Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research
Report
RR-90-13

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

Franz Baader

December 1990

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
D-6750 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11, FRG
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern und Saarbrücken is a non-profit organization which was founded in 1988 by the shareholder companies ADV/Orga, AEG, IBM, Insiders, Fraunhofer Gesellschaft, GMD, Krupp-Atlas, Mannesmann-Kienzle, Nixdorf, Philips and Siemens. Research projects conducted at the DFKI are funded by the German Ministry for Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct *systems with technical knowledge and common sense* which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- ❏ Intelligent Engineering Systems
- ❏ Intelligent User Interfaces
- ❏ Intelligent Communication Networks
- ❏ Intelligent Cooperative Systems.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Prof. Dr. Gerhard Barth
Director

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

Franz Baader

DFKI-RR-90-13

Augmenting Concept Languages by Transitive Closure of Roles: An Alternative tBaader

German Research Center for Artificial Intelligence
Postfach 2080
W-6750 Kaiserslautern, Germany

Abstract

In Baader (1990a,1990b), we have considered different types of semantics for terminologicial cycles in the concept language $\mathcal{FL}_0$ 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 $\mathcal{FL}_0$ 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 $\mathcal{FL}_0$ with cyclic definitions without running into the troubles caused by cycles in larger languages. Starting with the language $\mathcal{ALC}$ of Schmidt-Schauß&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.

Surprisingly, this algorithm can also be used to decide subsumption with respect to concept equations, i.e., arbitrary equational axioms of the form C = D where C and D are concept terms. This is so because concept terms of our extended language can be used to encode finite sets of concept equations.

Table of Contents

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., $\mathcal{FL}$ and $\mathcal{FL}^-$ of Levesque& Brachman (1987), $\mathcal{TF}$ and $\mathcal{NTF}$ of Nebel (1990a), or the $\mathcal{AL}$-languages considered in Donini et al. (1991)) 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.

However, for some 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 above mentioned languages 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, trans(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. This seems to be a severe drawback because computing subsumption relationships is one of the major reasoning steps in KL-ONE-based KR-systems.

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 (1990a,b)). However, cyclic definitions are prohibited in most terminological knowledge representation languages (e.g., in KRYPTON (Brachman et al. (1985)), NIKL (Kaczmarek et al. (1986)) or LOOM (MacGregor&Bates (1987))) because, from a theoretical point of view, their semantics is not clear and, from a practical point of view, existing inference algorithms may go astray in the presence of cycles.

The first thorough investigation of cycles in terminological knowledge representation languages can be found in Nebel (1987,1990a,1990b). Nebel considered 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 $\mathcal{NTF}$. But, due to the fact that this language is relatively strong[1], he does not

[1] The language allows concept and role conjunction, value restrictions, number restrictions and negation of primitive concepts.

provide a deep insight into the meaning of cycles with respect to these three types of semantics.

Baader (1990a,b) considers terminological cycles in a very small KL-ONE-based language which allows only concept conjunctions and value restrictions. For this language, which will be called $\mathcal{FL}_0$ 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 (more or less) well-known decision problem for finite automata. For the language $\mathcal{FL}_0$, the gfp-semantics comes off best. The characterization of this semantics is easy and has an obvious intuitive interpretation. It involves only regular languages over the alphabet of role names, and for that reason subsumption can be reduced to inclusion of regular languages. This characterization also shows that gfp-semantics is the appropriate semantics for expressing value restrictions with respect to the transitive closure of roles.

However, the results obtained in Baader (1990a) have two major drawbacks which we intend to overcome in the present paper. First, the language $\mathcal{FL}_0$ is too small to be sufficient for practical purposes. As shown in Baader (1990b), the results can be extended to the language $\mathcal{FL}^-$ 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[2], the unpleasant features which lfp-semantics had for $\mathcal{FL}_0$ (see Baader (1990a,b)) 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 $\mathcal{FL}_0$ – though relatively easy and intuitive – still involves concepts from formal language theory such as regular languages and finite automata, that is, concepts which may not be very familiar in the area of knowledge representation. In the present paper we shall show that the concept defining facilities of $\mathcal{FL}_0$ with cyclic definitions and gfp-semantics can also be obtained in a different way. One may prohibit cycles and instead allow role definitions involving union, composition, and transitive closure of roles. The regular languages which occur in the characterization of gfp-semantics for $\mathcal{FL}_0$ 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 $\mathcal{FL}_0$ with cyclic definitions without running into the troubles caused by cycles in larger languages. Starting with the language $\mathcal{ALC}$ of Schmidt-Schauß&Smolka (1988) – which allows negation, conjunction and disjunction of concepts as well as value restrictions and exists-in restrictions – we shall not allow cyclic concept definitions, but instead we shall add the possibility of role definitions involving union, composition, and transitive closure of roles. This yields the "transitive extension" $\mathcal{ALC}_{\text{trans}}$ of $\mathcal{ALC}$. 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. The connection between role definitions involving union, composition, and transitive closure of

[2]See Definitions 2.1 and 2.2 below. This construct is called "c-some" in Nebel (1990a), and "unrestricted existential quantification" in Donini et al. (1991).

roles on the one hand, and regular languages over the alphabet of all role names on the other hand will be important for this algorithm. In particular, the quotient criterion for regular languages (see Eilenberg (1974), Theorem 8.1) will be crucial for its termination.

In Section 2 we shall recall syntax and semantics of the language $\mathcal{ALC}$, and the characterization of gfp-semantics for the sublanguage $\mathcal{FL}_0$ given in Baader (1990a,b). This section will also contain the alternative characterization of gfp-semantics for $\mathcal{FL}_0$ with the help of role-forming operators. This characterization motivates the definition of the extension of $\mathcal{ALC}$ given in the next section. In Section 3 we shall also recall by an example how a subsumption algorithm for $\mathcal{ALC}$ works. It will then be shown how the ideas underlying this algorithm may be generalized to our extension $\mathcal{ALC}_{\text{trans}}$ of $\mathcal{ALC}$, and what new problems may appear. Section 4 describes the algorithm for the extended language, and contains the proof of its completeness and soundness.

In Section 5 we shall show that the same algorithm can also be used to check subsumption with respect to a restricted semantics which allows only finite models without cyclic role chains. Considering this kind of semantics is motivated by the fact that usually, for languages without cyclic definitions or role definitions using the transitive closure operator, the existence of a model already implies the existence of a finite model without cyclic role chains. In addition, for some applications (e.g., if we want to express something like lists of arbitrary finite length) this semantics is more appropriate.

In Section 6 it will be shown that concept terms of $\mathcal{ALC}_{\text{trans}}$ (as introduced in Section 3) can be used to encode concept equations, i.e., arbitrary equational axioms of the form C = D where C and D are concept terms (of $\mathcal{ALC}$ or $\mathcal{ALC}_{\text{trans}}$). To be more precise, it will be shown how a finite set $\mathcal{E} = \{C_1 = D_1, ..., C_n = D_n\}$ of concept equations can be transformed into a concept term C of $\mathcal{ALC}_{\text{trans}}$ such that for all concept terms D we have: D is consistent w.r.t. $\mathcal{E}$ iff $C \sqcap D$ is consistent. This process of encoding explicit axioms into concept terms – and thus making them only implicitly available – will be called “internalization” of concept equations.[3] As a consequence of this internalization the algorithm developed in Section 4 can also be used to decide consistency and subsumption w.r.t. concept equations in $\mathcal{ALC}_{\text{trans}}$, and thus also in $\mathcal{ALC}$. Since cyclic terminologies of $\mathcal{ALC}$ are just sets of concept equations of a very specific form, we thus also get a solution of the consistency and the subsumption problem for terminological cycles in $\mathcal{ALC}$, provided that descriptive semantics is used.

2. KL-ONE-based KR-languages

The language which we shall use as a starting point for the extension described in Section 3 is called “attributive concept description language with unions and complements”, for short $\mathcal{ALC}$ (Schmidt-Schauß&Smolka (1988)). The reason for choosing $\mathcal{ALC}$ 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. The sublanguage of $\mathcal{ALC}$ for which cyclic definitions where considered in Baader (1990a,b) was called $\mathcal{FL}_0$ in Baader (1990b).

[3]This name is due to G. Smolka; see Baader et al. (1991).

2.1 The Languages $\mathcal{ALC}$ and $\mathcal{FL}_0$

The next definition describes the syntax of the language $\mathcal{ALC}$.

Definition 2.1. (concept terms and terminologies of $\mathcal{ALC}$)
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,
(1) 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.
(2) Then C ⊓ D, C ⊔ D, and ¬C are concept terms.

(conjunction, disjunction, and negation of concepts)

(3) Then ∀R:C and ∃R:C are concept terms.

(value restriction and exists-in restriction)

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: part (2) of Definition 2.1 is restricted to concept conjunction and part (3) to value restriction.

A T-box contains two different kinds of concept names. *Defined concepts* occur on the left hand side of a terminological axiom. The other concepts are called *primitive concepts* [4]. The following is an example of a T-box in the $\mathcal{ALC}$-formalism. Let Man, Human, Male, Father and Mos (for "*m*an who has *o*nly *s*ons") be concept names and let child be a role name. The T-box consists of the following axioms:

$$\begin{aligned} \text{Man} &= \text{Human} \sqcap \text{Male} \\ \text{Mos} &= \text{Man} \sqcap \forall \text{child: Man} \\ \text{Father} &= \text{Man} \sqcap \exists \text{child: Human} \end{aligned}$$

That means that a man is human and male. A man who has only sons is a man such that all his children are male humans. A father is a man who has at least one human child. The first two axioms are axioms of $\mathcal{FL}_0$. Male and Human are primitive concepts while Man, Mos and Father are defined concepts. Assume that we want to express a concept "*m*an who has *o*nly *m*ale *o*ffsprings", for short Momo. We cannot just introduce a new role name offspring because there would be no connection between the two primitive roles child and offspring. But the intended meaning of offspring is that it is the transitive closure of child. It seems quite natural to use a cyclic definition for Momo: A man who has only male off-springs is himself a man and all his children are men who have only male off-springs, i.e.,

$$\text{Momo} = \text{Man} \sqcap \forall \text{child: Momo}.$$

This is a very simple cyclic definition. In general, 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

[4]For the language $\mathcal{ALC}$, roles are always primitive since it does not have role definitions.

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.

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 sets A^I, R^I are called extensions of A, R with respect to 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

$$(C \sqcap D)^I := C^I \cap D^I, \ (C \sqcup D)^I := C^I \cup D^I, \text{ and } (\neg C)^I := \mathrm{dom}(I) \setminus C^I,$$

$$(\forall R{:}C)^I := \{x \in \mathrm{dom}(I); \text{ for all } y \text{ such that } (x,y) \in R^I \text{ we have } y \in C^I\},$$

$$(\exists R{:}C)^I := \{x \in \mathrm{dom}(I); \text{ there exists } y \text{ such that } (x,y) \in R^I \text{ and } y \in C^I\}.$$

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 most terminological representation systems provide is computing the subsumption hierarchy.

Definition 2.3. (subsumption of concepts)
Let T be a T-box and let A, B be concept names.

$$A \sqsubseteq_T B \quad \text{iff} \quad A^I \subseteq B^I \quad \text{for all models I of T}.$$

In this case we say that B *subsumes* A in T.

Many of the existing subsumption algorithms for terminological KR-languages (see e.g. Levesque&Brachman (1987), Schmidt-Schauß&Smolka (1988), Nebel (1990a), or Hollunder et al. (1990)) do not work on T-boxes but on concept terms. For that reason they have to "unfold" the T-box. Unfolding of a T-box means substituting defined concepts which occur on the right hand side of a definition by their defining terms. This process has to be iterated until there remain only primitive concepts on the right hand sides of the definitions. Obviously, this procedure terminates if and only if the terminology is acyclic. This was one more reason for prohibiting cyclic definitions. We shall say that a terminology is *unfolded* iff the right hand sides of its axioms only contain primitive concepts. Please note that the size of the unfolded T-box may be exponential in the size of the original T-box (see Nebel (1990c)).

In the example, the unfolded definition of Mos would be

$$\mathsf{Mos} \ = \ \mathsf{Human} \sqcap \mathsf{Male} \sqcap \forall \mathsf{child}{:}\ (\mathsf{Human} \sqcap \mathsf{Male}),$$

whereas unfolding of the definition of Momo would not terminate.

Let T be an acyclic terminology, and let A, B be defined concepts of T. Let A = C and B = D be the corresponding unfolded definitions of A, B. Then we have

$$A \sqsubseteq_T B \text{ iff } C^I \subseteq D^I \text{ for all interpretations I},$$

which shows that subsumption with respect to acyclic terminologies can be reduced to subsumption of concept terms. For concept terms C, D, the subsumption relation is defined as $C \sqsubseteq D$ iff $C^I \subseteq D^I$ for all interpretations I.

The semantics we have given in Definition 2.2[5] is not restricted to non-cyclic terminologies. But for cyclic terminologies this kind of semantics may seem unsatisfactory. One might think that the extension of a defined concept should be completely determined by the extensions of the primitive concepts and roles. This is the case for non-cyclic terminologies. More precisely, let T be a T-box containing the primitive concepts $P_1, ..., P_n$ and the roles $R_1, ..., R_m$. If T does not contain cycles, then any interpretation $P_1^I, ..., P_n^I, R_1^I, ..., R_m^I$ of the primitive concepts and roles can uniquely be extended to a model of T. If T contains cycles, a given interpretation of all primitive concepts and roles may have different extensions to models of T. This phenomenon already occurs in the Momo-example from above; with the consequence that our definition of the concept Momo is not correct if we use descriptive semantics (see Baader (1990b), Example 2.3).

For these reasons, alternative types of semantics for terminological cycles in $\mathcal{FL}_0$ have been considered in Baader (1990a,b), 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 (1990a) or Baader (1990a,b) for details). Please note that, for cycle-free terminologies, lfp-, gfp- and descriptive semantics coincide

Subsumption with respect to gfp-semantics (lfp-semantics) is defined in the obvious way, namely, $A \sqsubseteq_{gfp,T} B$ ($A \sqsubseteq_{lfp,T} B$) iff $A^I \subseteq B^I$ for all gfp-models (lfp-models) I of T.

2.2 Characterization of gfp-Semantics for $\mathcal{FL}_0$ using Regular Languages

Before we can associate a finite automaton $\mathcal{A}_T$ to a terminology T of $\mathcal{FL}_0$ we must transform T into some kind of normal form. It is easy to see that the concept terms $\forall R:(B \sqcap C)$ and $(\forall R:B) \sqcap (\forall R:C)$ are equivalent. Hence any concept term of $\mathcal{FL}_0$ can be transformed into a finite conjunction of terms of the form $\forall R_1:\forall R_2:...\forall R_n:A$, where A is a concept name. We shall abbreviate the prefix "$\forall R_1:\forall R_2:...\forall R_n$" by "$\forall W$" where $W = R_1R_2...R_n$ is a word over $\mathbf{R_T}$, the set of role names occurring in T. In the case n = 0 we also write "$\forall \varepsilon$:A"[6] instead of simply "A". For an interpretation I and a word $W = R_1R_2...R_n$, W^I denotes the composition $R_1^I \circ R_2^I \circ ... \circ R_n^I$ of the binary relations $R_1^I, R_2^I, ..., R_n^I$. The term ε^I denotes the identity relation, i.e., $\varepsilon^I = \{(d,d); d \in \text{dom}(I)\}$.

Let T be a terminology of $\mathcal{FL}_0$ where all terms are normalized as described above.

Definition 2.4. The generalized (nondeterministic) automaton $\mathcal{A}_T$ is defined as follows: 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

[5]This semantics will be called "descriptive semantics" in the following.
[6]"ε" denotes the empty word.

concept names occurring in T; a terminological axiom of the form $A = \forall W_1{:}A_1 \sqcap \ldots \sqcap \forall W_k{:}A_k$ gives rise to k transitions, where the transition from A to A_i is labeled by the word W_i.

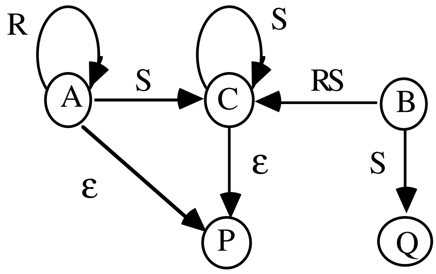
The automaton $\mathcal{A}_T$ is called "generalized" because transitions are labeled by words over the alphabet and not only by symbols of the alphabet. However, it is well-known that any generalized finite automaton can be transformed into an equivalent finite automaton (see Manna (1974), p. 9). Definition 4.1 will now be illustrated by an example.

Example 2.5. (A normalized terminology and the corresponding automaton)

$A = \forall R{:}\, A \sqcap \forall S{:}\, C \sqcap P$

$B = \forall RS{:}\, C \sqcap \forall S{:}\, Q$

$C = \forall S{:}\, C \sqcap P$



A pair of states p, q of an automaton defines a regular language L(p,q), namely the set of all words which are labels of paths from p to q. In the example, $L(A,P) = R^*S^* = \{R^nS^m;\ n,m \geq 0\}$, $L(B,P) = RSS^* = \{RSS^m;\ m \geq 0\}$, $L(C,P) = S^* = \{S^m;\ m \geq 0\}$, $L(A,Q) = L(C,Q) = \emptyset$, and $L(B,Q) = S = \{S\}$.

Above, we have already used the fact that regular languages can be described by regular expression. The *set of all regular expressions* over a finite alphabet is inductively defined:
(1) $\emptyset$ and ε are a regular expressions denoting the empty set and the set $\{\varepsilon\}$, respectively.
(2) Every symbol S of the alphabet is a regular expression and denotes the singleton $\{S\}$.
(3) If h, k are regular expressions denoting the languages H, K, respectively, then hk, $h \cup k$, and h^* are regular expressions denoting the languages $HK = \{UV;\ U \in H, V \in K\}$, $H \cup K$, and $H^* = \bigcup_{n \geq 0} H^n = \{U_1 \ldots U_n;\ n \geq 0 \text{ and } U_i \in H \text{ for } 1 \leq i \leq n\}$, respectively.
In the following, we shall not distinguish between a regular expression and the language it describes.

We are now ready to recall the characterization of the gfp-semantics given in Baader (1990a,b).

Theorem 2.6. 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, B be concept names occurring in T.
(1) 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 $e \in P^I$.
(2) Subsumption in T can be reduced to inclusion of regular languages defined by $\mathcal{A}_T$. More precisely, $A \sqsubseteq_{gfp,T} B$ iff $L(B,P) \subseteq L(A,P)$ for all primitive concepts P.

The theorem can intuitively be understood as follows: The language L(A,P) stands for the possibly infinite number of constraints of the form $\forall W{:}\, P$ which the terminology imposes on A. The more constraints are imposed the smaller the concept is. In the example, C subsumes A w.r.t. gfp-semantics since $L(C,P) = \{S^m;\ m \geq 0\}$ is a subset of $L(A,P) = \{R^nS^m;\ n,m \geq 0\}$, and $L(A,Q) = L(C,Q)$. Part (1) of Theorem 2.6 motivates the definition of regular value restrictions[7].

Definition 2.7. (A "regular extension" of $\mathcal{FL}_0$)
(1) Let L be a regular language over a give finite set of role names, and let C be a concept term already defined. Then $\forall$L:C is a *regular value restriction*. Its semantics is defined as $(\forall L{:}C)^I := \{d \in \mathrm{dom}(I);$ for all words $W \in L$ and all $e \in \mathrm{dom}(I)$, $(d,e) \in W^I$ 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.

Part (1) of Theorem 2.6 implies that, with respect to gfp-semantics, cyclic terminologies of $\mathcal{FL}_0$ can be expressed by unfolded – and thus necessarily acyclic – terminologies of $\mathcal{FL}_{reg}$. The cyclic T-box of Example 2.5 corresponds to the following unfolded T-box of $\mathcal{FL}_{reg}$:

$$\begin{aligned} A &= \forall R^*S^*{:}P \\ B &= \forall RSS^*{:}P \sqcap \forall S{:}Q \\ C &= \forall S^*{:}P \end{aligned}$$

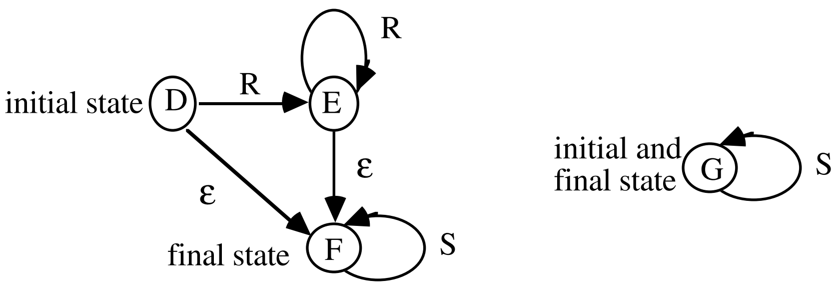
On the other hand, it is easy to see that any unfolded terminology of $\mathcal{FL}_{reg}$ can be expressed by a possibly cyclic terminology of $\mathcal{FL}_0$. We shall demonstrate this by an example. Consider the following unfolded T-box of $\mathcal{FL}_{reg}$.

$$A = \forall RR^*{:}\forall S^*{:}P \sqcap \forall S^*{:}(P \sqcap Q)$$

In the first step we use the fact that the concept terms $\forall\varepsilon$:B and B, $\forall$L:(B $\sqcap$ C) and ($\forall$L:B) $\sqcap$ ($\forall$L:C), $\forall$K$\forall$L:B and $\forall$KL:B, as well as ($\forall$K:B) $\sqcap$ ($\forall$L:B) and $\forall$(K $\cup$ L):B are equivalent. This enables us to transform the terminology such that the right hand sides are finite conjunctions of terms of the form $\forall$L:P where the primitive concept P occurs only once in the conjunction. In the example, we get

$$A = \forall(RR^*S^* \cup S^*){:}P \sqcap \forall S^*{:}Q$$

We may now interpret the finite automata for the languages occurring in the regular value restrictions as parts of a possibly cyclic terminology of $\mathcal{FL}_0$. In the example, we have the following two automata for the languages $RR^*S^* \cup S^*$ and S^* occurring in the definition of A:



Thus the above definition of A involving regular value restrictions can be transformed into the following definitions in $\mathcal{FL}_0$.

[7]This method of extending a give formalism with the help of regular languages is also used in other areas; see e.g., the functional descriptions that contain regular uncertainties which are described in Kaplan&Maxwell (1988), or the extended temporal logic ETL_f in Thayse (1989), Section 4.2.12.

$$
\begin{aligned}
A &= D \sqcap G \\
D &= \forall R{:}E \sqcap F \\
E &= \forall R{:}E \sqcap F \\
F &= \forall S{:}F \sqcap P \\
G &= \forall S{:}G \sqcap Q
\end{aligned}
$$

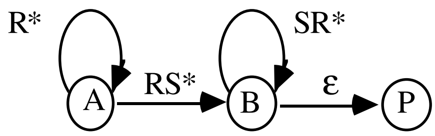
Part (1) of Theorem 2.6 ensures that the finally obtained T-box of $\mathcal{FL}_0$, if considered with gfp-semantics, really expresses the original T-box of $\mathcal{FL}_{\mathrm{reg}}$. We have thus established a 1–1-correspondence between possibly cyclic terminology of $\mathcal{FL}_0$ and unfolded terminology of $\mathcal{FL}_{\mathrm{reg}}$.

As for $\mathcal{ALC}$ and $\mathcal{FL}_0$, any acyclic terminology of $\mathcal{FL}_{\mathrm{reg}}$ can be transformed into an equivalent unfolded terminology of $\mathcal{FL}_{\mathrm{reg}}$. However, unlike the situation for $\mathcal{FL}_0$, unfolding is also possible for cyclic T-boxes, if they are interpreted with gfp-semantics. We have seen in Definition 2.4 above how to associate a generalized finite automaton $\mathcal{A}_T$ to a terminology T of $\mathcal{FL}_0$. In the same way we may associate a so-called Reg-graph $\mathcal{G}_T$ to a terminology T of $\mathcal{FL}_{\mathrm{reg}}$. A Reg-graph is an even more generalized automaton where transitions may be labeled by regular languages. Nevertheless, these Reg-graphs are just accepting regular languages (see Manna (1974), p. 11; Manna calls Reg-graphs "generalized transition graphs").

Example 2.8. (a cyclic terminology of $\mathcal{FL}_{\mathrm{reg}}$ and the corresponding Reg-graph)

$A = \forall R^*{:}\ A \sqcap \forall RS^*{:}\ B$

$B = \forall SR^*{:}\ B \sqcap P$



We have $L(B,P) = (SR^*)^*$, and $L(A,P) = (R^*)^*(RS^*)(SR^*)^*$.

In the same way as for $\mathcal{FL}_0$, Theorem 2.6 can now be proved for $\mathcal{FL}_{\mathrm{reg}}$, with $\mathcal{G}_T$ in place of $\mathcal{A}_T$. Part (1) of the theorem yields the representation by an unfolded T-box of $\mathcal{FL}_{\mathrm{reg}}$. For the example we obtain the following unfolded T-box:

$$
\begin{aligned}
A &= \forall (R^*)^*(RS^*)(SR^*)^*{:}P \\
B &= \forall (SR^*)^*{:}P
\end{aligned}
$$

Theorem 2.9. Possibly cyclic terminologies of $\mathcal{FL}_0$ considered with gfp-semantics, unfolded terminologies of $\mathcal{FL}_{\mathrm{reg}}$, acyclic terminologies of $\mathcal{FL}_{\mathrm{reg}}$, and possibly cyclic terminologies of $\mathcal{FL}_{\mathrm{reg}}$ considered with gfp-semantics have the same expressive power.

2.3 An Alternative Characterization of gfp-Semantics for $\mathcal{FL}_0$ using Role Terms

In addition to the concept forming operators of $\mathcal{FL}_0$, we shall now consider role forming operators.

Definition 2.10. (A "transitive extension" of $\mathcal{FL}_0$)
(1) Let **R** be a finite 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 of roles), $R \circ S$ (composition of roles), and trans(R) (transitive closure of a role) are role terms.
(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.

The semantics of the role forming operators is defined in the obvious way.

Definition 2.11. Let I be an interpretation. The interpretation function – which gives an interpretation for atomic roles – can be extended to arbitrary role terms as follows: Let R, S be role terms, and assume that R^I and S^I are already defined. Then

$Ø^I := Ø$, $(R \sqcup S)^I := R^I \cup S^I$, $(R \circ S)^I := R^I \circ S^I$, and

$(\text{trans}(R))^I := \bigcup_{n \geq 1} (R^I)^n$, i.e., $(\text{trans}(R))^I$ is the transitive closure of R^I.

In the following we want to demonstrate that acyclic terminologies of $\mathcal{FL}_{trans}$ have the same expressive power as possibly cyclic terminologies of $\mathcal{FL}_0$ considered with gfp-semantics. By Theorem 2.9, we may consider acyclic terminologies of $\mathcal{FL}_{reg}$ instead of possibly cyclic terminologies of $\mathcal{FL}_0$.

In order to get a direct correspondence between the regular languages in value restrictions of $\mathcal{FL}_{reg}$ and the role terms in value restrictions of $\mathcal{FL}_{trans}$ it is convenient to restrict the regular value restrictions of $\mathcal{FL}_{reg}$ to regular languages not containing the empty word ε[8]. This can be done without loss of expressive power. In fact, if L is a regular language containing ε, then $L \setminus \{\varepsilon\}$ is also regular, and the concept terms $\forall L{:}C$ and $C \sqcap \forall(L \setminus \{\varepsilon\}){:}C$ are equivalent.

It is easy to see that regular languages not containing the empty word can be described by so-called positive regular expressions where ε is not used, and where the plus operator is used instead of the star operator. If h is a positive regular expression denoting the ε-free language H, then h^+ denotes the language $H^+ := HH^* = \bigcup_{n \geq 1} H^n$. For example, the ε-free regular language R^*SR^* can be denoted by the positive regular expression $R^+SR^+ \cup R^+S \cup SR^+ \cup S$.

A role term R can now be translated into a positive regular expression $\psi(R)$ by replacing the empty role by the empty language, union of roles by union of languages, composition of roles by concatenation of languages, and transitive closure of roles by the operation plus on languages. Obviously, ψ is a bijection between role terms and positive regular expressions, and thus we also have the mapping ψ^{-1} which translates positive regular expressions back into role terms. For the example from above we have $\psi^{-1}(R^+SR^+ \cup R^+S \cup SR^+ \cup S) = \text{trans}(R) \circ S \circ \text{trans}(R) \sqcup \text{trans}(R) \circ S \sqcup S \circ \text{trans}(R) \sqcup S$.

A concept term C of $\mathcal{FL}_{trans}$ can be translated into a concept term $\psi(C)$ of $\mathcal{FL}_{reg}$ by replacing the role terms R in value restrictions of C by $\psi(R)$. Accordingly, a concept term D of $\mathcal{FL}_{reg}$ is translated into a concept term $\psi^{-1}(D)$ of $\mathcal{FL}_{trans}$, and we have $\psi(\psi^{-1}(D)) = D$ and $\psi^{-1}(\psi(C)) = C$.

[8] Alternatively, we could use the reflexive-transitive closure of roles instead of the transitive closure.

Lemma 2.12. Let C be a concept term of $\mathcal{FL}_{trans}$. Then we have $\psi(C)^I = C^I$ for any interpretation I.

Proof. By structural induction on the definition of concept terms. The only interesting case is the case $C = \forall R{:}B$ where R is a role term and B is a concept term of $\mathcal{FL}_{trans}$. We have $\psi(C) = \forall\psi(R){:}\,\psi(B)$, and we know by induction that $\psi(B)^I = B^I$. Thus $C^I = \{d \in dom(I); (d,e) \in R^I$ implies $e \in \psi(B)^I\}$, and by the definition of the semantics for $\mathcal{FL}_{reg}$, $\psi(C)^I = \{d \in dom(I)$; for all words $W \in \psi(R)$ and all individuals $e \in dom(I)$, $(d,e) \in W^I$ implies $e \in \psi(B)^I\}$.
Hence it is sufficient to show that, for any $d \in dom(I)$, the sets $d_{\psi(R)} := \{e \in dom(I)$; there exists $W \in \psi(R)$ such that $(d,e) \in W^I\}$ and $d_R := \{e \in dom(I); (d,e) \in R^I\}$ are equal.
This will be proved by **induction on the size of the role term** R.
(1) If R is the empty role, then $\psi(R)$ is the empty language and we have $d_R = \emptyset = d_{\psi(R)}$.
(2) If R is a role symbol, then $\psi(R) = R$, and $d_R = \{e \in dom(I); (d,e) \in R^I\} = d_{\psi(R)}$ since the positive regular expression R denotes the singleton $\{R\}$.
(3) Let $R = S \sqcup T$ for role terms S, T. We have $R^I = S^I \cup T^I$, $\psi(R) = \psi(S) \cup \psi(T)$, and by induction $d_S = d_{\psi(S)}$ and $d_T = d_{\psi(T)}$. But then $d_R = d_S \cup d_T = d_{\psi(S)} \cup d_{\psi(T)} = d_{\psi(R)}$.
(4) Let $R = S \circ T$ for role terms S, T. We have $R^I = S^I \circ T^I$, and $\psi(R) = \psi(S)\psi(T)$.
Assume that $e \in d_R$. Because of $(d,e) \in R^I = S^I \circ T^I$ there exists f such that $(d,f) \in S^I$ and $(f,e) \in T^I$. That means that $f \in d_S$ and $e \in f_T$. By induction, $d_S = d_{\psi(S)}$ and $f_T = f_{\psi(T)}$, and thus there exist $U \in \psi(S)$ and $V \in \psi(T)$ such that $(d,f) \in U^I$ and $(f,e) \in V^I$. But then $UV \in \psi(S)\psi(T)$ and $(d,e) \in (UV)^I$.
Assume that $e \in d_{\psi(R)}$. Thus there exists $W \in \psi(R)$ such that $(d,e) \in W^I$. Since $\psi(R) = \psi(S)\psi(T)$ there exist $U \in \psi(S)$ and $V \in \psi(T)$ such that $W = UV$. From $(d,e) \in W^I$ we can now deduce that there exists f such that $(d,f) \in U^I$ and $(f,e) \in V^I$. That means that $f \in d_{\psi(S)}$ and $e \in f_{\psi(T)}$. By induction, $d_S = d_{\psi(S)}$ and $f_T = f_{\psi(T)}$, and thus $f \in d_S$ and $e \in f_T$. This means that $(d,f) \in S^I$ and $(f,e) \in T^I$, which implies $(d,e) \in R^I$.
(5) Let $R = trans(S)$ for a role term S. We have $\psi(R) = (\psi(S))^+ = \bigcup_{n\geq 1} \psi(S)^n$, and R^I is the transitive closure of S^I, i.e., $R^I = \bigcup_{n\geq 1} (S^I)^n = \bigcup_{n\geq 1} (S^n)^I$.
As in (4) above it is easy to show that for all $n \geq 1$, $d_{S^n} = d_{\psi(S)^n}$, and thus $d_R = \bigcup_{n\geq 1} d_{S^n} = \bigcup_{n\geq 1} d_{\psi(S)^n} = d_{\psi(R)}$. ❑

If D is a concept term of $\mathcal{FL}_{reg}$, then $C := \psi^{-1}(D)$ is a concept term of $\mathcal{FL}_{trans}$, and by Lemma 2.12, $\psi^{-1}(D)^I = C^I = \psi(C)^I = \psi(\psi^{-1}(D))^I = D^I$. As an easy consequence we get

Theorem 2.13. Acyclic terminologies of $\mathcal{FL}_{trans}$, acyclic terminologies of $\mathcal{FL}_{reg}$, and possibly cyclic terminologies of $\mathcal{FL}_0$ considered with gfp-semantics have the same expressive power.

3. Extensions 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 included into the larger language $\mathcal{ALC}$ without causing any of the troubles we should have with cyclic definitions in $\mathcal{ALC}$.

Definition 3.1. (transitive and regular extensions of $\mathcal{ALC}$)
(1) In the "transitive extension" $\mathcal{ALC}_{trans}$ of $\mathcal{ALC}$ we allow to use role terms (as defined in

part (1) of Definition 2.10) instead of simply roles in value restrictions and exists-in restrictions. The semantics of $\mathcal{ALC}_{\text{trans}}$ is given by Definition 2.2 and 2.11.
(2) In the "regular extension" $\mathcal{ALC}_{\text{reg}}$ 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 2.7. 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 $(\exists L{:}C)^I := \{d \in \text{dom}(I)$; there exists a word $W \in L$ and an individual $e \in \text{dom}(I)$ such that $(d,e) \in W^I$ and $e \in C^I\}$.

As in Section 2.3 above we can now translate positive regular expressions into role terms and vice versa. It is easy to see that Lemma 2.12 also holds for concept terms of $\mathcal{ALC}_{\text{trans}}$, and thus we obtain

Proposition 3.2. Acyclic terminologies of $\mathcal{ALC}_{\text{trans}}$ and acyclic terminologies of $\mathcal{ALC}_{\text{reg}}$ have the same expressive power. In particular, subsumption in $\mathcal{ALC}_{\text{trans}}$ can be reduced in linear time to subsumption in $\mathcal{ALC}_{\text{reg}}$ and vice versa.

This shows that we may restrict our attention to one of these two languages. The definition of $\mathcal{ALC}_{\text{trans}}$ is more intuitive, and thus $\mathcal{ALC}_{\text{trans}}$ may be more appropriate if we want to apply the language to actual representation problems. But $\mathcal{ALC}_{\text{reg}}$ will turn out to be more convenient for describing the subsumption algorithm. For that reason we shall only consider $\mathcal{ALC}_{\text{reg}}$ in the remainder of this paper.

Since we only allow acyclic terminologies of $\mathcal{ALC}_{\text{reg}}$, subsumption with respect to terminologies can be reduced to subsumption of concept terms (see Section 2.1 above). As for $\mathcal{ALC}$, the subsumption problem for concept terms can further be reduced to the consistency problem.

Definition 3.3. A concept term C is called *inconsistent* iff $C^I = \emptyset$ for all interpretations I. If C is not inconsistent, it is called *consistent*.

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 inconsistent. Since our language $\mathcal{ALC}_{\text{reg}}$ allows negation of concepts, the term $C \sqcap \neg D$ is also an admissible concept term. Thus it is sufficient to have an algorithm which decides consistency of concept terms. Let us first recall by an example how consistency can be checked for concept terms of $\mathcal{ALC}$ (see Schmidt-Schauß&Smolka (1988), and Hollunder et al. (1990) for details).

3.1 An Example for the Consistency Test for $\mathcal{ALC}$

Assume that C is a concept term of $\mathcal{ALC}$ which has to be checked for consistency. 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 (nnf) where negation is only applied to concept names.

Example 3.4. Let A, B be concept symbols, and let R be role a symbol. 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 inconsistent.

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 consistent, 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) = {a, b, c}; $R^{I} := \{(a,b), (a,c)\}$; $A^{I} := \{b\}$ and $B^{I} := \{c\}$. For this interpretation, $a \in C'^{I}$. That means that $a \in (\exists R{:}A \sqcap \exists R{:}B)^{I}$, but $a \notin (\exists R{:}(A \sqcap B))^{I}$.

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

3.2 Some Ideas for a Generalization to $\mathcal{ALC}_{reg}$

A consistency 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 $bR^{I}c$ 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 $bW^{I}c$ 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.

As shown in Section 2.3, 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 the job. All we know about U is that it is an element of the set $R^{-1}L := \{V;\ RV \in L\}$.

Definition 3.5. 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 U, V we have $(UV)^{-1}L = V^{-1}(U^{-1}L)$. For example, let L be the regular language $(RS)^+$. Then $R^{-1}L = S(RS)^*$, $S^{-1}L = \emptyset$, and $(RS)^{-1}L = S^{-1}(R^{-1}L) = (RS)^*$.

We can now choose between two possibilities: U 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 consistency algorithm for $\mathcal{ALC}$ can be generalized to $\mathcal{ALC}_{\text{reg}}$. 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 $d \in C^I$ (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 $\mathcal{ALC}$ we can no longer be sure of the termination of the algorithm. This will be demonstrated by the following example.

Example 3.6. Let A be a concept name, and let R be a role name. Consider the following concept term of $\mathcal{ALC}_{\text{reg}}$: $C := A \sqcap \exists R{:}A \sqcap \forall R^+{:}(\exists R{:}A)$.
(1) We introduce an individual a_0 which has to satisfy the constraints $a_0 \in A^I$, $a_0 \in (\exists R{:}A)^I$, and $a_0 \in (\forall R^+{:}(\exists R{:}A))^I$.
(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 $a_1 \in A$.
(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 3.7. Let **K** be a finite set of regular languages. Then the set $\{W^{-1}L \setminus \{\varepsilon\}$; where $L \in$ **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 3.6 is a "good cycle"; its occurrence indicated that the concept term under consideration is in fact consistent. The following example will demonstrate how "bad cycles" may arise.

Example 3.8. Let A be a concept name, and let R be a role name. 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 the constraints $a_0 \in (\neg A)^I$, $a_0 \in (\exists R^+{:}A)^I$, 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 $a_0 R^I a_1$. 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 take 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 inconsistent while the term C of Example 3.6 was consistent.

We may now ask what makes the difference between the cycle of Example 3.6 and that of Example 3.8. In the second example we have postponed satisfying the exists-in restriction for a_0 by introducing the new exists-in restriction for a_1. It is easy to see that we should have to postpone satisfying the restriction for ever because trying to actually satisfy it will always result in a clash. In the first example however, we have already satisfied the exists-in restriction before the cycle occurs.

Building up on these ideas the next section presents a formal description of an algorithm for deciding consistency of concept terms of $\mathcal{ALC}_{reg}$.

4. An Algorithm for Testing Consistency of Concept Terms of $\mathcal{ALC}_{\mathbf{reg}}$

To keep our algorithm simple, we single out a special class of concept terms as normal forms. A concept term C is called *simple* iff C is a concept name, or a complemented concept name, or if C is of the form $\forall$L:D or $\exists$L:D where L is a nonempty regular language not containing the empty word. A *conjunctive* concept term has the form $C_1 \sqcap \ldots \sqcap C_n$ where each C_i is a simple concept term. A *subconjunction* for $C_1 \sqcap \ldots \sqcap C_n$ has the form $C_{i_1} \sqcap \ldots \sqcap C_{i_m}$. By grouping together exists and value restrictions we can write conjunctive concept terms in the form

$$A_1 \sqcap \ldots \sqcap A_m \sqcap \exists L_1{:}E_1 \sqcap \ldots \sqcap \exists L_r{:}E_r \sqcap \forall K_1{:}D_1 \sqcap \ldots \sqcap \forall K_k{:}D_k.$$

This concept term contains a *clash* iff there exist A_i and A_j such that $A_i = \neg A_j$, and it contains an *exists restriction* iff $r > 0$. A *disjunctive* concept term has the form $C_1 \sqcup \ldots \sqcup C_n$ where each C_i is a conjunctive concept term.

It is easy to see that any concept term C_0 can be transformed into an equivalent disjunctive term. This transformation can be performed as follows: First, one can eliminate the empty word from regular value and exists-in restrictions by using the fact that, for $\varepsilon \in L$, the concept terms $\forall L{:}C$ and $C \sqcap \forall(L \setminus \{\varepsilon\}){:}C$ (resp. $\exists L{:}C$ and $C \sqcup \exists(L \setminus \{\varepsilon\}){:}C$) are equivalent. In addition, the term $\forall \emptyset{:}C$ is equivalent to $A \sqcup \neg A$ for an arbitrary concept name A, and $\exists \emptyset{:}C$ is equivalent to $A \sqcap \neg A$. In a second step, we compute the *negation normal form* of the concept term, that is, we bring the negation signs immediately in front of concept symbols by rewriting the concept term via de Morgan's laws and with rules $\neg\forall L{:}C \Rightarrow \exists L{:}\neg C$ and $\neg\exists L{:}C \Rightarrow \forall L{:}\neg C$. Then we transform this concept term into disjunctive form by applying (modulo associativity and commutativity of conjunction and disjunction) the distributivity laws of conjunction over disjunction on top level. A disjunctive concept term which can be obtained by these transformations from the term C_0 is called a *disjunctive normal form* of C_0.

We now define so-called concept trees, which will be used to impose a control structure on the algorithm. A *concept tree* is a rooted tree such that every node is equipped with the following components: *type, extended, concept-term,* and *value*. The values for the component *type* range over the symbols "$\sqcap$", "$\sqcup$", "$\exists$" and "$\exists\sqcup$", for the component *extended* they range over the symbols "yes" and "no", and for *value* they range over the symbols "solved", "clash", "good cycle", "bad cycle" and "null". The values for the component *concept-term* are concept terms. Given a node N in a concept tree we will access the content of the corresponding component with N.component. A concept tree T is called *extended* if for every node N in T one has N.extended = "yes". Some of the edges of a concept tree may be marked by a role name. See Figure 4.1 for an example of a concept tree.

4.1 The Algorithm

We are now ready to present the algorithm which decides whether a given concept term of $\mathcal{ALC}_{\mathrm{reg}}$ is consistent. The algorithm proceeds as follows: First, a concept tree consisting of a single node is created. Then, in successive propagation steps, new nodes are added until we obtain an extended concept tree. The given concept term is consistent if and only if the extended concept tree satisfies a certain condition, which can be checked easily.

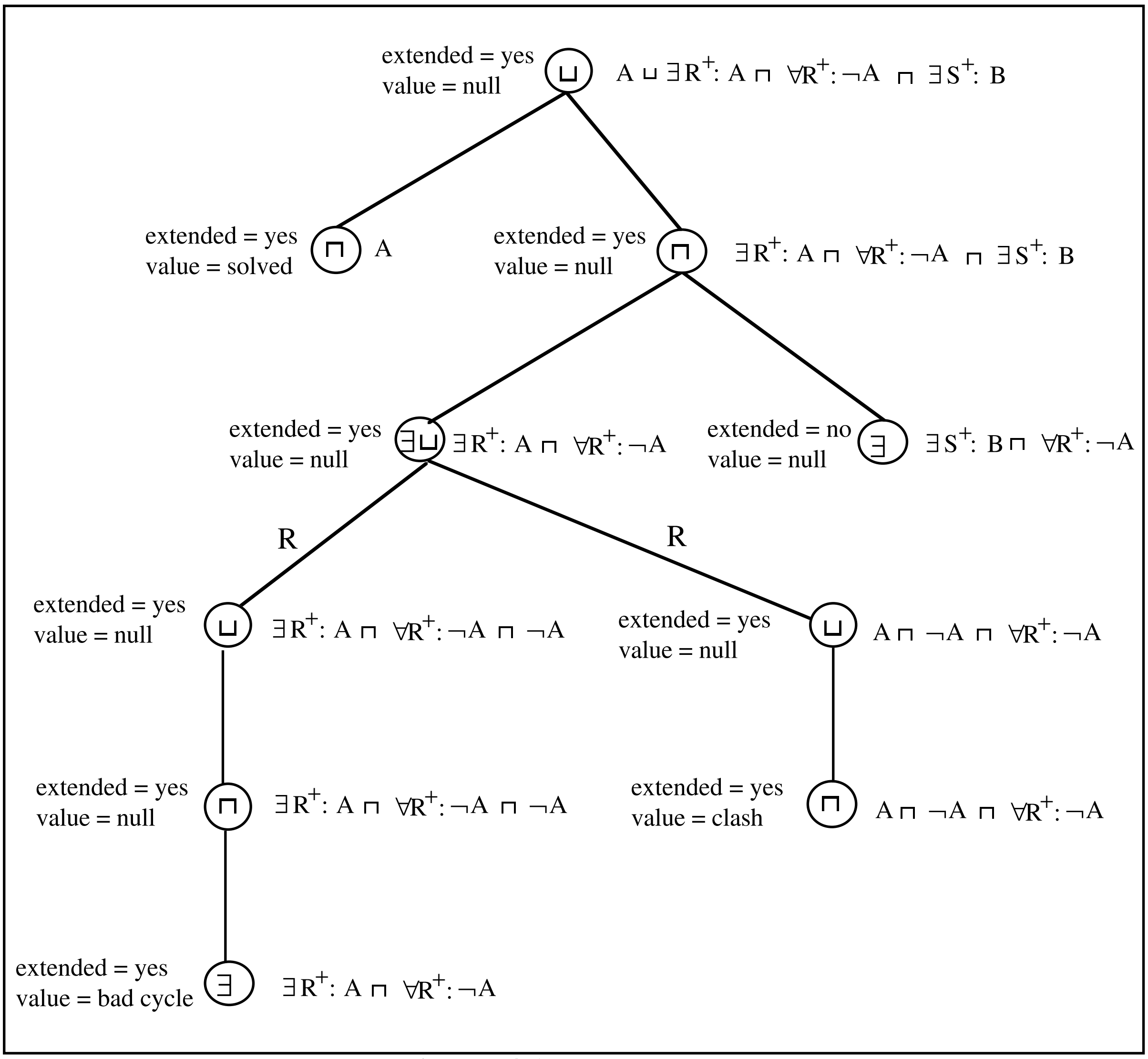


Figure 4.1 A concept tree

The algorithm uses several functions which will now be defined. The function Consistency takes a concept term as input, creates a concept tree, and returns this tree as argument to the function Extend-concept-tree. This function extends the concept tree by iterated calls of the functions Expand-or-node, Expand-and-node, and Expand-∃-node until an extended concept tree is obtained.

The function **Consistency** takes a concept term C as input and creates a concept tree T. This concept tree consists of the node root with root.type = "⊔", root.extended = "no", root.value = "null". The component root.concept-term contains the term C itself. Then the function Extend-concept-tree is called with T as argument.

The function **Extend-concept-tree** takes a concept tree as argument and returns an extended concept tree. It uses the functions Expand-or-node, Expand-and-node, and Expand-∃-node as subfunctions. Here is the formulation of the function Extend-concept-tree in a Pascal-like notation.

```
Algorithm  Extend-concept-tree  (T)
        if T is extended
                then  return  T
        elsif T contains a node N such that  N.type = "⊔" and  N.extended = "no"
                then  Extend-concept-tree  (Expand-or-node(T,N))
        elsif T contains a node N such that  N.type = "⊓" and  N.extended = "no"
                then  Extend-concept-tree  (Expand-and-node(T,N))
        else let N  be a node in T such that  N.type = "∃" and  N.extended = "no"
                Extend-concept-tree  (Expand-∃-node(T,N))
end  Extend-concept-tree.
```

The function **Expand-or-node** takes a concept tree T and a node N of type "$\sqcup$" occurring in T as arguments and returns a concept tree T'. Suppose $C_1 \sqcup C_2 \sqcup \ldots \sqcup C_n$ is a disjunctive normal form of N.concept-term. We modify T (and thereby obtain T') such that N.extended = "yes", and the (newly created) nodes N_i, $1 \le i \le n$, with N_i.type = "$\sqcap$", N_i.extended = "no", N_i.concept-term = C_i, and N_i.value = "null" are successors of N.

The function **Expand-and-node** takes a concept tree T and a node N of type "$\sqcap$" occurring in T as arguments and returns a concept tree T'. We modify T (and thereby obtain T') such that N.extended = "yes" and N.value is

- "clash" if N.concept-term contains a clash,
- "solved" if N.concept-term does not contain an exists restriction or a clash,
- "null" otherwise.

Furthermore, if N.value remains "null", we create successors for N in the following way. Suppose N.concept-term = $A_1 \sqcap \ldots \sqcap A_m \sqcap \exists L_1{:}E_1 \sqcap \ldots \sqcap \exists L_r{:}E_r \sqcap \forall K_1{:}D_1 \sqcap \ldots \sqcap \forall K_k{:}D_k$. Then for every i, $1 \le i \le r$, the (newly created) node N_i with N_i.type = "$\exists$", N_i.extended = "no", N_i.concept-term = $\exists L_i{:}C_i \sqcap \forall K_1.D_1 \sqcap \ldots \sqcap \forall K_k{:}D_k$, and N_i.value = "null" is a successor of N.

The function **Expand-∃-node** takes a concept tree T and a node N of type "$\exists$" occurring in T as arguments and returns a concept tree T'. Suppose N.concept-term = $\exists L{:}C \sqcap \forall K_1{:}D_1 \sqcap \ldots \sqcap \forall K_k{:}D_k$. We have to distinguish between two cases.

Case 1. Assume that there exists a predecessor node M of N such that M.type = "$\exists\sqcup$" and M.concept-term is equal to N.concept-term modulo associativity, commutativity, and idempotency of conjunction. That means that we have detected a cycle. The decision whether this cycle is a "good" one or a "bad" one depends on where the subterm $\exists L{:}C$ of N.concept-term comes from.

Case 1.1. The restriction $\exists L{:}C$ in N.concept-term comes from the $\exists L{:}C$ restriction in M.concept-term. That means that the $\exists L{:}C$ restriction in M.concept-term has never really been removed on the path from M to N, but only been modified to $\exists(R_1^{-1}L \setminus \{\varepsilon\}){:}C$, $\exists((R_1R_2)^{-1}L \setminus \{\varepsilon\}){:}C$, and so on, until at node N the same restriction has been reproduced

because of $(R_1R_2...R_n)^{-1}L \setminus \{\varepsilon\} = L$.
In this case we modify T (and thereby obtain T') such that N.extended = "yes" and N.value is "bad cycle".
Case 1.2. The restriction $\exists L{:}C$ in N.concept-term does not come from the $\exists L{:}C$ restriction in M.concept-term (i.e., it comes out of one of the $\forall K_i{:}D_i$ restrictions of M.concept-term). In this case we modify T (and thereby obtain T') such that N.extended = "yes" and N.value is "good cycle".
Case 2. Otherwise, we modify T (and thereby obtain T') as follows. We change N.type from "$\exists$" to "$\exists\sqcup$", and N.extended from "no" to "yes". In addition, we have to introduce new successor nodes N' for N. All these nodes get N'.type = "$\sqcup$", N'.extended = "no", and N'.value = "null". In order to describe the concept-term component of these successors we need the following definitions: Let R be a role name and let $I := \{i;\ 1 \le i \le k \text{ and } R \in K_i\}$ and $J := \{j;\ 1 \le j \le k \text{ and } R^{-1}K_j \setminus \{\varepsilon\} \neq \emptyset\}$. Then $D_R := \bigsqcap_{j \in J} \forall(R^{-1}K_j \setminus \{\varepsilon\}){:}D_j \sqcap \bigsqcap_{i \in I} D_i$.
For any role name R such that $R^{-1}L \setminus \{\varepsilon\} \neq \emptyset$ we create a successor node N' of N such that N'.concept-term = $D_R \sqcap \exists(R^{-1}L \setminus \{\varepsilon\}){:}C$. The edge from N to N' gets the label R. In addition, for any role name R such that $R \in L$ we create a successor node N' of N such that N'.concept-term = $D_R \sqcap C$. The edge from N to N' gets the label R.

We have now completed the description of the function Consistency. The first important property – which can be shown with the help of Proposition 3.7 – is that a call of this function always terminates.

Proposition 4.2. Let C_0 be a concept term. Then the call Consistency(C_0) terminates.
Proof. Assume that the algorithm Consistency does not terminate. Then an infinite concept tree is generated since each call of Expand-or-node, Expand-and-node, or Expand-$\exists$-node adds new nodes to the concept tree. Since every node has only finitely many direct successors we conclude with König's Lemma that there exists an infinite path in this tree. This infinite path contains infinitely many nodes $N_1, N_2, \ldots$ with N_i.type = "$\exists\sqcup$" and for all $i < j$, N_i.concept-term is not equal to N_j.concept-term modulo associativity, idempotency and commutativity. The concept-term components of these nodes are different because otherwise we would have N_j.value = "good cycle" or N_j.value = "bad cycle", and thus N_j would be a leaf.
On the other hand, let $\exists L{:}C \sqcap \forall K_1{:}D_1 \sqcap \ldots \sqcap \forall K_k{:}D_k$ be the value of the concept-term component of one of these nodes N_i. It is easy to see (by induction on the length of the path from the root to N_i) that the languages $L, K_1, \ldots, K_k$ are all elements of the set $\{W^{-1}M \setminus \{\varepsilon\};$ where M is a regular language occurring in one of the restrictions of C_0 and W is a word$\}$, which is finite by Proposition 3.7. In addition, the terms $C, D_1, \ldots, D_k$ are all subterms of C_0. But that means that there can be – modulo associativity, idempotency and commutativity of conjunction – only finitely many different such terms, which yields a contradiction. ❑

An *instance* of a concept tree T is obtained from T by keeping for any node N with N.type $\in$ {"$\sqcup$", "$\exists\sqcup$"} only one of its direct successors, and for all other nodes all their direct successors. Such an instance is called *successful* iff for every leaf N in this instance we have N.value = "solved" or N.value = "good cycle". Figure 4.3 shows all the instances of the concept tree of Figure 4.1. The first instance is successful whereas the other two instances are not successful.

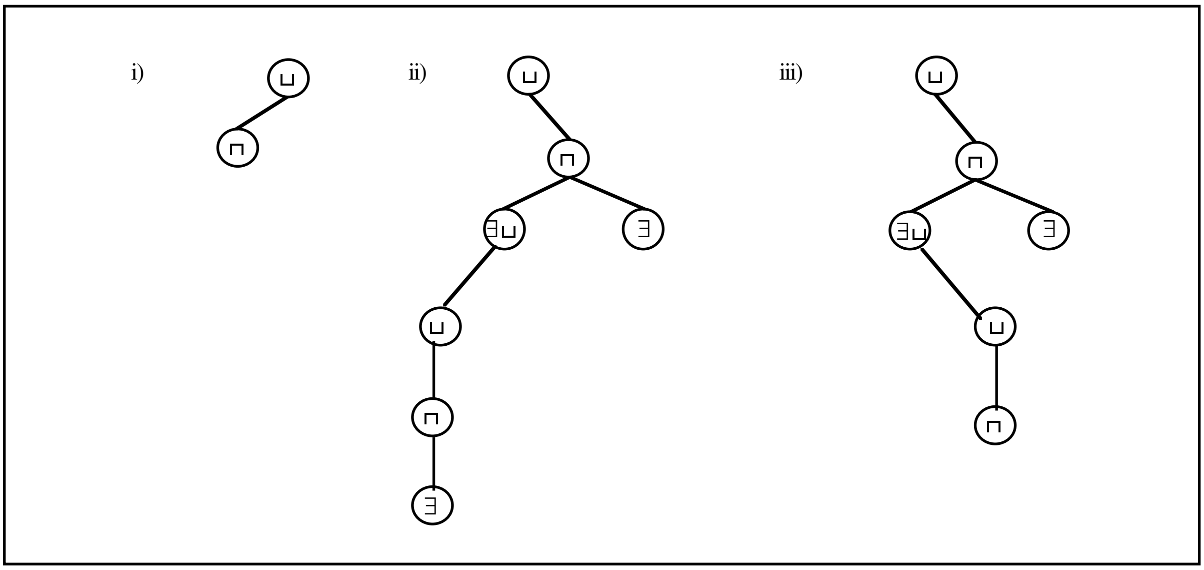


Figure 4.3 Instances of the concept tree of Figure 4.1

Now let T be the extended concept tree which is returned by the call Consistency(C_0). The existence of a successful instance of T is the criterion for consistency of C_0.

Theorem 4.4. Let C_0 be a concept term and let T be the extended concept tree computed by Consistency(C_0). Then C_0 is consistent if and only if there exists a successful instance of T.

Obviously, one can easily decide whether the concept tree computed by Consistency(C_0) contains a successful instance by using depth-first search. This finally yields the algorithm for testing consistency of concept terms of $\mathcal{ALC}_{\text{reg}}$.

4.2 Proof of Soundness and Completeness

The proof of Theorem 4.4 will be devided into two parts, namely soundness and completeness of the decision criterion it yields for consistency.

Proposition 4.5. (soundness)
Let C_0 be a concept term and let T be the extended concept tree computed by Consistency(C_0). If there exists a successful instance of T then C_0 is consistent.

In order to prove the proposition we shall show how a successful instance can be used to define an interpretation I satisfying $C_0{}^I \neq \emptyset$. Let S be a successful instance of the extended concept tree computed by Consistency(C_0). Then S yields the *canonical interpretation* I which is defined as follows:
(1) The elements of the dom(I) are all nodes N in S such that N.type = "$\sqcap$"
(2) Interpretation of role names: Let N be an element of dom(I). Then N.type = "$\sqcap$", and N.value = "solved" or N.value = "null".
If N.value = "solved", then N is a leaf in S. In this case we define for any role name R, that N does not have an R-successor in I.
If N.value = "null", then there exist n > 0 direct successors N_1, ..., N_n of N. For such a node N_i we may have N_i.type = "$\exists$" and N_i.value = "good cycle", or N_i.type = "$\exists\sqcup$" and N_i.value = "null". Assume first that N_i.type = "$\exists\sqcup$" and N_i.value = "null". Then N_i has exactly one direct

successor N_i' with N_i'.type = "$\sqcup$", and the edge from N_i to N_i' is labeled with some role name R_i. In addition, N_i' has exactly one direct successor N_i" with N_i".type = "$\sqcap$".
Now assume that N_i.value = "good cycle". Then there exists a node M_i such that M_i is a predecessor of N_i and M_i.concept-term = N_i.concept-term (modulo associativity, commutativity, and idempotency). Let N_i' be the direct succcessor of M_i. Obviously, N_i'.type = "$\sqcup$", and the edge from M_i to N_i' is labeled with some role name R_i. In addition, N_i' has exactly one direct successor N_i" with N_i".type = "$\sqcap$".
In both cases, the node N_i" is an element of dom(I). We define the interpretation of the roles such that $(N,N_i") \in R_i^I$ for i = 1, ..., n; and these are the only role successors of the individual N.
(3) Interpretation of concept names: For a node $N \in$ dom(I) we have that N.concept-term is of the form $A_1 \sqcap ... \sqcap A_m \sqcap \exists L_1:E_1 \sqcap ... \sqcap \exists L_r:E_r \sqcap \forall K_1:D_1 \sqcap ... \sqcap \forall K_k:D_k$. For a concept name A we define $N \in A^I$ iff there exist an index i such that $A = A_i$. Please note that N.value ≠ "clash" because S was assumed to be a successful instance. Hence, if $A = A_i$, then there does not exist an A_j with $A_j = \neg A$. This shows that $N \in (A_1 \sqcap ... \sqcap A_m)^I$.

Before we can show that this canonical interpretation satisfies $C_0^I \neq \emptyset$, we need one more definition. The *depth* τ of a concept term in negation normal form is defined as:

- $\tau(A) = \tau(\neg A) = 0$ if A is a concept name
- $\tau(\forall L:C) = \tau(\exists L:C) = 1 + \tau(C)$
- $\tau(C \sqcap D) = \tau(C \sqcup D) = \max \{ \tau(C), \tau(D)\}$

Lemma 4.6. Let I be the canonical interpretation induced by the successful instance S of T = Consistency(C_0), and let $N \in$ dom(I). If the concept term G is a subconjunction of N.concept-term, then $N \in G^I$.
Proof. The lemma is proved by induction on the depth $\tau(G)$.
τ(D) = 0. Then G is of the form $A_{i_1} \sqcap ... \sqcap A_{i_n}$. By definition of the interpretation of concept names we have $N \in (A_1 \sqcap ... \sqcap A_m)^I$, which implies $N \in (A_{i_1} \sqcap ... \sqcap A_{i_n})^I$.
τ(D) > 0. Let $G = G_1 \sqcap ... \sqcap G_n$. We have to show for all i, $1 \leq i \leq n$, that $N \in G_i^I$. If $\tau(G_i) < \tau(G)$ we know by the induction hypothesis that $N \in G_i^I$. Now suppose $\tau(G_i) = \tau(G)$. Then G_i is of the form $\forall K:E$ or $\exists L:E$ where $\tau(E) = \tau(G) - 1$.
(1) First we assume that **$G_i = \forall K:E$**. For any $M \in$ dom(I) and any $W \in K$ we have to show that $(N,M) \in W^I$ implies $M \in E^I$. Let $W = R_1...R_t$ be a word in K, and $M_1, ..., M_t = M$ be elements of dom(I) such that $N\, R_1^I\, M_1\, R_2^I ...\, R_t^I\, M_t = M$.
In the successful instance S there is a path from N to M_1. On this path we have a direct successor node N' of N such that N'.type = "$\exists\sqcup$" or N'.type = "$\exists$". By definition of the function Expand-and-node, $\forall K:E$ is a subconjunction of N'.concept-term.
Assume first that **N'.type = "$\exists\sqcup$"**. The node N' has a direct successor N" of type "$\sqcup$" which is reached by an edge labeled with R_1. The concept-term of the node N" is of the form $C_1 \sqcap \forall(R_1^{-1}K \setminus \{\varepsilon\}):E$ since $R_1^{-1}K \setminus \{\varepsilon\} \neq \emptyset$. Let $D_1 \sqcup ... \sqcup D_r$ be a disjunctive normal form of this term. Then the node M_1, which is a direct successor of N", has one of these terms D_i as its concept term. Thus we get that $\forall(R_1^{-1}K \setminus \{\varepsilon\}):E$ is a subconjunction of M_1.concept-term.
If **N'.type = "$\exists$"** then N'.value = "good cycle", and there exists a predecessor node O' of N' such that O'.concept-term is equal to N'.concept-term (modulo associativity, commutativity and idempotency of conjunction). Now the node O' has a direct successor N"

of type "⊔" which is reached by an edge labeled with R_1, and we can proceed as above to get that $(R_1^{-1}K \setminus \{\varepsilon\})$:E is a subconjunction of M_1.concept-term.
Analogously, one can show that the concept term $\forall((R_1R_2)^{-1}K \setminus \{\varepsilon\})$:E is a subconjunction of M_2.concept-term, ..., $\forall((R_1R_2...R_{t-1})^{-1}K \setminus \{\varepsilon\})$:E is a subconjunction of M_{t-1}.concept-term. The node M_{t-1} has a direct successor node M' on the path from M_{t-1} to M_t such that M'.type = "∃⊔" or M'.type = "∃". Assume that M'.type = "∃⊔" (as above the other case is similar). By definition of the function Expand-and-node, $\forall((R_1R_2...R_{t-1})^{-1}K \setminus \{\varepsilon\})$:E is a subconjunction of M'.concept-term. The node M' has a direct successor M" of type "⊔" which is reached by an edge labeled with R_t. The concept-term of the node M" is of the form $F = C_{t-1} \sqcap E$ since $R_t \in (R_1R_2...R_{t-1})^{-1}K \setminus \{\varepsilon\}$. Let $E_1 \sqcup ... \sqcup E_s$ be a disjunctive normal form of E, and $F_1 \sqcup ... \sqcup F_r$ be a disjunctive normal form of F. The node M_t, which is a direct successor of M", has one of the terms F_i as its concept term. Since $F = C_{t-1} \sqcap E$, we have that $F_i = M_t$.concept-term has one of the terms E_j as subconjunction. Obviously, $\tau(E_j) \leq \tau(E) < \tau(\forall K{:}E) = \tau(G)$, and thus we can apply the induction hypothesis to M_t and E_j. This yields $M_t \in E_j^I$, and thus $M_t \in E^I$, which is what we wanted to show.
(2) Assume that $\mathbf{G_i = \exists L{:}E}$. In the successful instance S we thus have a direct successor node N_1' of N such that ∃L:E is a subconjunction of N_1'.concept-term and N_1'.type = "∃⊔" or N_1'.type = "∃".
Assume first that $\mathbf{N_1}$**'.type = "∃⊔"**. The node N_1' has a direct successor N_1" of type "⊔" which is reached by an edge labeled with some role R_1. The concept term of N_1" is either of the form $C_1 \sqcap E$ or of the form $C_1 \sqcap \exists(R_1^{-1}L \setminus \{\varepsilon\})$:E. In the second case $R_1^{-1}L \setminus \{\varepsilon\} \neq \emptyset$, and in the first case $R_1 \in L$. Let $D_1 \sqcup ... \sqcup D_r$ be a disjunctive normal form of N_1".concept-term. Then the direct successor M_1 of N_1" has one of these terms D_i as its concept term.
In the first case we can show by induction (as in the case "$G_i = \forall K{:}E$" above) that $M_1 \in E^I$; and since $(N,M_1) \in R_1^I$ and $R_1 \in L$ we have $N \in (\exists L{:}E)^I$.
In the second case we get that $\exists(R_1^{-1}L \setminus \{\varepsilon\})$:E is a subconjunction of M_1.concept-term.
If $\mathbf{N_1}$**'.type = "∃"** then N_1'.value = "good cycle", and there exists a predecessor node O_1' of N_1' such that O_1'.concept-term is equal to N_1'.concept-term (modulo associativity, commutativity and idempotency of conjunction). Now the node O_1' has a direct successor N_1" of type "⊔", and we can proceed as for the case "N_1'.type = "∃⊔" " above.
Now assume that we already have nodes $M_1, ..., M_t \in$ dom(I) and roles $R_1, ..., R_t$ such that $N\ R_1^I\ M_1\ R_2^I\ ...\ R_t^I\ M_t$, $(R_1R_2...R_t)^{-1}L \setminus \{\varepsilon\} \neq \emptyset$, and $\exists((R_1R_2...R_t)^{-1}L \setminus \{\varepsilon\})$:E is a subconjunction of M_t.concept-term. As above we can now find a role R_{t+1} and a node $M_{t+1} \in$ dom(I) such that $M_t\ R_{t+1}^I\ M_{t+1}$, and either $M_{t+1} \in E^I$, or $(R_1R_2...R_tR_{t+1})^{-1}L \setminus \{\varepsilon\} \neq \emptyset$ and $\exists((R_1R_2...R_tR_{t+1})^{-1}L \setminus \{\varepsilon\})$:E is a subconjunction of M_{t+1}.concept-term.
If we iterate this process, and finally get the first case for some t then we are done. Thus assume that we always get the second case. Then we have an infinite sequence of nodes N, M_1, M_2, M_3, ..., and an infinite sequence of roles R_1, R_2, R_3, ... such that $N\ R_1^I\ M_1\ R_2^I\ M_2\ R_3^I$... and, for all $t \geq 1$, $(R_1R_2...R_t)^{-1}L \setminus \{\varepsilon\} \neq \emptyset$ and $\exists((R_1R_2...R_t)^{-1}L \setminus \{\varepsilon\})$:E is a subconjunction of M_t.concept-term. For all $t \geq 1$ let N_t' be the direct successor of M_t of type "∃⊔" or "∃" which has $\exists((R_1R_2...R_t)^{-1}L \setminus \{\varepsilon\})$:E as subconjunction of its concept term. Since the successful instance S is finite, we have infinitely many indices t_1, t_2, t_3, ... such that N_{t_i}'.type = "∃", i.e., N_{t_i}'.value = "good cycle". We assume that N_{t_1}', N_{t_2}', ... are all the nodes of type "∃" in the sequence N_1', N_2', N_3', ... Let O_{t_i}' be the predecessor node of N_{t_i}' such that O_{t_i}'.concept-term and N_{t_i}'.concept-term are equal (modulo associativity, commutativity

and idempotency of conjunction). For any $i \geq 1$ we have that the nodes N_{t_i+1}', ..., $N_{t_{i+1}-1}$' are of type "∃⊔" Thus there is a path in S from O_{t_i}' to $N_{t_{i+1}}$'. Obviously, we have that the exists-in restriction in the concept term of $N_{t_{i+1}}$' comes from the exists-in restriction in O_{t_i}'. Since $N_{t_{i+1}}$'.value = "good cycle" this means that $O_{t_{i+1}}$' has to be a strict predecessor of O_{t_i}'. But this is a contradiction because we cannot have an infinite chain of strict predecessors O_{t_1}', O_{t_2}', ... in S.
This completes the proof of the lemma. ❑

Lemma 4.6 can now be used to prove Proposition 4.5 as follows. Let I be the canonical interpretation induced by the successful instance S of T = Consistency(C_0). The root of S has type "⊔" and concept-term C_0. If $C_1 \sqcup ... \sqcup C_n$ is the disjunctive normal form of C_0, then there exists an i, $1 \leq i \leq n$, such that the direct successor N_i of the root has C_i as its concept term. We have $N_i \in$ dom(I), and if we take C_i as subconjunction of itself the lemma yields $N_i \in C_i^I$. Thus we also have $N_i \in C_0^I$, which completes the proof of Proposition 4.5.

Proposition 4.7. (completeness)
Let C_0 be a concept term and let T be the extended concept tree computed by Consistency(C_0). If C_0 is consistent then there exists a successful instance of T.
Proof. Let I be an interpretation such that $C_0^I \neq \emptyset$. In order to prove the proposition we show how this interpretation can be used to guide the search for a successful instance of T. To that purpose we shall label nodes of T with elements of dom(I) such that a node N labeled by $b \in$ dom(I) satisfies

❍ $b \in (N.\text{concept-term})^I$

To begin with the labeling, we label the **root** of T with an element b_0 of dom(I) satisfying $b_0 \in C_0^I$. Because root.concept-term = C_0 the above condition is satisfied.
Now assume that N is a **node of type "⊔"** which has already been labeled with the individual b, and let N.concept-term = C. By induction we can assume that $b \in C^I$. Let $C_1 \sqcup ... \sqcup C_n$ be the disjunctive normal form of C generated by the algorithm. Obviously, there exists some i, $1 \leq i \leq n$, such that $b \in C_i^I$. The node N has direct successors N_1, ..., N_n with N_1.concept-term = C_1, ..., N_n.concept-term = C_n. We choose an arbitrary index i such that $b \in C_i^I$. Now the node N_i gets label b, and all the other nodes N_j with $i \neq j$ remain unlabeled.
Assume that N is a **node of type "⊓"** which has already been labeled with the individual b, and let N.concept-term = C. By induction we can assume that $b \in C^I$. If N is a leaf there is nothing to be done. Otherwise, we label all the direct successors N_i of N with b. Since the concept-term component of the nodes N_i are subconjunctions of C, we have $b \in (N_i.\text{concept-term})^I$ for all these successor nodes N_i.
Assume that N is a **node of type "∃⊔"** which has already been labeled with the individual b, and let N.concept-term = C. By induction we can assume that $b \in C^I$. Obviously, the term C is of the form $C = \exists L{:}B \sqcap \forall K_1{:}D_1 \sqcap ... \sqcap \forall K_k{:}D_k$. We consider the set

$$S(C) := \{(c,W);\ c \in C^I,\ W \in L, \text{ and there exists } d \in B^I \text{ with } cW^Id\}$$

Since we have $b \in C^I$, this set is not empty. Let (c,W) be an element of S(C) where the length of W is minimal, and let $d \in B^I$ be such that cW^Id.
Assume first that W = R for a role name R. Then N has a direct successor N_i such that N_i.concept-term = $D_R \sqcap B$ (see the definition of the function Expand-∃-node). It is easy to see that $d \in (D_R \sqcap B)^I$. Thus we may label the node N_i with d.

Now assume that $W = RU$ for a nonempty word U. From cW^Id we derive that there exists an individual d' such that cR^Id' and $d'U^Id$. Since $R^{-1}L \setminus \{\varepsilon\} \neq \emptyset$ there exists a direct successor N_i of N such that N_i.concept-term = $D_R \sqcap \exists(R^{-1}L \setminus \{\varepsilon\})$:B (see the definition of the function Expand-∃-node). It is easy to see that $d' \in (D_R \sqcap \exists(R^{-1}L \setminus \{\varepsilon\}):B)^I$. Thus we may label the node N_i with d'.

Assume that N is a **node of type "∃"** which has already been labeled with the individual b. Then N is a leaf, and there is nothing to be done.

This completes the description of the labeling process. The result of this process describes an **instance of T** as follows: we just have to remove all the nodes without label. It remains to be shown that this instance is successful. This is an immediate consequence of the following claim.

Claim. If N is a leaf which has been labeled by some individual $b \in$ dom(I), then N.value = "solved" or N.value = "good cycle".

Proof of the claim. If a leaf does not have value "solved" or "good cycle" it must have value "clash" or "bad cycle".

Assume that **N.value = "clash"**. Then N.type = "⊓" and N.concept-term is of the form $A \sqcap \neg A \sqcap C'$ for a concept name A and a concept term C'. Since N is labeled by b we know that $b \in (A \sqcap \neg A \sqcap C')^I$. But this is a contradiction since b cannot be both in A^I and $(\neg A)^I$.

Now assume that **N.value = "bad cycle"**. In this case N.type = "∃" and there exists a predecessor N_0 of N such that N_0.type = "∃⊔" and N_0.concept-term is equal to N.concept-term modulo associativity, commutativity and idempotency of conjunction.

Let $N_0, N_1, ..., N_{k-1}$ be all the nodes of type "∃⊔" on the path from N_0 to $N =: N_k$, and let $C_0, C_1, ..., C_k$ denote their concept-term component. If C_0 is of the form $D_0 \sqcap \exists L$:B, then for all i, $1 \leq i \leq k$, C_i is of the form $D_i \sqcap \exists((R_1...R_i)^{-1}L \setminus \{\varepsilon\})$:B since N_k.value = "bad cycle".

For all i, $0 \leq i \leq k-1$, let (c_i, W_i) be the element of $S(C_i)$ chosen in the labeling process, and let m_i be the length of the word W_i. For i = k let (c_k, W_k) be an element of $S(C_k)$ such that the length m_k of W_k is minimal. Since C_0 and C_k are equal up to associativity, commutativity and idempotency of conjunction, we must have $m_0 = m_k$.

We consider the step from N_i to N_{i+1} more closely. We have $c_i \in (D_i \sqcap \exists((R_1...R_i)^{-1}L \setminus \{\varepsilon\}):B)^I$ and there exists $d_i \in B^I$ such that $c_i W_i^I d_i$. Since the node N_{i+1} has $C_{i+1} = D_{i+1} \sqcap \exists((R_1...R_{i+1})^{-1}L \setminus \{\varepsilon\})$:B as its concept term we know that $W_i = R_{i+1}U_i$ for a nonempty word U_i Thus there exists $d_i' \in$ dom(I) such that $c_i R_{i+1}{}^I d_i'$ and $d_i' U_i{}^I d_i$. Obviously, $|U_i| < |W_i| = m_i$. By the definition of the labeling process, the node N_{i+1} gets d_i' as its label. Thus $d_i' \in C_{i+1}{}^I$ and since $W_i \in (R_1...R_i)^{-1}L \setminus \{\varepsilon\}$ we have $U_i \in (R_1...R_{i+1})^{-1}L \setminus \{\varepsilon\}$. This shows that the pair (d_i', U_i) is an element of the set $S(C_{i+1})$. Since the length m_{i+1} of W_{i+1} is minimal, we have $m_{i+1} \leq |U_i| < |W_i| = m_i$. Thus we have shown that $m_0 < m_1 < ... < m_k$, which is a contradiction since we know that $m_0 = m_k$.

This completes the proof of the claim, and thus the proof of the proposition. ❑

To sum up we have now proved Theorem 4.4 which yields a decision criterion for consistency of concept terms of $\mathcal{ALC}_{reg}$.

4.3 Some Remarks on the Implementation

In order to detect the "good cycles" and "bad cycles" in the calls of the function Expand-∃-node one has to decide equality of regular languages. To be more precise, we have until now assumed that the regular languages in value and exists-in restrictions are given by positive regular expressions, and we have not distiguished between the expressions and the languages they describe.

If α is a positive regular expression for the regular language L then we know that $R^{-1}L \setminus \{\varepsilon\}$ is also a regular language not containing the empty word. Thus there exists a positive regular expression β describing this language. But of course, the expression "$R^{-1}\alpha \setminus \{\varepsilon\}$" is not a regular expression. However, it is easy to see how a positive regular expression for $R^{-1}L \setminus \{\varepsilon\}$ can be obtained from a given positive regular expression for L.[9]

The problem is that a given regular language not containing the empty word can be described by different positive regular expressions. In order to detect the cycles in our algorithm we thus have to decide whether two positive regular expressions describe the same language. This is not a trivial problem. In fact, it can be shown that this problem is PSPACE-complete (see Garey&Johnson (1979)). If one wants to avoid employing a decision procedure for equivalence of positive regular expressions in each call of `Expand-∃-node` one can use the following preprocessing step.

Let $\alpha_1, \alpha_2, ..., \alpha_k$ be the positive regular expressions occurring in the concept term C_0 which has to be tested for consistency, and let $L_1, L_2, ..., L_k$ be the regular languages described by these expressions. We want to construct a finite automaton $\mathcal{A}$ which can be used to accept the languages $L_1, L_2, ..., L_k$ and all their left quotients.

If $\mathcal{A}$ is a finite automaton with a fixed set of terminal states, and q is a state in $\mathcal{A}$ then we denote by L(q) the regular language accepted by $\mathcal{A}$ if q is taken as initial state.

Proposition 4.8. Let $\alpha_1, \alpha_2, ..., \alpha_k$ be positive regular expressions, and let $L_1, L_2, ..., L_k$ be the regular languages described by these expressions. Then one can construct a finite deterministic and complete automaton $\mathcal{A}$ which satisfies the following properties:
(1) For all i, $1 \leq i \leq k$, there exists a state q_i of $\mathcal{A}$ such that $L_i = L(q_i)$.
(2) For all state q, q' of $\mathcal{A}$ we have $L(q) \setminus \{\varepsilon\} = L(q') \setminus \{\varepsilon\}$ iff $q = q'$.
(3) Let q be a state of the automaton, let R be a symbol of the alphabet, and let q' be the state reached from q by the transition with label R. Then $L(q') = R^{-1}L(q)$.
Sketch of the proof. First one constructs nondeterministic automata $\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_k$ out of the positive regular expressions $\alpha_1, \alpha_2, ..., \alpha_k$. Let $\mathcal{B}$ denote the disjoint union of these automata. With the help of the well-known subset construction, $\mathcal{B}$ can now be transformed into a complete and deterministic automaton. Please note that it is enough to construct all the subsets which can be reached from the sets of initial states of the automata $\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_k$. We have thus obtained an automaton satisfying (1) and (3) of the proposition. This automaton can now be transformed into an automaton satisfying also property (2) of the proposition. To that purpose one can use a process which is very similar to the usual minimization of deterministic automata. ❑

Because of the subset construction, the automaton $\mathcal{A}$ may be exponential in the size of the expressions $\alpha_1, \alpha_2, ..., \alpha_k$. But we already have another exponential preprocessing step,

[9] See e.g. Savage (1976) where this is described for regular expressions.

namely unfolding of the original T-box. Unfolding of the T-box and constructing the automaton $\mathcal{A}$ together also yield one exponential preprocessing step, and thus we are not worse off than before.

Once we have the automaton $\mathcal{A}$, we can use states of $\mathcal{A}$ instead of regular expressions in the value and exists-in restrictions. Going from L to $R^{-1}L \setminus \{\varepsilon\}$ now just means that we have to make a transition in $\mathcal{A}$. Instead of testing equivalence of regular expressions we must only test for physical identity of states of $\mathcal{A}$.

5. Restricting the Semantics

Since $\mathcal{ALC}$ can be seen as a sublanguage of $\mathcal{ALC}_{reg}$[10], the algorithm given in the previous section also yields a consistency test for concept terms of $\mathcal{ALC}$. It is easy to see that for a concept term C_0 of $\mathcal{ALC}$ the extended concept tree T computed by Consistency(C_0) does not contain nodes with value "good cycle" or "bad cycle". In fact, if N and N' are nodes of type "∃" or "∃⊔", and if N' is a successor of N, then the depth of N'.concept-term is smaller than the depth of N.concept-term.

If S is a successful instance of T then S does not contain nodes of type "good cycle". This means that the canonical interpretation I defined by S does not contain so-called assertional cycles[11].

Definition 5.1. Let I be an interpretation. Then we say that I contains an *assertional cycle* iff there exists an individual d in dom(I) and a nonempty word W over the alphabet of all role names such that dW^Id.

Since the canonical interpretation I defined by a successful instance S of an extended concept tree is always finite we thus have

Proposition 5.2. Let C_0 be a concept term of $\mathcal{ALC}$. Then C_0 is consistent iff there exists a finite interpretation I without assertional cycles such that $C_0{}^I \neq \emptyset$.[12]

For concept terms of $\mathcal{ALC}_{reg}$ this proposition does not hold. As an example one can take the concept term $A \sqcap \exists R{:}A \sqcap \forall R^+{:}(\exists R{:}A)$ of Example 3.6. This term is consistent, but it is easy to see that there does not exist a finite interpretation I without assertional cycles such that $(A \sqcap \exists R{:}A \sqcap \forall R^+{:}(\exists R{:}A))^I \neq \emptyset$.

If one wants to avoid such cyclic models one can consider $\mathcal{ALC}_{reg}$ with the following restricted semantics: Instead of all interpretations we allow only finite interpretations without assertional cycles. We shall call consistency with respect to this semantics "strong consistency".

Definition 5.3. Let C_0 be a concept term of $\mathcal{ALC}_{reg}$. Then C_0 is called *strongly consistent*

---

[10]The restriction $\exists R{:}C$ can be seen as an ordinary exists-in restriction in $\mathcal{ALC}$, but also as a regular exists-in restriction in $\mathcal{ALC}_{reg}$ where R stands for the singleton set {R}.

[11]This name was introduced in Nebel (1990a).

[12]Of course, this proposition can also be obtained as a consequence of the algorithm described in Schmidt-Schauß&Smolka (1988).

iff there exists a finite interpretation I without assertional cycles such that $C_0^I \neq \emptyset$.

We shall now demonstrate by an example why this kind of semantics may be interesting. Assume that we want to describe knowledge about procedures and correctness of procedures within the language $\mathcal{ALC}_{\text{reg}}$. Let Procedure and Locally_correct be primitive concepts, and let sub_proc be a role. A procedure with subprocedures can now be described by the concept term

$$\text{Procedure} \sqcap \exists \text{sub\_proc}: \text{Procedure}.$$

The individuals of an interpretation are procedures which stand in a subprocedure relationship given by the interpretation of the role sub_proc. By a subprocedure we mean a procedure which has an explicit call in the procedure in question. A procedure is correct if it is locally correct (i.e., the code explicitly given in this procedure is correct), and all its sub-procedures are correct. Thus correct procedures can be expressed by the concept term

$$\begin{aligned}\text{Procedure} \sqcap\ & \text{Locally\_correct} \sqcap \\ & \forall(\text{sub\_proc})^+: (\text{Procedure} \sqcap \text{Locally\_correct}).\end{aligned}$$

Of course we assume that we have only finitely many different procedures in a given programming environment. Under this condition, the following concept term can only contain procedures which eventually call a recursive procedure:

$$\begin{aligned}\text{Procedure} \sqcap\ & \exists \text{sub\_proc}: \text{Procedure} \sqcap \\ & \forall(\text{sub\_proc})^+: (\exists \text{sub\_proc}: \text{Procedure}).\end{aligned}$$

If our programming language does not allow recursive procedures this term should be inconsistent. This can be achieved by restricting the semantics to finite interpretations without assertional cycles, i.e., by considering strong consistency instead of consistency.

The algorithm described in Section 4 can also be used to decide strong consistency of concept terms of $\mathcal{ALC}_{\text{reg}}$.

Theorem 5.4. Let C_0 be a concept term of $\mathcal{ALC}_{\text{reg}}$, and let T be the extended concept tree computed by Consistency(C_0). Then C_0 is strongly consistent if and only if there exists an instance S of T such that for every leaf N of S we have N.value = "solved".

Proof. As in Section 4.2 we have to prove soundness and completeness of this criterion.

(1) The proof of soundness is trivial. In fact, if S is an instance of T such that for every leaf N of S we have N.value = "solved", then the canonical interpretation I defined by this successful instance S is finite and does not contain assertional cycles. In the proof of Proposition 4.5 we have shown that this interpretation I satisfies $C_0^I \neq \emptyset$.

(2) Assume that I is a finite interpretation without assertional cycles such that $C_0^I \neq \emptyset$. Similar to the proof of Proposition 4.7 we shall show how this interpretation can be used to find an appropriate instance of T. The labeling process as defined in the proof of Proposition 4.7 is modified as follows. The **nodes of type "∃", "⊔", and "⊓"** are treated as before. Let N be a **node of type "∃⊔"** which has already been labeled with the individual b, and let N.concept-term = $C = \exists L{:}B \sqcap \forall K_1{:}D_1 \sqcap \ldots \sqcap \forall K_k{:}D_k$. By induction we can assume that $b \in C^I$. For an individual $c \in \text{dom}(I)$ we define

$$\kappa(c) := \max\{m;\ \text{there exists } d \in \text{dom}(I) \text{ with } cW^I d \text{ and } m = |W|\}.$$

This maximum always exists since I is finite and does not contain assertional cycles. Now we choose $c \in C^I$ such that $\kappa(c)$ is minimal. Such an individual exists since $b \in C^I$. Since $c \in C^I$ we have $c \in (\exists L{:}B)^I$, and thus there exists a word W and an individual d with cW^Id and $d \in B^I$. We can now proceed with (c,W) as in the proof of Proposition 4.7.
The result of this labeling process describes an **instance S of T** as in the proof of Proposition 4.7: we just have to remove all the nodes without label. It remains to be shown that this instance is such that for every leaf N of S we have N.value = “solved”.
As before it is easy to show that S cannot have a leaf with value “clash”. Now assume that N is a leaf of S such that N.value is “good cycle” or “bad cycle”. Thus there exists a predecessor N_0 of N such that N_0.type = “$\exists\sqcup$”, and N_0.concept-term is equal to N.concept-term up to associativity, commutativity and idempotency of conjunction. Let $N_0, N_1, ..., N_{k-1}$ be all the nodes of type “$\exists\sqcup$” on the path from N_0 to $N =: N_k$, and let $C_0, C_1, ..., C_k$ denote their concept-term component. For all i, $0 \leq i \leq k-1$, let c_i be the element of C_i^I chosen in the labeling process, and let $m_i := \kappa(c_i)$. For $i = k$ let c_k be an element of C_k^I such that $m_k := \kappa(c_k)$ is minimal. Since C_0 and C_k are equal up to associativity, commutativity and idempotency of conjunction, we must have $m_0 = m_k$.
We consider the step from N_i to N_{i+1} more closely. By the definition of the labeling process there exists a role R_{i+1} and an individual $d_i \in C_{i+1}^I$ such that $c_iR_{i+1}^Id_i$. Obviously, $c_iR_{i+1}^Id_i$ implies $\kappa(c_i) > \kappa(d_i)$, and $d_i \in C_{i+1}^I$ implies $\kappa(d_i) \geq \kappa(c_{i+1})$. Thus we have shown that $m_0 < m_1 < ... < m_k$, which is a contradiction since we know that $m_0 = m_k$. ❑

6. Internalizing Concept Equations

Until now we have only allowed concept definitions in our T-boxes. That means that we have considered equations of the form A = D where A is a concept name and D is a concept term. The additional restriction was that each concept name occurs only once as left hand side of a definition.

A *concept equation* is of the form C = D where C and D are arbitrary concept terms. An interpretation I satisfies the equation C = D iff $C^I = D^I$ holds. The interpretation I is a model of the set $\mathcal{E}$ of concept equations iff satisfies all the equations in $\mathcal{E}$. A concept term C is *consistent* w.r.t. $\mathcal{E}$ iff there exists a model I of $\mathcal{E}$ such that $C^I \neq \emptyset$; and C is *subsumed* by D w.r.t. $\mathcal{E}$ iff $C^I \subseteq D^I$ holds for all models I of $\mathcal{E}$.

Obviously, we have C is subsumed by D w.r.t. $\mathcal{E}$ iff the term $C \sqcap \neg D$ is inconsistent w.r.t. $\mathcal{E}$. Subsumption and consistency w.r.t. a T-box is a special case of this notion of subsumption and consistency w.r.t. a set of concept equations because T-boxes are just sets of concept equations of a very specific form.

Though we have defined subsumption and consistency with respect to finite sets of concept equations it is always enough to consider only one concept equation of the simplified form C = Top. Here Top denotes the concept term $A \sqcup \neg A$ for an arbitrary concept name A. Obviously, we have $Top^I = dom(I)$ for all interpretations I. An interpretation I satisfies a concept equation C = D iff it satisfies $(C \sqcup \neg D) \sqcap (\neg C \sqcup D) = Top$, and it satisfies $C_1 = Top$, ..., $C_n = Top$ iff it satisfies $C_1 \sqcap ... \sqcap C_n = Top$. This shows how a finite set of concept equations can be encoded into a single equation of the form C = Top.

In remainder of this section we shall show that consistency (and thus also subsumption) of concept terms of $\mathcal{ALC}_{reg}$ w.r.t. finite sets of concept equations can be reduced to ordinary consistency of concept terms of $\mathcal{ALC}_{reg}$. That means that for this language concept equations can be internalized into the term language. In order to show this result we need some auxiliary definitions and facts.

Let I be an interpretation and let d be an element of dom(I). We define

$$\text{gen}(d) := \{ e \in \text{dom}(I);\ \text{there exists a word W over the alphabet of role names with } dW^I e\}$$

and say that an element of gen(d) is generated by d. Obviously, $d \in$ gen(d), and $e \in$ gen(d) implies that gen(e) $\subseteq$ gen(d). Our intention is now to restrict the domains of our interpretations to such sets gen(d). We say that an interpretation I is *rooted* iff there exists an individual $d \in$ dom(I) such that dom(I) = gen(d). In this case, d is called a *root* of I.

In order to show that it is sufficient to consider such rooted interpretations when interested in consistency of concept terms, we need the following notion of restriction of an interpretation. Let I be an interpretation, and let M be a subset of dom(I). Then the *restriction* $I|_M$ of I to the subset M is the interpretation defined by

(1) $\text{dom}(I|_M) := M \cap \text{dom}(I)$,
(2) $A^{I|_M} := M \cap A^I$ for all concept names A, and
(3) $R^{I|_M} := M \times M \cap R^I$ for all role names R.

Lemma 6.1. Let I be an interpretation, and let M be a subset of dom(I). Then for all concept terms C of $\mathcal{ALC}_{reg}$, and all elements d of dom(I) satisfying gen(d) $\subseteq$ M we have

$$d \in C^I \text{ if and only if } d \in C^{I|_M}.$$

Proof. The lemma is proved by induction on the structure of C.
(1) C = A for a concept name A. We have $A^{I|_M} = M \cap A^I$ by the definition of restriction, and $d \in M$ since gen(d) $\subseteq$ M. This yields $d \in A^I$ iff $d \in A^{I|_M}$.
(2) $C = C_1 \sqcap C_2$ for concept terms C_1 and C_2. By induction we have for i = 1, 2 that $d \in C_i^I$ iff $d \in C_i^{I|_M}$. This yields $d \in (C_1 \sqcap C_2)^I$ iff $d \in (C_1 \sqcap C_2)^{I|_M}$.
(3) The cases $C = C_1 \sqcup C_2$ and $C = \neg D$ can be treated similarly.
(4) $C = \exists L{:}D$ for a concept term D and a regular language L over the alphabet of role names. Assume first that $d \in (\exists L{:}D)^I$, i.e., there exists a word $W = R_1 \ldots R_n \in L$ and an individual $e \in D^I$ such that $dW^I e$. Thus there exist individuals $d_1, \ldots, d_{n-1}$ such that $dR_1{}^I d_1 R_2{}^I d_2 \ldots d_{n-1} R_n{}^I e$. Obviously, d, $d_1, \ldots, d_{n-1}$, and e are elements of gen(d), and thus of M. But then we also have $dW^{I|_M}e$. In addition, $e \in$ gen(d) yields gen(e) $\subseteq$ gen(d) $\subseteq$ M. Thus we can apply the induction hypothesis to D and e, and get $e \in D^{I|_M}$. This shows $d \in (\exists L{:}D)^{I|_M}$.
The other direction can be proved in a similar way.
(5) The case $C = \forall L{:}D$ can be treated similarly. ❑

If we take M = gen(d) in the lemma we get

Proposition 6.2. Let I be an interpretation, C be a concept term of $\mathcal{ALC}_{reg}$, and d be an element of dom(I). Then we have

$$d \in C^{I} \text{ if and only if } d \in C^{I|_{gen(d)}}.$$

We are now ready to show how to internalize concept equations. As mentioned above it is sufficient to consider only one concept equation of the form C = Top. Let **R** denote the finite alphabet of all relevant role names. By $\mathbf{R}^*$ we denote the regular language of all words over **R**.

Theorem 6.3. The concept term D of $\mathcal{ALC}_{reg}$ is consistent w.r.t. the concept equation C = Top if and only if the concept term $D \sqcap \forall\mathbf{R}^*{:}C$ is consistent.

Proof. **(1)** Assume that I is an interpretation such that $C^I = dom(I)$ and $D^I \neq \emptyset$. Let $d \in dom(I)$ be such that $d \in D^I$. In order to prove that $d \in (D \sqcap \forall\mathbf{R}^*{:}C)^I$ it is enough to show $d \in (\forall\mathbf{R}^*{:}C)^I$. Let $W \in \mathbf{R}^*$ and $e \in dom(I)$ be such that dW^Ie. Because $C^I = dom(I)$ we have $e \in C^I$, which completes the proof of the "only-if" part of the theorem.

(2) On the other hand, assume that I is an interpretation such that $(D \sqcap \forall\mathbf{R}^*{:}C)^I \neq \emptyset$. Let $d \in dom(I)$ be such that $d \in (D \sqcap \forall\mathbf{R}^*{:}C)^I$.

By Proposition 6.2 we also have $d \in (D \sqcap \forall\mathbf{R}^*{:}C)^{I|_{gen(d)}}$. In particular, this yields $d \in D^{I|_{gen(d)}}$. It remains to be shown that $I|_{gen(d)}$ satisfies the concept equation C = Top, i.e., that $C^{I|_{gen(d)}} = dom(I|_{gen(d)})$. Let e be an element of $dom(I|_{gen(d)}) = dom(I) \cap gen(d)$. That means that there exists a word $W \in \mathbf{R}^*$ such that dW^Ie. Since $d \in (\forall\mathbf{R}^*{:}C)^I$ we get $d \in C^I$, and by Proposition 6.2 this yields $d \in C^{I|_{gen(d)}}$. This completes the proof of the theorem. ❑

As an immediate consequence of this theorem we get

Corollary 6.4. The algorithm of Section 4 can be used to decide consistency and subsumption of concept terms of $\mathcal{ALC}_{reg}$ w.r.t. finite sets of concept equations.

We have already mentioned above that the (possibly cyclic) T-boxes of $\mathcal{ALC}$ can be seen as specific sets of concept equations. For a cyclic T-box of $\mathcal{ALC}$ the semantics which we have used for our sort equations coincides with what we have called "descriptive semantics" in Section 2.

Corollary 6.5. The algorithm of Section 4 can be used to decide consistency and subsumption with respect to cyclic T-boxes of $\mathcal{ALC}$, provided that these T-boxes are considered with descriptive semantics.

This last corollary shows that the treatment of our transitive (or regular) extension of $\mathcal{ALC}$ does not only provide an alternative to terminological cycles in $\mathcal{ALC}$; it also yields a solution of the consistency and the subsumption problem for cyclic T-boxes of $\mathcal{ALC}$.

7. Conclusion

Augmenting $\mathcal{ALC}$ 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 is demonstrated by the following facts.

First, by adding transitive closure, we are leaving the realm of first order logic. This is so because the transitive closure of roles cannot be expressed in first-order predicate logic (see Aho&Ullman (1979)). Second, the algorithm depends on new methods, namely on the use of results from automata theory, and on a data structure, namely the concept trees, which is more sophisticated than the one used in Schmidt-Schauß&Smolka (1988) or Hollunder et al.(1990). This data structure was necessary for coping with the non-termination problem. Third, adding features (i.e., functional roles) and agreements of feature chains (see Nebel&Smolka (1989)) to $\mathcal{FL}_{\mathrm{reg}}$ would make the subsumption problem undecidable (see Nebel (1990b) where this is shown for cyclic T-boxes of $\mathcal{FL}_0$ considered with gfp-semantics), whereas adding features and agreements was never a problem for the languages considered by Hollunder et al. (see Hollunder&Nutt (1990)).

The expressiveness of $\mathcal{ALC}_{\mathrm{reg}}$ (or equivalently $\mathcal{ALC}_{\mathrm{trans}}$) is also demonstrated by the fact that concept terms of this language can be used to internalize concept equations. For that reason, the algorithm developed in Section 4 can also be used to decide consistency and subsumption w.r.t. concept equations in $\mathcal{ALC}_{\mathrm{reg}}$, and thus also in $\mathcal{ALC}$. In particular, we thus get a solution of the consistency and the subsumption problem for cyclic T-boxes of $\mathcal{ALC}$, provided that these T-boxes are considered with descriptive semantics.

8. References

Aho, A. V., Ullman, J. D. (1979). Universality of Data Retrieval Languages. Proceedings of the *6th ACM Symposium on Principles of Programming Languages*.

Baader, F. (1990a). Terminological Cycles in KL-ONE-based Knowledge Representation Languages. *DFKI Research Report* **RR-90-01**, DFKI, Kaiserslautern.

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

Baader, F. (1990c). A Formal Definition for Expressive Power of Knowledge Representation Languages. Proceedings of the *9th European Conference on Artificial Intelligence*, ECAI-90.

Baader, F., Bürckert, H.-J., Hollunder, B., Nutt, W., Siekmann, J. (1990). Concept Logic. Proceedings of the *Symposium on Computational Logic*, Brussels.

Baader, F., Bürckert, H. J., Nebel, B., Nutt, W., Smolka, G. (1991). On the Expressivity of Feature Logics with Negation, Functional Uncertainty, and Sort Equations. *DFKI Research Report*, forthcoming, DFKI, Kaiserslautern.

Brachman, R. J., Schmolze, J. G. (1985). An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science* **16**.

Brachman, R. J., Pigman-Gilbert, V., Levesque, H. J. (1985). An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts in KRYPTON. Proceedings of the *9th International Joint Conference on Artificial Intelligence*, Los Angeles.

Donini, F. M., Lenzerini, M., Nardi, D., Nutt, W. (1991). The Complexity of Concept Languages. To appear in the Proceedings of the *Second International Conference on Principles of Knowledge Representation and Reasoning*, KR-91.

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

Garey, M. R., Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco.

Hollunder, B., Nutt, W., Schmidt-Schauß, 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. *DFKI Research Report* **RR-90-04**, DFKI, Kaiserslautern.

Kaczmarek, T. S., Mark, W., Sondheimer, N. (1986). Recent Developments in NIKL. Proceedings of the *5th National Conference of the American Association for Artificial Intelligence*, AAAI 86.

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

Levesque, H. J., Brachman, R. J. (1987). Expressiveness and Tractability in Knowledge Representation and Reasoning. *Computational Intelligence* **3**.

MacGregor, R., Bates, R. (1987). The Loom Knowledge Representation Language. *Technical Report* **ISI/RS-87-188**, University of Southern California, Information Science Institute.

Manna, Z. (1974). *Mathematical Theory of Computation.* McGraw-Hill, New York.

Nebel, B. (1987). On Terminological Cycles. *KIT Report* **58**, Technische Universität Berlin, KIT Group.

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

Nebel, B. (1990b). Terminological Cycles: Semantics and Computational Properties. Proceedings of the *Workshop on Formal Aspects of Semantic Networks*, Two Harbors.

Nebel, B. (1990c). Terminological Reasoning is Inherently Intractable. *Artificial Intelligence* **43**.

Nebel, B., Smolka, G. (1989). Representation and Reasoning with Attributive Descriptions. *IWBS Report* **81**, IBM Deutschland, Stuttgart. To appear in Bläsius, K.H. et al. (eds.). Sorts and Types for Artificial Intelligence, Lecture Notes in Artificial Intelligence.

Savage, J. E. (1976). *The Complexity of Computing.* Wiley, New York.

Schmidt-Schauß, M., Smolka, G. (1988). Attributive Concept Descriptions with Unions and Complements. *SEKI Report* **SR-88-21**. To appear in Artificial Intelligence.

Thayse, A.(ed.) (1989). *From Modal Logic to Deductive Databases, Introducing a Logic Based Approach to Artificial Intelligence.* John Wiley & Sons, Chichester.

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
67608 Kaiserslautern
FRG

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse oder per anonymem ftp von ftp.dfki.uni-kl.de (131.246.241.100) unter pub/Publications bezogen werden.
Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far are obtainable from the above address or per anonymous ftp from ftp.dfki.uni-kl.de (131.246.241.100) under pub/Publications.
The reports are distributed free of charge except if otherwise indicated.

DFKI Research Reports

RR-92-48
Bernhard Nebel, Jana Koehler:
Plan Modifications versus Plan Generation: A Complexity-Theoretic Perspective
15 pages

RR-92-49
Christoph Klauck, Ralf Legleitner, Ansgar Bernardi:
Heuristic Classification for Automated CAPP
15 pages

RR-92-50
Stephan Busemann:
Generierung natürlicher Sprache
61 Seiten

RR-92-51
Hans-Jürgen Bürckert, Werner Nutt:
On Abduction and Answer Generation through Constrained Resolution
20 pages

RR-92-52
Mathias Bauer, Susanne Biundo, Dietmar Dengler, Jana Koehler, Gabriele Paul: PHI - A Logic-Based Tool for Intelligent Help Systems
14 pages

RR-92-53
Werner Stephan, Susanne Biundo:
A New Logical Framework for Deductive Planning
15 pages

RR-92-54
Harold Boley: A Direkt Semantic Characterization of RELFUN
30 pages

RR-92-55
John Nerbonne, Joachim Laubsch, Abdel Kader Diagne, Stephan Oepen: Natural Language Semantics and Compiler Technology
17 pages

RR-92-56
Armin Laux: Integrating a Modal Logic of Knowledge into Terminological Logics
34 pages

RR-92-58
Franz Baader, Bernhard Hollunder:
How to Prefer More Specific Defaults in Terminological Default Logic
31 pages

RR-92-59
Karl Schlechta and David Makinson: On Principles and Problems of Defeasible Inheritance
13 pages

RR-92-60
Karl Schlechta: Defaults, Preorder Semantics and Circumscription
19 pages

RR-93-02
Wolfgang Wahlster, Elisabeth André, Wolfgang Finkler, Hans-Jürgen Profitlich, Thomas Rist:
Plan-based Integration of Natural Language and Graphics Generation
50 pages

RR-93-03
Franz Baader, Berhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, Enrico Franconi:
An Empirical Analysis of Optimization Techniques for Terminological Representation Systems
28 pages

RR-93-04
Christoph Klauck, Johannes Schwagereit:
GGD: Graph Grammar Developer for features in CAD/CAM
13 pages

RR-93-05
Franz Baader, Klaus Schulz: Combination Techniques and Decision Problems for Disunification
29 pages

RR-93-06
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux: On Skolemization in Constrained Logics
40 pages

RR-93-07
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux: Concept Logics with Function Symbols
36 pages

RR-93-08
Harold Boley, Philipp Hanschke, Knut Hinkelmann, Manfred Meyer: COLAB: A Hybrid Knowledge Representation and Compilation Laboratory
64 pages

RR-93-09
Philipp Hanschke, Jörg Würtz:
Satisfiability of the Smallest Binary Program
8 Seiten

RR-93-10
Martin Buchheit, Francesco M. Donini, Andrea Schaerf: Decidable Reasoning in Terminological Knowledge Representation Systems
35 pages

RR-93-11
Bernhard Nebel, Hans-Juergen Buerckert:
Reasoning about Temporal Relations:
A Maximal Tractable Subclass of Allen's Interval Algebra
28 pages

RR-93-12
Pierre Sablayrolles: A Two-Level Semantics for French Expressions of Motion
51 pages

RR-93-13
Franz Baader, Karl Schlechta:
A Semantics for Open Normal Defaults via a Modified Preferential Approach
25 pages

RR-93-14
Joachim Niehren, Andreas Podelski,Ralf Treinen:
Equational and Membership Constraints for Infinite Trees
33 pages

RR-93-15
Frank Berger, Thomas Fehrle, Kristof Klöckner, Volker Schölles, Markus A. Thies, Wolfgang Wahlster: PLUS - Plan-based User Support
Final Project Report
33 pages

RR-93-16
Gert Smolka, Martin Henz, Jörg Würtz: Object-Oriented Concurrent Constraint Programming in Oz
17 pages

RR-93-17
Rolf Backofen:
Regular Path Expressions in Feature Logic
37 pages

RR-93-18
Klaus Schild: Terminological Cycles and the Propositional μ-Calculus
32 pages

RR-93-20
Franz Baader, Bernhard Hollunder:
Embedding Defaults into Terminological Knowledge Representation Formalisms
34 pages

RR-93-22
Manfred Meyer, Jörg Müller:
Weak Looking-Ahead and its Application in Computer-Aided Process Planning
17 pages

RR-93-23
Andreas Dengel, Ottmar Lutzy:
Comparative Study of Connectionist Simulators
20 pages

RR-93-24
Rainer Hoch, Andreas Dengel:
Document Highlighting —
Message Classification in Printed Business Letters
17 pages

RR-93-25
Klaus Fischer, Norbert Kuhn: A DAI Approach to Modeling the Transportation Domain
93 pages

RR-93-26
Jörg P. Müller, Markus Pischel: The Agent Architecture InteRRaP: Concept and Application
99 pages

RR-93-27
Hans-Ulrich Krieger:
Derivation Without Lexical Rules
33 pages

RR-93-28
Hans-Ulrich Krieger, John Nerbonne, Hannes Pirker: Feature-Based Allomorphy
8 pages

RR-93-29
Armin Laux: Representing Belief in Multi-Agent Worlds viaTerminological Logics
35 pages

RR-93-33
Bernhard Nebel, Jana Koehler:
Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis
33 pages

RR-93-34
Wolfgang Wahlster:
Verbmobil Translation of Face-To-Face Dialogs
10 pages

RR-93-35
Harold Boley, François Bry, Ulrich Geske (Eds.):
Neuere Entwicklungen der deklarativen KI-Programmierung — *Proceedings*
150 Seiten
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

RR-93-36
Michael M. Richter, Bernd Bachmann, Ansgar Bernardi, Christoph Klauck, Ralf Legleitner, Gabriele Schmidt: Von IDA bis IMCOD: Expertensysteme im CIM-Umfeld
13 Seiten

RR-93-38
Stephan Baumann: Document Recognition of Printed Scores and Transformation into MIDI
24 pages

RR-93-40
Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, Andrea Schaerf:
Queries, Rules and Definitions as Epistemic Statements in Concept Languages
23 pages

RR-93-41
Winfried H. Graf: LAYLAB: A Constraint-Based Layout Manager for Multimedia Presentations
9 pages

RR-93-42
Hubert Comon, Ralf Treinen:
The First-Order Theory of Lexicographic Path Orderings is Undecidable
9 pages

RR-93-45
Rainer Hoch: On Virtual Partitioning of Large Dictionaries for Contextual Post-Processing to Improve Character Recognition
21 pages

DFKI Technical Memos

TM-91-14
Rainer Bleisinger, Rainer Hoch, Andreas Dengel:
ODA-based modeling for document analysis
14 pages

TM-91-15
Stefan Busemann: Prototypical Concept Formation
An Alternative Approach to Knowledge Representation
28 pages

TM-92-01
Lijuan Zhang: Entwurf und Implementierung eines Compilers zur Transformation von Werkstückrepräsentationen
34 Seiten

TM-92-02
Achim Schupeta: Organizing Communication and Introspection in a Multi-Agent Blocksworld
32 pages

TM-92-03
Mona Singh:
A Cognitiv Analysis of Event Structure
21 pages

TM-92-04
Jürgen Müller, Jörg Müller, Markus Pischel, Ralf Scheidhauer:
On the Representation of Temporal Knowledge
61 pages

TM-92-05
Franz Schmalhofer, Christoph Globig, Jörg Thoben:
The refitting of plans by a human expert
10 pages

TM-92-06
Otto Kühn, Franz Schmalhofer: Hierarchical skeletal plan refinement: Task- and inference structures
14 pages

TM-92-08
Anne Kilger: Realization of Tree Adjoining Grammars with Unification
27 pages

TM-93-01
Otto Kühn, Andreas Birk: Reconstructive Integrated Explanation of Lathe Production Plans
20 pages

TM-93-02
Pierre Sablayrolles, Achim Schupeta:
Conlfict Resolving Negotiation for COoperative Schedule Management
21 pages

TM-93-03
Harold Boley, Ulrich Buhrmann, Christof Kremer:
Konzeption einer deklarativen Wissensbasis über recyclingrelevante Materialien
11 pages

DFKI Documents

D-92-21
Anne Schauder: Incremental Syntactic Generation of Natural Language with Tree Adjoining Grammars
57 pages

D-92-22
Werner Stein: Indexing Principles for Relational Languages Applied to PROLOG Code Generation
80 pages

D-92-23
Michael Herfert: Parsen und Generieren der Prolog-artigen Syntax von RELFUN
51 Seiten

D-92-24
Jürgen Müller, Donald Steiner (Hrsg.):
Kooperierende Agenten
78 Seiten

D-92-25
Martin Buchheit: Klassische Kommunikations- und Koordinationsmodelle
31 Seiten

D-92-26
Enno Tolzmann:
Realisierung eines Werkzeugauswahlmoduls mit Hilfe des Constraint-Systems CONTAX
28 Seiten

D-92-27
Martin Harm, Knut Hinkelmann, Thomas Labisch:
Integrating Top-down and Bottom-up Reasoning in COLAB
40 pages

D-92-28
Klaus-Peter Gores, Rainer Bleisinger: Ein Modell zur Repräsentation von Nachrichtentypen
56 Seiten

D-93-01
Philipp Hanschke, Thom Frühwirth: Terminological Reasoning with Constraint Handling Rules
12 pages

D-93-02
Gabriele Schmidt, Frank Peters, Gernod Laufkötter: User Manual of COKAM+
23 pages

D-93-03
Stephan Busemann, Karin Harbusch(Eds.):
DFKI Workshop on Natural Language Systems: Reusability and Modularity - Proceedings
74 pages

D-93-04
DFKI Wissenschaftlich-Technischer Jahresbericht 1992
194 Seiten

D-93-05
Elisabeth André, Winfried Graf, Jochen Heinsohn, Bernhard Nebel, Hans-Jürgen Profitlich, Thomas Rist, Wolfgang Wahlster:
PPP: Personalized Plan-Based Presenter
70 pages

D-93-06
Jürgen Müller (Hrsg.):
Beiträge zum Gründungsworkshop der Fachgruppe Verteilte Künstliche Intelligenz Saarbrücken 29.-30. April 1993
235 Seiten
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-93-07
Klaus-Peter Gores, Rainer Bleisinger:
Ein erwartungsgesteuerter Koordinator zur partiellen Textanalyse
53 Seiten

D-93-08
Thomas Kieninger, Rainer Hoch: Ein Generator mit Anfragesystem für strukturierte Wörterbücher zur Unterstützung von Texterkennung und Textanalyse
125 Seiten

D-93-09
Hans-Ulrich Krieger, Ulrich Schäfer:
TDL ExtraLight User's Guide
35 pages

D-93-10
Elizabeth Hinkelman, Markus Vonerden,Christoph Jung: Natural Language Software Registry
(Second Edition)
174 pages

D-93-11
Knut Hinkelmann, Armin Laux (Eds.):
DFKI Workshop on Knowledge Representation Techniques — Proceedings
88 pages

D-93-12
Harold Boley, Klaus Elsbernd, Michael Herfert, Michael Sintek, Werner Stein:
RELFUN Guide: Programming with Relations and Functions Made Easy
86 pages

D-93-14
Manfred Meyer (Ed.): Constraint Processing – Proceedings of the International Workshop at CSAM'93, July 20-21, 1993
264 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).