcal{ALC}$ one can no longer be sure of the termination of the algorithm, unless one imposes an appropriate control structure, and tests for cycles. The problem of nontermination will be demonstrated by the following example.

Example 4.5. We consider the following concept term of $\mathcal{ALC}_{reg}$: $C := A \sqcap \exists R:A \sqcap \forall R^+:(\exists R:A)$.
(1) We introduce an individual a_0 which has to satisfy the constraints $a_0 \in A^I$, $a_0 \in (\exists R:A)^I$, $a_0 \in (\forall R^+:(\exists R:A))^I$.
(2) Because of the exists-in-restriction for a_0 we introduce a new individual a_1 such that $a_0R^Ia_1$, and this individual has to satisfy the constraint $a_1 \in A^I$.
(3) Now the interaction between $a_0R^Ia_1$ and the value-restriction $a_0 \in (\forall R^+:(\exists R:A))^I$ has to be taken into account. Because of $R \in R^+$ we obtain the constraint $a_1 \in (\exists R:A)^I$. In addition, we have $R^{-1}R^+ \setminus \{\varepsilon\} = R^+ \neq \emptyset$, which yields the constraint $a_1 \in (\forall R^+:(\exists R:A))^I$. To sum up, a_1 has to satisfy the constraints $a_1 \in A^I$, $a_1 \in (\exists R:A)^I$, and $a_1 \in (\forall R^+:(\exists R:A))^I$, i.e., the same constraints as previously a_0. If we continue with the constraints on a_1 we get an individual a_2 which, in the end, has to satisfy the same constraints as a_1. This yields an individual a_3, and so on. In other words, the algorithm has run into a cycle.
On the other hand, we could just identify a_0 with a_1. This *would yield the following interpretation J:* $dom(J) := \{a_0\}$; $R^J := \{ (a_0,a_0) \}$; $A^J := \{a_0\}$. It is easy to see that this interpretation satisfies $a_0 \in C^J$. ❑

The phenomenon that such cycles may occur is not particular for this example. After sufficiently long computation, the algorithm will always reproduce sets of constraints which have already been considered. Basically, this is a consequence of the following fact, which in turn is an easy consequence of the quotient criterion for regular languages (see Eilenberg (1974), Theorem 8.1).

Proposition 4.6. Let **K** be a finite set of regular languages. Then the set $\{W^{-1}L \setminus \{\varepsilon\}$; where $L \in \mathbf{K}$ and W is a word$\}$ is also finite. ❑

However, it turns out that there are two different types of cycles: "good cycles" and "bad cycles". The cycle of Example 4.5 is a "good cycle"; its occurrence indicated that the concept term under consideration is in fact satisfiable. The following example will demonstrate how "bad cycles" may arise.

Example 4.7. We consider the following concept term of $\mathcal{ALC}_{reg}$: $D := \neg A \sqcap \exists R^+:A \sqcap \forall R^+:(\neg A)$.
(1) We introduce an individual a_0 which has to satisfy $a_0 \in (\neg A)^I$, $a_0 \in (\exists R^+:A)^I$, and $a_0 \in (\forall R^+:(\neg A))^I$.
(2) Because of the exists-in-restriction for a_0 we introduce a new individual a_1 such that $a_0R^Ia_1$. But now we have $R \in R^+$ as well as $R^{-1}R^+ \setminus \{\varepsilon\} = R^+ \neq \emptyset$. Thus we have to choose between two possibilities for the constraint on a_1.

(3) First, we may consider the constraint $a_1 \in A^I$ (corresponding to the case $U = \varepsilon$ from above). But $a_0 R^I a_1$ together with the value-restriction $a_0 \in (\forall R^+:(\neg A))^I$ yields $a_1 \in (\neg A)^I$, and we have a clash with $a_1 \in A^I$.
(4) Thus we have to backtrack and choose the constraint $a_1 \in (\exists R^+:A)^I$ (corresponding to the case $U \neq \varepsilon$ from above). As before, $a_0 R^I a_1$ together with the value-restriction on a_0 yields $a_1 \in (\neg A)^I$ and $a_1 \in (\forall R^+:(\neg A))^I$. Thus a_1 has to satisfy the same constraints as previously a_0. This shows that we have again run into a cycle; but this time the situation is different. In fact, it is easy to see that the concept term D is unsatisfiable whereas the term C of Example 4.5 was satisfiable. ❑

We may now ask what makes the difference between the cycle of Example 4.5 and that of Example 4.7. In the second example we have postponed satisfying the exists-in-restriction for a_o by introducing the new exists-in-restriction for a_1. It is easy to see that we should have to postpone satisfying the restriction forever because trying to actually satisfy it will always result in a clash. In the first example however, we had already satisfied the exists-in-restriction before the cycle occurred

Building up on these ideas, an algorithm for deciding satisfiability of concept terms of ALC_{reg} is presented, and proved to be sound, complete, and terminating in Baader (1990b). The algorithm uses so-called concept trees (which are similar to AND/OR search trees) to impose an appropriate control on the search for a finite model. This makes it possible to detect cycles at the right moment, and to distinguish between good ones and bad ones. The termination proof mainly depends on Proposition 4.6, that is, on a result from formal language theory.

5 Conclusion

Augmenting *ALC by* a transitive closure operator for roles means not just adding yet another construct to this languages, and thus getting a language and an algorithm which are only slightly different from those previously considered. The transitive closure is of a rather different quality.

This claim is substantiated by the following observations. Firstly, by adding transitive closure, we are leaving the realm of first order logic. Secondly, the algorithm depends on new methods, namely on the use of results from formal language theory, and on a more sophisticated data structure to cope with the nontermination problem. Thirdly, adding features (i.e., functional roles) and feature agreements would make the subsumption problem undecidable (Baader et al. (1991)), whereas this was never a problem for the languages considered by Hollunder&Nutt (1990).

Finally, the expressiveness of ALC_{reg} is also demonstrated by the fact that concept terms of this language can be used to simulate general concept equations, i.e., equations of the form C = D where both C and D may be complex concept terms of *ALC OR* ALC_{reg} (see Baader (1990b), Section 6, Baader et al. (1991)). For this reason, the algorithm for ALC_{reg} can be used to decide satisfiability and subsumption of concepts with respect to finite sets of concept equations. As a special case, one thus gets algorithms for satisfiability and subsumption of concepts with respect to cyclic T-boxes of ALC, provided that these T-boxes are interpreted with descriptive semantics.

References

Aho, A.V., and Uliman, J.D. 1979. Universality of Data Retrieval Languages. In Proceedings of the 6th ACM Symposium on Principles of Programming Languages, 110-120.

Baader, F. 1990. Terminological Cycles in KL-ONE-based KR-languages. Research Report, RR-90-01, DFKI, Kaiserslautem.

Baader, F. 1990a. Terminological Cycles in KL-ONE-based Knowledge Representation Languages. Proceedings of the *8th National Conference on Artificial Intelligence,* AAAI-90.

Baader, F. 1990b. Augmenting Concept Languages by Transitive Closure of Roles; An Alternative to Terminological Cycles. DFKI Research Report RR-90-13, DFKI, Kaiserslautem.

Baader, F., Bilrckert, H.-J., Nebel, B., Nutt, W. Smolka, G. 1991. On the Expressivity of Feature Logics with Negation, Functional Uncertainty, and Sort Equations. Research Report, RR-91-01, DFKI, Kaiserslautem.

Brachman, RJ., and Schmoize, J.G. 1985. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science* 16: 171-216.

Eilenberg, S. 1974. *Automata, Languages and Machines, Vol A.* New York/London; Academic Press.

Hollunder, B., Nutt, W., Schmidt-SchauB, M. (1990). Subsumption Algorithms for Concept Languages. Proceedings of the 9th *European Conference on Artificial Intelligence.* ECAI-90.

Hollunder, B., Nutt, W. 1990, Subsumption Algorithms for Concept Languages, Research Report RR-90-04, DFKI, Kaiserslautem.

Kaplan, R.M., Maxwell III, J. T. 1988. An Algorithm for Functional Uncertainty. Proceedings of the COLIN 88.

Levesque, HJ., and Brachman, RJ. 1987. Expressiveness and Tractability in Knowledge Representation and Reasoning, *Computational Intelligence* 3: 78-93.

MacGregor, R., and Bates, R. 1987. The Loom Knowledge Representation Language. Technical Report IS1/RS-87-188, Information Science Institute, Univ. of Southern California.

Nebel, B. 1990. *Reasoning and Revision in Hybrid Representation Systems.* Lecture Notes in Artificial Intelligence, Subseries of Lecture Notes in Computer Science 422.

Nebel, B. 1990a. Terminological Cycles; Semantics and Computational Properties. To appear in Sowa, J. ed. 1990. *Formal Aspects of Semantic Networks.*

Schmidt-SchauB, M.$_t$ Smolka, G, 1988. Attributive Concept Descriptions with Unions and Complements. SEKI Report SR-88-21. To appear in Artificial Intelligence.