Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research
Report
RR-90-10

Concept Logics

Franz Baader, Hans-Jürgen Bürckert,
Bernhard Hollunder,
Werner Nutt, Jörg H. Siekmann

September 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

Concept Logics

Franz Baader, Hans-Jürgen Bürckert, Bernhard Hollunder, Werner Nutt, Jörg H. Siekmann

DFKI-RR-90-10

Concept Logics

Franz Baader, Hans-Jürgen Bürckert, Bernhard Hollunder, Werner Nutt

DFKI Kaiserslautern, Projektgruppe WINO,
Postfach 2080, D-6750 Kaiserslautern, FR Germany
E-mail: {baader, buerkert, hollunde, nutt}@informatik.uni-kl.de

Jörg H. Siekmann

Universität Kaiserslautern, FB Informatik,
Postfach 3049, D-6750 Kaiserslautern, FR Germany
E-mail: siekmann@informatik.uni-kl.de

Abstract: Concept languages (as used in BACK, KL-ONE, KRYPTON, LOOM) are employed as knowledge representation formalisms in Artificial Intelligence. Their main purpose is to represent the generic concepts and the taxonomical hierarchies of the domain to be modeled. This paper addresses the combination of the fast taxonomical reasoning algorithms (e.g. subsumption, the classifier etc.) that come with these languages and reasoning in first order predicate logic. The interface between these two different modes of reasoning is accomplished by a new rule of inference, called *constrained resolution*. Correctness, completeness as well as the decidability of the constraints (in a restricted constraint language) are shown.

Keywords: concept description languages, KL-ONE, constrained resolution, taxonomical reasoning, knowledge representation languages.

Contents

1 Introduction

All knowledge representation (KR) languages in Artificial Intelligence (AI) have one deficiency in common: None of them is adequate to represent every kind of knowledge, but instead the systems that support a particular representation formalism usually specialize on some mode of reasoning, while leaving others out.

This is in particular the case for those knowledge representation languages that are built on some extension of first order logic on the one hand, and the class of concept languages of the KL-ONE family on the other hand. The systems that support either of these two classes of formalisms have highly tuned and specialized inference mechanisms that pertain to the task at hand, but are utterly inefficient (or outright impossible) when it comes to reasoning in a mode that is more typical for the other class. Hence one would like a combination of the KR-formalisms for a more satisfactory representation of the knowledge domain.

What is required are hybrid KR-formalisms that support an adequate representation of different kinds of knowledge chunks. One step in this direction has been made in the field of automated deduction, where sorted logics have been proposed, i.e., an extension of standard predicate logic (PL1) with so-called sort hierarchies (Frisch, 1989; Walther, 1987; Schmidt-Schauß, 1989; Beierle et al., 1990; Weidenbach & Ohlbach, 1990; Cohn, 1987). Sort hierarchies are set description languages with a relatively weak expressiveness[1], but they turned out to provide a powerful extension of predicate logic that can still be based on the Resolution Principle (Robinson, 1965). It has been shown that sort hierarchies – or their operationalization via sort resolution –allow a substantial pruning of the search space by replacing certain deduction sequences with a deterministic computation of the subset relationships (Frisch, 1986; Walther, 1988). The essential idea is this: based on work carried out in the logic community in the 1930's and 40's (Herbrand, 1930; Oberschelp, 1962; Schmidt, 1938, 1951) the universe of discourse (the domain for the interpretation) is partitioned into several classes (the sorts) in the case of many sorted logics, which again may have subsorts in the case of order sorted logics. Instead of a unary predicate that expresses this information, sort symbols are attached to variables, constants etc. For example the fact that all natural numbers are either even or odd can be expressed as:

$$\forall x.\ NAT(x) \text{ implies } EVEN(x) \text{ or } ODD(x)$$

or in a sorted logic as

$$\forall x{:}NAT\ .\ EVEN(x) \text{ or } ODD(x).$$

Since there is a general translation (called relativization) between these variants of logic and since also the expressive power is not principally enhanced but remains that of a first order language, interest in the logic community faded by the mid sixties.

[1] Sorts are interpreted as sets, sort hierarchies specify the subset relationship.

This work was rediscovered however by workers in the field of automated reasoning (Walther 1987), when it became apparent that although the expressive power does not increase in principle, there is still an enormous difference in practice, when it comes to actually building an inference engine for the respective logics.

The same general observation holds for current concept description languages in AI and their relationship to predicate logic: while they do not enhance the expressiveness of PL1 in principle, they definitely advance the state of the art, when it comes to actually reason – i.e. draw conclusions – in their respective formalisms.

Concept description formalisms of the KL-ONE family (Brachman & Schmolze, 1985; Brachman & Levesque, 1985; Vilain, 1985; MacGregor & Bates, 1987; Nebel, 1990) are more expressive than mere sort hierarchies. They provide terminologies of concepts or sets whose underlying taxonomical hierarchies determine subset relationships quite similar to sort hierarchies. But in contrast to sort hierarchies concept languages allow far more complex constructs for set descriptions themselves. We shall show one possible combination of this terminological reasoning with that of standard predicate logic, based on the idea to enhance PL1 with concept descriptions, which are then taken as constraints for the variables occurring in the formulae – just as in the above mentioned approaches to sorted logics. However, there are two main differences: The first is that these constraints have a more complex structure: sets are not just denoted by sort symbols as in sorted logics, but they are denoted by complex *concept description terms*. The second difference is that instead of simple sort hierarchies we now have more complicated taxonomies, which do not necessarily possess generic models (least Herbrand models) as in the case of ordinary sort hierarchies (Frisch, 1989).

This paper addresses the combination of the fast taxonomical reasoning algorithms (e.g. subsumption, the classifier etc.) that come with these languages and reasoning in first order predicate logic. The interface between these two different modes of reasoning is accomplished by a constrained-based modification of the Resolution Principle. This can be seen as a generalization of M. Stickel's theory resolution principle (Stickel, 1985) on the one hand and of the ideas of constrained logic programming (Höhfeld & Smolka, 1988; Jaffar & Lassez, 1987) on the other hand, and is worked out in (Bürckert, 1990).

2 Constrained Resolution

The resolution principle elaborates the idea that we can infer the *resolvent* formula $B \vee C$, from the *parent* formulae $A \vee B$ and $\neg A \vee C$. Here the formulae have to be clauses, i.e., universally quantified disjunctions of literals. The A's of the two parent formulae are complementary literals, and in order to obtain the resolvent we have to *unify* corresponding arguments of those literals and apply the *unifying substitution* to the resolvent, for example:

$$\frac{\begin{array}{l} P(s_1,\ldots,s_n) \vee B \\ \neg P(t_1,\ldots,t_n) \vee C \end{array}}{\sigma(B \vee C)} \quad \textit{if there is some } \sigma \textit{ with } \sigma s_i = \sigma t_i\ (1 \le i \le n)$$[2]

An analysis of Robinson's soundness and completeness proof for resolution (Robinson, 1965), provides an argument that the computation of a unifying substitution can be replaced by a unifiability test, provided we add a *constraint* consisting of the conjunction of the term equations $s_i = t_i\ (1 \le i \le n)$ to the resolvent (instead of instantiating it with the substitution)[3].

A slight modification of this approach transformes the clauses into homogeneous form, where the argument terms of the literals are replaced by new variables. Then constraints are attached to the clauses, that specify the equality of these new variables with the substituted terms. For example

$$P(b, x) \vee Q(a, f(x, b), g(z))$$

is replaced by

$$P(x_1, x_2) \vee Q(x_3, x_4, x_5) \ \|\ x_1=b, x_2=x, x_3=a, x_4=f(x, b), x_5=g(z).$$

Now a resolution step takes two clauses with such *equational constraints* and generates a resolvent with a new unified equational constraint.

Noticing that unifiability of equations is the same as the satisfiability of the existential closure of these equations in the (ground) term algebra, we obtain a more general view: clauses might have some arbitrary, not necessarily equational constraint for their variables. A resolvent of two clauses generates a new constraint that is unsatisfiable whenever one of the parents' constraints is unsatisfiable. Here the derived new constraint is a conjunction of the old ones after an identification of the corresponding variables. Let Γ and Γ' denote constraints that are separated from their corresponding clause by $\|$, then we denote constrained resolution as:

$$\frac{\begin{array}{l} P(x_1,\ldots,x_n) \vee C \ \|\ \Gamma \\ \neg P(y_1,\ldots,y_n) \vee C' \ \|\ \Gamma' \end{array}}{C \vee C' \ \|\ \Gamma \wedge \Gamma'_{[y_i = x_i\,(1\le i\le n)]}} \quad \textit{if } \Gamma \wedge \Gamma'_{[y_i = x_i\,(1\le i\le n)]} \textit{ is satisfiable}$$

The satisfiability of the constraint is defined with respect to a distinct constraint theory.

As in the classical case, the overall task is now to prove the unsatisfiability of a given set of constrained clauses, where we have to take into account that the constraints are interpreted with respect to the given constraint theory. This constraint theory could be any set of models – either given by a set of axioms or in any other way. In the classical case for instance, the

[2] This inference rule has to be read as follows: From the formulae above the line infer the formulae below the line provided the condition can be satisfied.

[3] Huet (Huet, 1972) in fact used this trick in order to have a resolution rule for higher order logics. He already used the name "constrained resolution" for this rule.

constraint theory might be thought of as a singleton set containing just the ground term algebra, i.e., the Herbrand universe.

Let us introduce these notions of constraint theories and constrained clauses more formally. A ***constraint theory*** consists of a set of ***constraint symbols*** (or constraints) and a set of ***constraint models***, in which these constraint symbols are interpreted (cf. also Höhfeld & Smolka, 1988; Smolka, 1989). Given an infinite set $\mathcal{V}$ of variables we assume that every constraint Γ comes with a finite set of variables $Var(\Gamma) = \{x_1,\ldots,x_n\}$. In this case the constraint is often denoted by $\Gamma(x_1,\ldots,x_n)$. Each model $\mathcal{A}$ consists of a nonempty set $D^{\mathcal{A}}$, the domain, and for each constraint symbol Γ it contains a set $\Gamma^{\mathcal{A}}$ of ***$\mathcal{A}$-solutions***, i.e., of assignments $\sigma\colon \mathcal{V} \to D^{\mathcal{A}}$. Up to now a constraint theory is nothing but a set of first order structures over a (possibly infinite) signature of predicate symbols, namely the constraint symbols. The set of n-tuples, given by the $\mathcal{A}$-solution values of the variables of an n-ary constraint, is just the denotation of the corresponding predicate symbol in the first order structure $\mathcal{A}$. However, for the constrained resolution rule we need in addition that the set of constraints has to be closed under conjunction, under renaming[4] and under identification of variables. That is, for every two constraints there must be a conjunct constraint, whose solution sets are the intersections of its constituents. Also for each constraint and each variable substitution, which renames or identifies variables, there must be a variant or an instance constraint, whose solutions are obtained by renaming the former solutions accordingly or by selecting just those solutions that assign the same value to identified variables. To be more precise: we assume that for every two constraints Γ_1 and Γ_2 there exists their conjunct constraint $\Gamma_1 \wedge \Gamma_2$ such that $(\Gamma_1 \wedge \Gamma_2)^{\mathcal{A}} = \Gamma_1^{\mathcal{A}} \cap \Gamma_2^{\mathcal{A}}$ for every constraint model $\mathcal{A}$; and we assume that for every constraint Γ and for every (renaming or identifying) variable substitution $\varphi\colon \mathcal{V} \to \mathcal{V}$ there exists the (variant or instance) constraint $\varphi\Gamma$ such that $(\varphi\Gamma)^{\mathcal{A}} = \{\sigma^* \mid \sigma \in \Gamma^{\mathcal{A}} \text{ with } \sigma x = \sigma y \text{ if } \varphi x = \varphi y\}$ for every constraint model $\mathcal{A}$, where σ^* is defined by $\sigma^*\varphi x := \sigma x$ for each variable $\varphi x \in \varphi(\mathcal{V})$, and $\sigma^* z := \sigma z$ for all other variables z.

We now want to extend constraint theories with further predicate symbols. Thus given a constraint theory and a set $\mathcal{P}$ of predicate symbols –disjoint from the set of constraint symbols – we augment the constraint models by any possible denotation for these new symbols: An ***$\mathcal{R}$-structure*** is given by a constraint model, where in addition each n-ary predicate symbol $P \in \mathcal{P}$ denotes an n-ary relation $P^{\mathcal{A}}$ on $D^{\mathcal{A}}$.

For technical reasons we do not allow arbitrary formulae over these predicate symbols as there might be problems transforming them into constrained clauses, depending on the constraint theory (Bürckert, 1990). Thus we restrict ourselves to formulae that are already in (constrained) clause form, i.e., to sets of constrained clauses.

An ***atom*** is a predicate symbol followed by a list of variables: $P(x_1,\ldots,x_n)$. A literal is an atom or its negation: $\neg P(x_1,\ldots,x_n)$. A ***constrained clause*** is a pair consisting of a set C

[4] As in the classical case clauses have always to be variable disjoint, which requires a renaming of the generated resolvents – and hence also of their constraints – before they can be added to the clause set.

of literals (called the kernel or the matrix of the clause) and a constraint Γ. Constrained clauses are written $C \,||\, \Gamma$.

The semantics of constrained clauses are defined as follows.

Let $\mathcal{A}$ be an $\mathcal{R}$-structure, let α be an $\mathcal{A}$-assignment, let P be a predicate symbol, and let C be a set of literals, i.e., the kernel of a constrained clause. Then

- $(\mathcal{A}, \alpha) \models P(x_1,\dots,x_n)$ iff $(\alpha x_1,\dots,\alpha x_n) \in P^{\mathcal{A}}$
- $(\mathcal{A}, \alpha) \models \neg P(x_1,\dots,x_n)$ iff $(\alpha x_1,\dots,\alpha x_n) \notin P^{\mathcal{A}}$
- $(\mathcal{A}, \alpha) \models C$ iff $(\mathcal{A}, \alpha) \models L$ for some literal L in the set C

Let $C||\Gamma$ be a constrained clause. Then

- $\mathcal{A} \models C\,||\,\Gamma$ iff $(\mathcal{A}, \alpha) \models C$ for each $\mathcal{A}$-solution α of Γ.

Note that the constrained clause is also satisfied by the $\mathcal{R}$-structure, if the constraint has no $\mathcal{A}$-solution. In particular, a constrained clause, whose constraint does not have an $\mathcal{A}$-solution in any $\mathcal{R}$-structure $\mathcal{A}$, is a tautology with respect to the constraint theory.

We call a set of constrained clauses ***$\mathcal{R}$-satisfiable*** iff there is an $\mathcal{R}$-structure $\mathcal{A}$ such that each of the clauses is satisfied by the structure; otherwise the clause set is called ***$\mathcal{R}$-unsatisfiable***.

Not every constraint needs to have solutions, but in order to specify the derivation rules for proving the unsatisfiability of a constrained clause set our main interest lies in constraints that have solutions – at least in one constraint model. A constraint Γ is called ***satisfiable*** or solvable iff there is at least one model $\mathcal{A}$, where the $\mathcal{A}$-solution set of the constraint is not empty. We have the following ***constrained resolution rule*** for a pair of variable disjoint constrained clauses:

$$\frac{\begin{array}{c}\{P(x_{11},\dots,x_{1n}),\dots,\ P(x_{k1},\dots,x_{kn})\} \cup C \;||\Gamma \\ \{\neg P(y_{11},\dots,y_{1n}),\dots,\neg P(y_{m1},\dots,y_{mn})\} \cup C'||\Gamma'\end{array}}{C \cup C'||\varphi(\Gamma \wedge \Gamma')} \quad \text{if } \varphi(\Gamma \wedge \Gamma') \text{ is satisfiable}$$

where C and C' are the remaining parts of the two clauses and the new constraint $\varphi(\Gamma \wedge \Gamma')$ is a conjunction instantiated by a variable substitution φ that identifies corresponding variables of the complementary literals: $\varphi x_{1i} = \dots = \varphi x_{ki} = \varphi y_{1i} = \dots = \varphi y_{mi}$ $(1 \leq i \leq n)$. The derived clause is called a ***resolvent*** of the two parent clauses, its variables have to be renamed, in order to keep the clauses' variables disjoint. This whole operation on clause sets is called a ***resolution step***. A ***derivation*** is a (possibly infinite) sequence of resolution steps.

On closer inspection we notice that the unsatisfiability of a constrained clause set cannot be proved by just deriving the empty clause, as in the classical resolution calculus. The reason is that such an empty clause might still have some constraints, which are only satisfied by *some* of the constraint models, but not by all of them. For classical resolution the derivation of the empty clause could be seen as a derivation of *false* from the starting clause set, but in the constraint resolution framework this provides just a derivation of *false* within those models that satisfy the constraints of the empty clause. The solution is to construct for each constraint model a suitable empty clause whose restriction is satisfied by that model (Bürckert, 1990). We call a set of (satisfiable) constraints ***valid*** iff for each constraint model $\mathcal{A}$ at least one of the constraints in the set has $\mathcal{A}$-solutions. With this notion we now define: A ***refutation*** is a derivation, such that the set of constraints of all derived empty clauses is valid. However, this is not an operational definition: we need terminating refutations, so we have to restrict the constraint theories to those where only a finite number of such constrained empty clauses is needed. A constraint theory is ***compact*** iff every valid set of constraints contains a finite subset, which is again valid. For compact theories every infinite refutation contains a finite subsequence of resolution steps that is a refutation.

The following completeness result for constrained resolution can be proved by a suitable generalization of the standard completeness proof for classical resolution (Bürckert, 1990):

Theorem: (Soundness and completeness of constrained resolution)
Let $\mathcal{R}$ be a (compact) constraint theory. A set $\mathcal{C}$ of constrained clauses is $\mathcal{R}$-unsatisfiable iff there exists a (finite) refutation of $\mathcal{C}$.

Any theory over a given first order language – given as a set of first order structures over this language – can be seen as a constraint theory, where the open first order formulae play the role of constraints. Such a constraint is satisfiable iff its existential closure is satisfiable by some of the models of the theory. The theory is a compact constraint theory iff it has an axiomatization by a set of first order formulae (this is an easy consequence of the compactness theorem of predicate logic and the above definition of validity of constraints). Obviously a finite set of constraints is valid iff the existential closure of the disjunction of its elements is a theorem of the theory, i.e. it is satisfied by each model of the theory.

Many concept languages are known to provide a special class of theories with first order axiomatizations, such that it is decidable if a concept description denotes a nonempty set in at least one of the models of that theory. It turns out that this test is essentially the basis of the satisfiability test we need in our constrained resolution steps. We shall see that the validity of a set of concept constraints is also decidable, for a reasonable class of concept languages, to be defined below.

3 Concept Languages

Concept languages were introduced by (Brachman & Schmolze, 1985) for the formalization and structuring of semantic networks (Quillian, 1968). In particular, they are used to represent the taxonomical and conceptual knowledge of a particular problem domain. To describe this kind of knowledge, one starts with given atomic concepts and roles, and defines new concepts and roles[5] using the operations provided for by the concept language. Concepts can then be considered as unary predicates which are interpreted as sets of individuals and roles as binary predicates which are interpreted as binary relations over individuals.

Examples for atomic concepts may be person and female, examples for roles may be child and friend. If logical connectives like conjunction, disjunction, and negation are present as language constructs, one can describe the concept of a man as those "persons who are not female" and represent it by the expression (person $\sqcap \neg$ female), where $\sqcap$ is the conjunction and $\neg$ is the negation operator provided by the concept language. Conjunction, disjunction, and negation are interpreted as set intersection, union, and complement. Most languages provide quantification over roles that allows for instance to describe the concepts of "individuals having a female child" and "individuals for which all children are female" by the expressions $\exists$child.female and $\forall$child.female, respectively. Number restrictions on roles denote sets of individuals having at least or at most a certain number of fillers for a role. For instance, ($\geq$ 3 friend) $\sqcap$ ($\leq$ 2 child) can be read as "all individuals having at least three friends and at most two children."

Concept languages, e.g., $\mathcal{FL}$ and $\mathcal{FL}^-$ (Levesque & Brachman, 1987), $\mathcal{TF}$ and $\mathcal{NTF}$ (Nebel, 1990), or the $\mathcal{AL}$-languages considered in (Schmidt-Schauß & Smolka, 1988; Hollunder & Nutt, 1990), differ in what kind of constructs are allowed for the definition of concepts and roles. Their common feature – besides the use of concepts and roles – is that the meaning of the constructs is defined by a model-theoretic semantics.

In this section we show how to conceive a concept language as a constraint theory. For this purpose we restrict our attention to the concept language $\mathcal{ALC}$ (Schmidt-Schauß & Smolka, 1988). This language offers language constructs for conjunction, disjunction, and negation of concepts as well as role quantification. It would also be possible to use an extension of $\mathcal{ALC}$, e.g. $\mathcal{ALC}$ amalgamated with number restrictions, but this would burden the presentation without any new insight.

We assume a language with two disjoint alphabets of symbols, ***concept symbols*** and ***role symbols***. We have two special concept symbols $\top$ (***top symbol***) and $\bot$ (***bottom symbol***). The set of ***concept descriptions*** of $\mathcal{ALC}$ is inductively defined:

- ➤ every concept symbol is a concept description (***atomic description***).

Now let C and D be concept descriptions already defined and let R be a role symbol. Then

- ➤ $C \sqcap D$ (***conjunction***), $C \sqcup D$ (***disjunction***) and $\neg C$ (***negation***) are concept descriptions, and

[5]In this paper we do not consider the definition of complex roles. See (Baader, 1990; Hollunder & Nutt, 1990) for a discussion of role-forming constructs such as the transitive closure or intersection of roles.

- $\forall R.C$ (***value restriction***) and $\exists R.C$ (***exists restriction***) are concept descriptions.

An ***interpretation*** I of a concept description consists of a nonempty set $\top^I$ (the ***domain*** of I), and a function $\bullet^I$ (the ***interpretation function*** of I). The interpretation function maps every concept symbol A to a subset A^I of $\top^I$ and every role symbol R to a subset R^I of $\top^I \times \top^I$. The interpretation function - which gives an interpretation for concept and role symbols - will be extended to arbitrary concept descriptions as follows. Let C and D be concept descriptions and let R be a role symbol. Assume that C^I and D^I is already defined. Then

- $(C \sqcap D)^I \quad := \quad C^I \cap D^I$
- $(C \sqcup D)^I \quad := \quad C^I \cup D^I$
- $(\neg C)^I \quad := \quad \top^I \setminus C^I$
- $(\forall R.C)^I \quad := \quad \{a \in \top^I \mid \forall b\colon (a, b) \in R^I \Rightarrow b \in C^I\}$
- $(\exists R.C)^I \quad := \quad \{a \in \top^I \mid \exists b\colon (a, b) \in R^I \wedge b \in C^I\}$.

In KL-ONE-based knowledge representation systems such as KL-ONE (Brachman & Schmolze, 1985), BACK (Nebel, 1990), KRYPTON (Brachman et al., 1985), LOOM (MacGregor & Bates, 1987), concept languages are used to describe a ***terminology***, i.e., the taxonomical knowledge. Starting with concepts such as person and female one can describe "persons that are not female" by the expression (person $\sqcap$ ¬female). If we want to use this expression in other concept descriptions it is appropriate to define the ***terminological axiom***

$$\text{man} = \text{person} \sqcap \neg\text{female}$$

where man is a concept. If child is a role, we can describe "not female persons with only female children" by the expression (man $\sqcap$ ∀child.female). Terminological axioms allow to define abbreviations for concept descriptions, and hence help to keep concept descriptions simple. However, for reason of simplicity of presentation we do not consider terminological axioms. Thus we assume that every concept in a concept description is completely undefined and not an abbreviation for another concept description.

A concept description C is called ***satisfiable*** iff there exists an interpretation I such that C^I is nonempty, and is called ***universally satisfiable*** iff there exists an interpretation I such that $C^I = \top^I$. We say D is subsumed by C if $D^I \subseteq C^I$ for every interpretation I, and C is equivalent to D if $C^I = D^I$ for every interpretation I.

For our view of concept languages as constraint theories, we are only interested in satisfiability and universal satisfiability of concept descriptions. KL-ONE-based knowledge representation systems offer in addition to a concept language an assertional component for the introduction of objects, which are instances of concept descriptions and roles. In this case there are notions as consistency of the knowledge base, instantiation, realization and retrieval. Definitions of these and relationships between them can be found in (Nebel, 1990; Hollunder, 1990).

We shall now describe how to view a concept language as a constraint theory using concept descriptions to define socalled concept constraints. Let $\mathcal{V}$ be an infinite set of variables. An ***atomic concept constraint*** is of the form $x{:}C$ where x is a variable and C is a concept description. A ***concept constraint*** $\Gamma(x_1,\ldots,x_n)$ is a finite set $\{x_1{:}C_1,\ldots, x_n{:}C_n\}$ of atomic concept constraints.

A ***solution*** of a concept constraint $\Gamma = \{x_1{:}C_1,\ldots, x_n{:}C_n\}$ in an interpretation $\mathcal{I}$ is an $\mathcal{I}$-assignment $\alpha\colon \mathcal{V} \to \mathsf{T}^{\mathcal{I}}$ such that $\alpha x_i \in C_i^{\mathcal{I}} (1 \leq i \leq n)$. A concept constraint Γ is ***satisfiable*** iff there exists an interpretation $\mathcal{I}$ and a solution of Γ in $\mathcal{I}$.

Let Γ be a concept constraint. Without loss of generality we can assume that the variables occurring in the atomic concept constraints of Γ are pairwise disjoint. To see this, suppose that $\Gamma = \{x_1{:}C_1,\ldots, x_n{:}C_n\}$ and $x_i = x_j$ for some $i \neq j$. Then we create a new concept constraint $\Gamma' := \Gamma \setminus \{x_i{:}C_i, x_j{:}C_j\} \cup \{x_i{:}(C_i \sqcap C_j)\}$. Obviously, Γ is satisfiable if and only if Γ' is satisfiable. This process is iterated until we obtain a concept constraint such that variables occurring in the atomic concept constraints are pairwise disjoint.

Now consider the constrained resolution rule: In order to make the application of the resolution step effective, we need an algorithm that decides whether the conjunction of the constraints of the parent clauses is satisfiable. In the following proposition we relate the satisfiability of concept descriptions to the satisfiability of concept constraints.

Proposition 3.1:

A concept constraint $\Gamma = \{x_1{:}C_1,\ldots, x_n{:}C_n\}$ with pairwise different variables is satisfiable if and only if every concept description C_i is satisfiable.

Proof: ($\Rightarrow$) If $\Gamma = \{x_1{:}C_1,\ldots, x_n{:}C_n\}$ is satisfiable, then there exists an interpretation $\mathcal{I}$ such that every $C_i^{\mathcal{I}}$ is nonempty. Hence every C_i is satisfiable.

($\Leftarrow$) Let $\Gamma = \{x_1{:}C_1,\ldots, x_n{:}C_n\}$ be a concept constraint such that $x_i \neq x_j$ for $i \neq j$. Suppose that every C_i is satisfiable. Then there exists for every C_i an interpretation $\mathcal{I}_i$ such that $C_i^{\mathcal{I}_i}$ is nonempty. Without loss of generality we can assume that $\mathsf{T}^{\mathcal{I}_i} \cap \mathsf{T}^{\mathcal{I}_j} = \emptyset$ for $i \neq j$. Now, let $\mathsf{T}^{\mathcal{I}} := \bigcup_{1\leq i\leq n} \mathsf{T}^{\mathcal{I}_i}$, $A^{\mathcal{I}} := \bigcup_{1\leq i\leq n} A^{\mathcal{I}_i}$ for every concept symbol A, and let $R^{\mathcal{I}} := \bigcup_{1\leq i\leq n} R^{\mathcal{I}_i}$ for every role symbol R. Then $\mathcal{I}$ is an interpretation and it is easy to see that Γ has a solution in $\mathcal{I}$. Hence Γ is satisfiable. ❑

The problem of checking the satisfiability of concept descriptions is considered in (Schmidt-Schauß & Smolka, 1988; Hollunder & Nutt, 1990), where it is shown that checking the satisfiability of concept descriptions of the $\mathcal{ALC}$-language is a decidable, PSPACE-complete problem.

In order to decide whether we have reached a refutation we need an algorithm that decides the validity of a finite set of concept constraints (cf. the completeness theorem of constrained resolution). We show that this can be reduced to decide the universal satisfiability of concept descriptions.

Proposition 3.2:

A set $\{\Gamma_1,\ldots,\Gamma_n\}$ of concept constraints with pairwise disjoint variables is valid iff for each sequence $(x_1{:}C_1,\ldots,x_n{:}C_n)$ with $x_i{:}C_i \in \Gamma_i$ $(1 \leq i \leq n)$ and each interpretation $\mathcal{I}$ some C_i has a solution in $\mathcal{I}$.

Proof: $(\Leftarrow)$ We show that $\{\Gamma_1,\ldots,\Gamma_n\}$ is valid: Let $\mathcal{I}$ be any interpretation. Assume that none of the concept constraints is satisfiable in $\mathcal{I}$. Then each Γ_i must contain an atomic constraint $x_i{:}C_i$ without a solution in $\mathcal{I}$. This contradicts the precondition.

$(\Rightarrow)$ Assume by contradiction that there is a sequence $(x_1{:}C_1,\ldots,x_n{:}C_n)$ with $x_i{:}C_i \in \Gamma_i$ $(1 \leq i \leq n)$ and an interpretation $\mathcal{I}$ such that $C_i^{\mathcal{I}} = \emptyset$ for each i $(1 \leq i \leq n)$. We show that $\{\Gamma_1,\ldots,\Gamma_n\}$ cannot be valid. Assume by contradiction that some Γ_{i_0} has a solution in $\mathcal{I}$. Then by Proposition 3.1 all its concept descriptions are satisfiable in $\mathcal{I}$, and hence especially $(C_{i_0})^{\mathcal{I}} \neq \emptyset$, which yields a contradiction. ❑

Now, let $C_1,\ldots,C_n$ be concept descriptions and let $\mathcal{I}$ be any interpretation. Then obviously $(\neg C_1 \sqcap \ldots \sqcap \neg C_n)^{\mathcal{I}} = \top^{\mathcal{I}}$ iff $C_i^{\mathcal{I}} = \emptyset$ for each i $(1 \leq i \leq n)$. Together with Proposition 3.2 this implies the following theorem:

Theorem 3.3:

A set $\{\Gamma_1,\ldots,\Gamma_n\}$ of concept constraints with pairwise disjoint variables is not *valid iff there is a sequence $(x_1{:}C_1,\ldots,x_n{:}C_n)$ with $x_i{:}C_i \in \Gamma_i$ $(1 \leq i \leq n)$ such that the concept $\neg C_1 \sqcap \ldots \sqcap \neg C_n$ is universally satisfiable.*

The above result shows that we can decide validity of a finite set $\{\Gamma_1,\ldots,\Gamma_n\}$ of concept descriptions as follows: We have to test if the concept description $\neg C_1 \sqcap \ldots \sqcap \neg C_n$ is universally satisfiable for all possible choices $(x_1{:}C_1,\ldots,x_n{:}C_n)$ collecting single atomic constraints $x_i{:}C_i$ from each of the concept constraints Γ_i. The set of concept descriptions is valid, iff none of these tests returns the answer *Yes*. In the following we will show how we can decide this universal satisfiability (section 4.2). A slight modification of this algorithm provides an algorithm for the satisfiability test for concept descriptions (section 4.3).

4 Proving Universal Satisfiability and Satisfiability of Concept Descriptions

Schmidt-Schauß and Smolka (1988) give an algorithm which checks the satisfiability of concept descriptions and we shall demonstrate by an example how this algorithm works.

Given a concept description C this algorithm essentially tries to construct an interpretation which interprets C as a nonempty set. If this process fails, i.e., if a contradiction occurs, the concept description is not satisfiable; otherwise the algorithm generates such an interpretation.

Let us demonstrate this by an example: In order to show that the concept description $C = \exists R.A \sqcap \forall R.B$ is valid, we want to construct a finite interpretation $\mathcal{I}$ such that $C^{\mathcal{I}}$ is a nonempty set, i.e., there exists an element in $\top^{\mathcal{I}}$ which is an element of $C^{\mathcal{I}}$. The algorithm generates such an element a and imposes the constraint $a \in C^{\mathcal{I}}$ on it. Obviously, this yields that $a \in (\exists R.A)^{\mathcal{I}}$ and $a \in (\forall R.B)^{\mathcal{I}}$. From $a \in (\exists R.A)^{\mathcal{I}}$ we can deduce that there has to exist an element b such that $(a, b) \in R^{\mathcal{I}}$ and $b \in A^{\mathcal{I}}$. Thus the algorithm introduces a new element b and constrains it as mentioned. Since $a \in (\forall R.B)^{\mathcal{I}}$ and $(a, b) \in R^{\mathcal{I}}$, we also get the constraint $b \in B^{\mathcal{I}}$. We have now satisfied all the constraints in the example without getting a contradiction. This shows that $C^{\mathcal{I}}$ is satisfiable. We have generated an interpretation $\mathcal{I}$ which interprets C as a nonempty set: $\top^{\mathcal{I}} = \{a, b\}$, $A^{\mathcal{I}} = B^{\mathcal{I}} = \{b\}$, and $R^{\mathcal{I}} = \{(a, b)\}$. Termination of the algorithm is ensured by the fact that the newly introduced constraints are always smaller than the constraints which enforced their introduction. Note that a constraint $a \in (C \sqcap D)^{\mathcal{I}}$ forces to generate two new constraints $a \in C^{\mathcal{I}}$ and $a \in D^{\mathcal{I}}$. On the other hand, if we have a constraint $a \in (C \sqcup D)^{\mathcal{I}}$, then we have to choose either $a \in C^{\mathcal{I}}$ or $a \in D^{\mathcal{I}}$.

Now let us consider an algorithm which checks the universal satisfiability of concept descriptions. Suppose $C = \exists R.A \sqcap \forall R.B$ is universally satisfiable. We try to construct an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} = \top^{\mathcal{I}}$. Since $\top^{\mathcal{I}}$ is a nonempty set, there has to exist an element in $\top^{\mathcal{I}}$, and hence in $C^{\mathcal{I}}$. Thus the algorithm generates such an element a and imposes the constraint $a \in C^{\mathcal{I}}$ on it. Furthermore, as argued above, the algorithm creates the constraints $(a, b) \in R^{\mathcal{I}}$, $b \in A^{\mathcal{I}}$, $b \in B^{\mathcal{I}}$ where b is a newly introduced element. Since b is an element of $\top^{\mathcal{I}}$, we have to impose the additional constraint $b \in C^{\mathcal{I}}$ on b. Again, because $b \in C^{\mathcal{I}}$, there exists an element $c \in \top^{\mathcal{I}}$ which satisfies the constraints $(b, c) \in R^{\mathcal{I}}$, $c \in A^{\mathcal{I}}$, $c \in B^{\mathcal{I}}$, $c \in C^{\mathcal{I}}$. Obviously, the algorithm that tries to construct an interpretation $\mathcal{I}$ with $C^{\mathcal{I}} = \top^{\mathcal{I}}$ would run forever creating more and more new elements. However, if we set $c = b$, then we have $\top^{\mathcal{I}} = \{a, b\}$, $A^{\mathcal{I}} = B^{\mathcal{I}} = \{b\}$, $R^{\mathcal{I}} = \{(a, b), (b, b)\}$, and hence $C^{\mathcal{I}} = \top^{\mathcal{I}}$. Thus, the main problem in extending the satisfiability algorithm to a universal satisfiability algorithm is to find an appropriate criterion for termination. This will be done with the help of so-called concept trees, which are introduced in the next subsection.

This section is organized as follows: In subsection 4.1 we state some basic definitions. The main result, an algorithm for deciding the universal satisfiability of concept descriptions, is presented in subsection 4.2. In subsection 4.3 we slightly modify this algorithm to obtain an algorithm for checking the satisfiability of concept descriptions.

4.1 Basic Definitions

To keep our algorithms simple, we single out a special class of concept descriptions as normal forms. We say a concept description C is ***equivalent*** to D iff $C^I = D^I$ for every interpretation I. A concept description C is called ***simple*** iff C is a concept symbol, or a complemented concept symbol, or if C is of the form $\forall R.D$ or $\exists R.D$. A ***conjunctive*** concept description has the form $C_1 \sqcap \ldots \sqcap C_n$ where each C_i is a simple concept description. 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 descriptions in the form

$$A_1 \sqcap \ldots \sqcap A_m \sqcap \exists R_1.E_1 \sqcap \ldots \sqcap \exists R_l.E_l \sqcap \forall S_1.D_1 \sqcap \ldots \sqcap \forall S_k.D_k.$$

This concept description contains a ***clash*** iff there exist A_i and A_j such that $A_i = \neg A_j$, and contains an ***exists-restriction*** iff $l > 0$. A ***disjunctive*** concept description has the form $C_1 \sqcup \ldots \sqcup C_n$ where each C_i is a conjunctive concept description. A disjunctive concept description which is equivalent to a concept description C is called a ***disjunctive normal form*** for C.

Every concept description can be transformed into a disjunctive normal form. This transformation can be performed as follows: First, we compute the ***negation normal form*** of a concept description, that is, we bring the negation signs immediately in front of concept symbols by rewriting the concept description via de Morgan's laws and with rules $\neg\forall R.C \Rightarrow \exists R.\neg C$ and $\neg\exists R.C \Rightarrow \forall R.\neg C$. Then we transform this concept description into a disjunctive normal form by applying the associativity, commutativity, idempotency and distributivity laws of conjunction and disjunction.

We now define concept trees, which are used to impose a control structure on the algorithm. A ***directed graph*** $G = (N, E)$ consists of a (not necessarily finite) set of nodes N and a set of edges $E \subseteq N \times N$. A ***path*** in a directed graph is a sequence $N_1, \ldots, N_k$ of nodes such that (N_i, N_{i+1}) is an edge for each i, $1 \le i < k$. Notice that paths contain at least two different nodes. We say that this path is a path ***from*** N_1 ***to*** N_k, N_1 is a ***predecessor*** of N_k and N_k is a ***successor*** of N_1. For a path consisting of two nodes N, M we say N is a ***direct predecessor*** of M and M is a ***direct successor*** of N. A node without successors is a ***leaf***. A ***tree*** is a directed graph such that

- there is one node, called the ***root***, that has no predecessor
- each node other than the root has exactly one predecessor.

A ***concept tree*** is a tree such that every node is equipped with components, namely

- *type*
- *extended*
- *concept-description*
- *value*.

The values for the component *type* range over the symbols "$\sqcap$", "$\sqcup$" and "$\exists R$" where R is a role symbol, for the component *extended* they range over the symbols "yes" and "no", and for *value* they range over the symbols “solved”, “clash”, “cycle” and “null”. The values for the component *concept-description* are concept descriptions. Given a node N in a concept tree we will access the contents of the corresponding component with N.component. Figure 4.1 shows a concept tree.

A concept tree T is called ***extended*** if for every node N in T N.extended = “yes”.

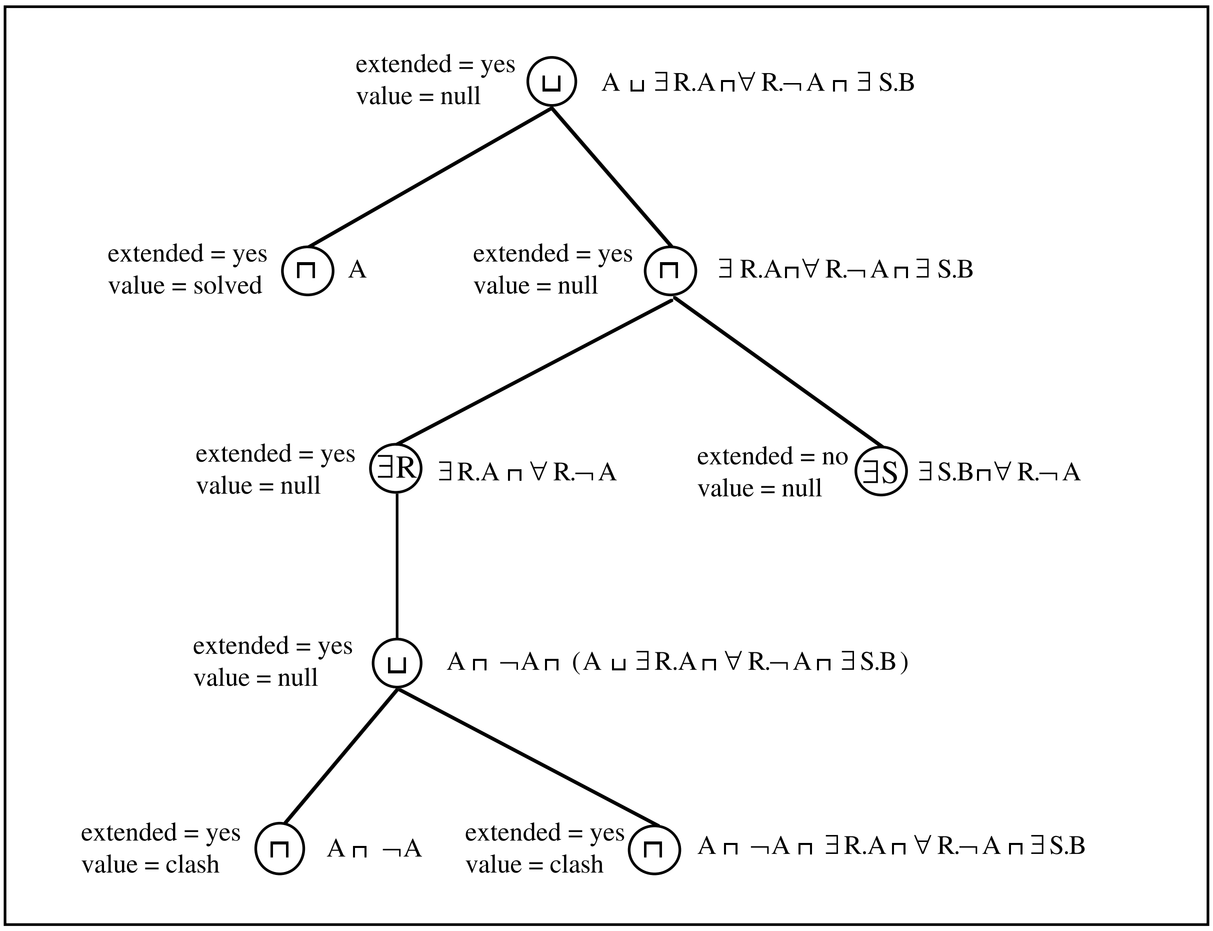

Figure 4.1: A concept tree.

4.2 An Algorithm for Deciding the Universal Satisfiability of Concept Descriptions

We will now present an algorithm that decides whether a given concept description is universally satisfiable. 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. We will see that the given concept description is universally satisfiable if and only if the extended concept tree satisfies a certain condition, which can be checked easily.

The algorithm uses several functions. The function Universal-Satisfiability takes a concept description as input, creates a concept tree, and returns this tree as argument to the function Extend-concept-tree. This function expands the concept tree by iterated calls of the functions Expand-or-node, Expand-and-node, and Expand-∃R-node until an expanded concept tree is obtained.

The function **Universal-Satisfiability** takes a concept description C as input and creates a concept tree T. This concept tree consists of the node root with

- ➤ root.type = "$\sqcup$"
- ➤ root.extended = "no"
- ➤ root.value = "null" and

root.concept-description contains a disjunctive normal form for C. 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 Extend-or-node, Extend-and-node, and Extend-∃R-node as subfunctions. Here is the formulation of the function Extend-concept-tree in a Pascal-like notation.

```
Algorithm Expand-concept-tree ($T$)
    if $T$ is extended
        then return $T$
    elsif $T$ contains a node $N$ such that $N$.type = "$\sqcup$" and $N$.extended = "no"
        then Expand-or-node($T$,$N$)
    elsif $T$ contains a node $N$ such that $N$.type = "$\sqcap$" and $N$.extended = "no"
        then Expand-and-node($T$,$N$)
    else let $N$ be a node in $T$ such that $N$.type = "$\exists R$" and $N$.extended = "no"
        Expand-∃R-node($T$,$N$)
end Expand-concept-tree.
```

The function **Expand-or-node** takes a concept tree T and a node N occurring in T as arguments and returns a concept tree T'. Suppose $C_1 \sqcup C_2 \sqcup \dots \sqcup C_n$ is a disjunctive normal form of N.concept-description. We modify T (and 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-description = C_i
- ➤ N_i.value = "null"

are successors of N.

The function **Expand-and-node** takes a concept tree T and a node N occurring in T as arguments and returns a concept tree T'. We modify T (and obtain T') such that N.extended = “yes” and N.value is

- “solved” if N.concept-description doesn't contain an exists-restriction
- “clash” if N.concept-description contains a clash
- “cycle” if there is a predecessor N' of N such that N'.type = “$\sqcap$” and N'.concept-description is equal to N.concept-description modulo associativity, commutativity, and idempotency
- “null” otherwise.

Furthermore, if N.value = “null”, we create successors for N in the following way. Suppose N.concept-description = $A_1 \sqcap \ldots \sqcap A_n \sqcap \exists R_1.C_1 \sqcap \ldots \sqcap \exists R_l.C_l \sqcap \forall S_1.D_1 \sqcap \ldots \sqcap \forall S_k.D_k$. Then for every i, $1 \leq i \leq l$, the (newly created) node N_i with

- N_i.type = “$\exists R_i$”
- N_i.extended = “no”
- N_i.concept-description = $A_1 \sqcap \ldots \sqcap A_n \sqcap \exists R_i.C_i \sqcap \forall S_1.D_1 \sqcap \ldots \sqcap \forall S_k.D_k$
- N_i.value = “null”

is a successor of N.

The function **Expand-∃R-node** takes a concept tree T and a node N occurring in T as arguments and returns a concept tree T'. Suppose N.concept-description = $A_1 \sqcap \ldots \sqcap A_n \sqcap \exists R.C \sqcap \forall S_1.D_1 \sqcap \ldots \sqcap \forall S_k.D_k$. We modify T (and obtain T') such that N.extended = “yes” and the (newly created) node N' with

- N'.type = “$\sqcup$”
- N'.extended = “no”
- N'.concept-description = root.concept-description $\sqcap\, C \sqcap \bigsqcap_{R = S_j,\, 1 \leq j \leq k} D_j$
- N'.value = “null”

is a successor of N.

Let C be a finite concept description. Obviously, C contains finitely many subterms. Hence $\{ \bigsqcap_{1 \leq i \leq n} C_i \mid C_i \text{ is a subterm of } C\}$ contains finitely many elements modulo associativity, commutativity, and idempotency.

Proposition 4.1: (Termination)

Let C be a concept description. Then Universal-Satisfiability(C) terminates.

Proof: Assume that the algorithm Universal-Satisfiability does not terminate. Then an infinite concept tree is generated since each call of Extend-or-node, Extend-and-node, or Extend-∃R-node adds new nodes to the concept tree. Since every node has finitely many successors we conclude with König’s Lemma that there exists an infinite path in this tree. This infinite path contains infinitely many nodes $N_1, N_2, \dots$ with N_i.type = "$\sqcap$" and N_i.concept-description is not equal to N_j.concept-description modulo idempotency and commutativity for $i < j$; or otherwise we would have then N_j.value = "cycle" and thus N_j would be a leaf. On the other hand, it is easy to show that any concept description of a node N with N.type = "$\sqcap$" is of the form $C_1 \sqcap \dots \sqcap C_n$ where the C_i are subterms of root.concept-description. Since root.concept-description is finite there exist only finitely many concept descriptions modulo associativity, commutativity, and idempotency that are conjunctions of subterms of root.concept-description as mentioned before. Thus, there cannot exist an infinite path which implies that the algorithm terminates. ❑

An ***instance*** is obtained from a concept tree by keeping for a node N with N.type = "$\sqcap$" or N.type = "$\exists R$" all direct successors, and by keeping for a node N with N.type = "$\sqcup$" only one of its direct successors. Figure 4.2 shows the instances for the concept tree given in Figure 4.1.

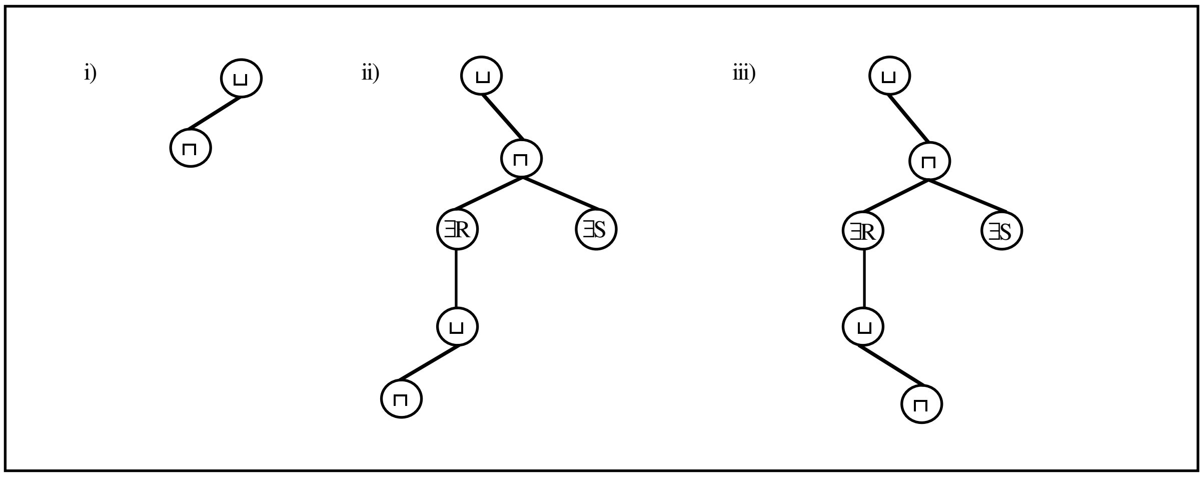

Figure 4.2: The three instances of the concept tree given in Figure 4.1.

An instance T is **successful** iff for every leaf N in T N.value = "solved" or N.value = "cycle".

Proposition 4.2: (Completeness)

Let C be a concept description and let T be the extended concept tree computed by Universal-Satisfiability*(C). If C is universally satisfiable, then there exists a successful instance of T.*

Proof: Let C be a concept description and let $\mathcal{I}$ be an interpretation such that $C^{\mathcal{I}} = \top^{\mathcal{I}}$. Furthermore, let T be the extended concept tree computed by Universal-Satisfiability*(C)*. Then root.concept-description = C. We use $\mathcal{I}$ to construct a successful instance of T. That means, for every node N with N.type = "$\sqcup$" we have to choose exactly one direct successor.

Suppose $a \in \top^{\mathcal{I}}$. The root is a node of type "$\sqcup$" and we have an element $a \in C^{\mathcal{I}}$. If $C_1 \sqcup \dots \sqcup C_n$ is a disjunctive normal form for C, then there exist direct successors $N_1, \dots, N_n$ of root such that N_i.type = "$\sqcap$" and N_i.concept-description = C_i for i, $1 \le i \le n$. Since $a \in (C_1 \sqcup$

$\ldots \sqcup C_n)^I$ there exists an i such that $a \in C_i^I$. We choose the node N_i as direct successor for the root in our instance. Since $a \in C_i^I$, there is no clash in C_i and hence N_i.value ≠ "clash". If N_i.value = "solved" or N_i.value = "cycle", then N_i doesn't have any successor and we are done. Otherwise, if N_i.value = "null", consider the concept description C_i, which has the form $A_1 \sqcap \ldots \sqcap A_m \sqcap \exists R_1.E_1 \sqcap \ldots \sqcap \exists R_l.E_l \sqcap \forall S_1.D_1 \sqcap \ldots \sqcap \forall S_k.D_k$. Since $a \in C_i^I$ we have $a \in \exists R_i.E_i^I$ for i, $1 \leq i \leq l$. There exists $b_i \in \top^I$ such that $(a, b_i) \in R_i^I$ and $b_i \in E_i^I$ $(1 \leq i \leq l.)$ Furthermore $b_i \in D_j$, if $R_i = S_j$ $(1 \leq j \leq k)$ since $a \in (\forall S_j.D_j)^I$. Since $b_i \in \top^I$, we also have $b_i \in C^I$. Hence $b_i \in (C \sqcap E_i \sqcap \bigsqcap_{R_i = S_j} D_j)^I$. By construction of the concept tree the node N_i has exactly l direct successors $M_1, \ldots, M_l$. Every M_i has exactly one direct successor M_i' with

- M_i'.concept-description = $C \sqcap E_i \sqcap \bigsqcap_{R_i = S_j} D_j$
- M_i'.type = "$\sqcup$".

and for each i we have an element $b_i \in (C \sqcap E_i \sqcap \bigsqcap_{R_i = S_j} D_j)^I$. We can now proceed with these nodes of type "$\sqcup$" as described above for root. Since the extended concept tree T computed by Universal-Satisfiability(C) is finite, this construction process terminates. Note that the constructed instance does not contain a node N with N.value = "clash" and hence we have a successful instance.

Thus we have shown that, given a universally satisfiable concept description C, the expanded concept tree contains a successful instance. ❑

Let S be a successful instance of the extended concept tree computed by Universal-Satisfiability(C). Then S yields a ***canonical interpretation*** I, which is defined as follows.

(1) The elements of the domain $\top^I$ are the nodes $N_1, \ldots, N_n$ in S such that N_i.type = "$\sqcap$" and N_i.value ≠ "cycle".

(2) Interpretation of role symbols: Let N be an element of $\top^I$. Then N.type = "$\sqcap$" and N.value = "solved" or N.value = "null". If N.value = "solved", then N is a leaf in S, and for any role R, N does not have an R-successor in I. If N.value = "null", then there exist direct successors $M_1, \ldots, M_n$ of N with M_i.type = "$\exists R_i$" for i, $1 \leq i \leq n$. Every M_i has exactly one direct successor M_i' with M_i'.type = "$\sqcup$", and every M_i' has exactly one direct successor M_i'' with M_i''.type = "$\sqcap$". If M_i''.value = "solved" or M_i''.value = "null", then we set $(N, M_i'') \in R_i^I$. Note that $M_i'' \in \top^I$. Otherwise M_i''.value = "cycle" and there exists a predecessor P of M_i'' such that P.type = "$\sqcap$", P.value ≠ "cycle", and P.concept-description is equal to M_i''.concept-description modulo associativity, commutativity, and idempotency. In this case we set $(N, P) \in R_i^I$. Note that $P \in \top^I$.

(3) Interpretation of concept symbols: Suppose $N \in \top^I$. Then N.concept-description is of the form $A_1 \sqcap \ldots \sqcap A_m \sqcap \exists R_1.E_1 \sqcap \ldots \sqcap \exists R_l.E_l \sqcap \forall S_1.D_1 \sqcap \ldots \sqcap \forall S_k.D_k$. If A_i $(1 \leq i \leq m)$ is a non-negated concept symbol, then we set $N \in A_i^I$. Note that N.value ≠ "clash", and hence there does not exist an A_j with $A_j = \neg A_i$. Thus $N \in (A_1 \sqcap \ldots \sqcap A_m)^I$.

Before we can show that this canonical interpretation satisfies $\top^I = C^I$, we need one more definition and an observation concerning concept descriptions of nodes with type "$\sqcap$".

The **depth** τ of a concept description in negation normal form is defined as follows.

- ➤ $\tau(C) = \tau(\neg C) = 0$ if C is a concept symbol
- ➤ $\tau(\forall R.C) = \tau(\exists R.C) = 1 + \tau(C)$
- ➤ $\tau(C \sqcap D) = \tau(C \sqcup D) = max\{\tau(C), \tau(D)\}$.

Let C be a concept description and let T be the extended concept tree computed by Universal-Satisfiability(C). If N is a node in T with N.type = "$\sqcup$", then N.concept-description has the form $C \sqcap C_{rest}$. Let $C_1 \sqcup ... \sqcup C_n$ be a disjunctive normal form of $C \sqcap C_{rest}$. There exist direct successors N_1, ..., N_n of N with N_i.type = "$\sqcap$" and N_i.concept-description = C_i. Obviously, C subsumes $C \sqcap C_{rest}$, and $C \sqcap C_{rest}$ subsumes each C_i. Thus, if N_i is a node in T with N_i.type = "$\sqcap$", then root.concept-description (which is C) subsumes N_i.concept-description.

Proposition 4.3: (Soundness)
Let C be a concept description and let T be the extended concept tree computed by Universal-Satisfiability(C). If there exists a successful instance of T, then C is universally satisfiable.

Proof: Let C be a concept description, let T be the extended concept tree computed by Universal-Satisfiability(C), and let S be a successful instance of T. Note that root.concept-description = C. We will show that the canonical interpretation I induced by S interprets C as $\top^I$. First we prove the following claim.

Claim: Let I be the canonical interpretation induced by S and let $N \in \top^I$. If D is a subconjunction of N.concept-description, then $N \in D^I$.
Proof: (by **induction on the depth** $\tau(D)$.) Let $N \in \top^I$. Then N is a node in S with N.type = "$\sqcap$" and N.value ≠ "cycle". We know that N.concept-description is of the form $A_1 \sqcap ... \sqcap A_m \sqcap \exists R_1.E_1 \sqcap ... \sqcap \exists R_l.E_l \sqcap \forall S_1.D_1 \sqcap ... \sqcap \forall S_k.D_k$. Now let D be a subconjunction of N.concept-description.
$\tau(D) = 0$**.** Then D is of the form $A_{i_1} \sqcap ... \sqcap A_{i_n}$. By definition of the interpretation of concept symbols we have $N \in (A_1 \sqcap ... \sqcap A_m)^I$, which implies $N \in (A_{i_1} \sqcap ... \sqcap A_{i_n})^I$.
$\tau(D) > 0$**.** Let $D = D_1 \sqcap ... \sqcap D_n$. We have to show for all i, $1 \leq i \leq n$, that $N \in D_i{}^I$. If $\tau(D_i) < \tau(D)$ we know by the induction hypothesis $N \in D_i{}^I$. Now suppose $\tau(D_i) = \tau(D)$. Then D_i is of the form $\forall R.E$ or $\exists R.E$, where $\tau(E) = \tau(D)$ - 1.
(1) Suppose $D_i = \forall R.E$. We have to show that for any $M \in \top^I$, $(N, M) \in R^I$ implies $M \in E^I$. Consider the node N. If $(N, M) \in R^I$, then there exists a direct successor N' of N with N'.type = "$\exists R$". The node N' has a direct successor N'' with N''.type = "$\sqcup$". Furthermore, N''.concept-description = $C \sqcap E \sqcap \bigsqcap_{R = S_j} D_j$. Suppose $E_1 \sqcup ... \sqcup E_n$ is a disjunctive normal form of E. It is easy to see that, if $C_1 \sqcup ... \sqcup C_m$ is a disjunctive normal form of N''.concept-description, then every C_i contains an E_j as subconjunction for some j, $1 \leq j \leq n$. Since N''.type = "$\sqcup$", N'' has exactly one direct successor P with P.type = "$\sqcap$" and P.concept-description = C_i for some i, $1 \leq i \leq m$. If P.value ≠ "cycle", P is equal to M. Otherwise P.value = "cycle", and M is a

predecessor of N" with M.type = "$\sqcap$" and M.concept-description is equal to C_i modulo idempotency and commutativity. As mentioned before E_j is a subconjunction of C_i. Since $\tau(E_j) \leq \tau(E) < \tau(\forall R.E)$ we conclude by induction that $M \in E_j^I$ and hence $M \in E^I$. Thus we have shown that $N \in (\forall R.E)^I$.

(2) Suppose $D_i = \exists R.E$. We have to show that there exists an $M \in \top^I$ with $(N, M) \in R^I$ and $M \in E^I$. Consider the node N. Since $\exists R.E$ is a subconjunction of N.concept-description there exists a direct successor N' of N with N'.type = "$\exists R$". Furthermore, N' has a direct successor N" with N".type = "$\sqcup$" and N".concept-description = $C \sqcap E \sqcap \bigsqcap_{R = S_j} D_j$. Suppose $E_1 \sqcup \ldots \sqcup E_n$ is a disjunctive normal form of E. As shown in (1) we know that, for every direct successor M of N", there exists j, $1 \leq j \leq n$, such that E_j is a subconjunction of M.concept-description. Note that M.type = "$\sqcap$". If M.value ≠ "cycle", then $M \in \top^I$ and by definition of I we have $(N, M) \in R^I$. Since $\tau(E_j) < \tau(\forall R.E)$ we conclude by induction that $M \in E_j^I$ and hence $M \in E^I$. Otherwise, if M.value = "cycle", then there exists a predecessor M' of M with M'.type = "$\sqcap$" and M'.concept-description is equal to M.concept-description modulo idempotency and commutativity. By definition of I we know that $M' \in \top^I$ and $(N, M') \in R^I$. As before we conclude by induction that $M' \in E_j^I$ and hence $M' \in E^I$. Thus we have shown that $N \in (\exists R.E)^I$. ❑

Let I be the canonical interpretation induced by S and let $N \in \top^I$. We have shown that, if D is a subconjunction of N.concept-description, then $N \in D^I$. Every concept description is a subconjunction of itself and hence $N \in (N\text{.concept-description})^I$. Since N.type = "$\sqcap$", we know that C subsumes N.concept-description (as mentioned above). Hence for every $N \in \top^I$ we have $N \in C^I$. Thus $\top^I = C^I$. ❑

Let C be a concept description. We have shown that the call Universal-Satisfiability(C) terminates and returns an expanded concept tree T(Proposition 4.1). If C is universal satisfiable, then T contains a successful instance (Proposition 4.2), and if T contains a successful instance, then C is universally satisfiable (Proposition 4.3). Thus we have proven the main result of this paper: (?)

Theorem 4.4:

A concept description C is universally satisfiable if and only if the extended concept tree computed by Universal-Satisfiability(C) *contains a successful instance.*

Note that it can be easily decided whether a concept tree does contain a successful instance by using a depth-first search. For further remarks about the implementation of the algorithm see subsection 4.4.

4.3 Deciding the Satisfiability of Concept Descriptions

As mentioned before (Schmidt-Schauß & Smolka, 1988) have already described an algorithm for deciding the satisfiability of concept descriptions. However, such an algorithm can also be obtained within our formalism developed for deciding universal satisfiability. Since this requires only a slight modification of the function Universal-Satisfiability, we will include a description of this algorithm. The idea underlying the modification is as follows. Suppose we want to decide whether a concept description C_0 is universally satisfiable. The function Universal-Satisfiability tries to construct an interpretation I such that $\top^I = C_0^I$ and $\top^I \neq \emptyset$. To do this the function generates an element which is in $\top^I$. In general, constraints imposed on this element force the function to generate further elements which are in $\top^I$. Since we want to construct an interpretation with $\top^I = C_0^I$, the newly introduced elements also have to be in C_0^I.

A satisfiability algorithm has to do less work. Since we only want to find an interpretation I such that C_0^I is nonempty, it is sufficient to guarantee that there exists one element which is in C_0^I. Thus, there is no need to force the newly introduced elements to be in C_0^I.

This observation leads us to the following straightforward modification of the function Extend-∃R-node used in Universal-Satisfiability. Let N be a node with N.type = "$\exists R$" and N.concept-description = $A_1 \sqcap \ldots \sqcap A_n \sqcap \exists R.C \sqcap \forall S_1.D_1 \sqcap \ldots \sqcap \forall S_k.D_k$. The function Extend-∃R-node creates a new node N' such that N'.concept-description = root.concept-description $\sqcap C \sqcap \bigsqcap_{R = S_j, 1 \leq j \leq k} D_j$. Recall that the initialization step of the algorithm was done in a way such that root.concept-description = C_0. Thus we just have to omit root.concept-description in the definition of N'.concept-description, i.e., we modify Extend-∃R-node such that N'.concept-description = $C \sqcap \bigsqcap_{R = S_j, 1 \leq j \leq k} D_j$. We obtain the function **Satisfiability** from Universal-Satisfiability by taking this modified version of Extend-∃R-node.

Let C_0 be a concept description and let T be an extended concept tree computed by Satisfiability(C_0). Furthermore, let N, M be nodes in T with N.type = M.type = "$\sqcap$". It is easy to see that $\tau(N$.concept-description$) > \tau(M$.concept-description$)$ if N is a predecessor of M. As a consequence, N.value ≠ "cycle" for every node N in T. Furthermore, if $\tau(C_0) = n$, the longest path in S contains at most n nodes of type "$\sqcap$", and hence at most $3 * n$ nodes. Since obviously every node in S has finitely many direct successors we have the following result.

Proposition 4.5: (Termination)
Let C_0 be a concept description. Then Satisfiability(C_0) terminates without creating a node with value "cycle".

In the following we will prove that a concept description C_0 is satisfiable if and only if there exists a successful instance in the extended concept tree computed by Satisfiability(C_0). As in Chapter 4.2 one can show the completeness (Proposition 4.6) and the soundness (Proposition 4.7) of the algorithm.

Proposition 4.6: (Completeness)
Let C_0 be a concept description and let T be the extended concept tree computed by Satisfiability(C_0). If C_0 is satisfiable, then there exists a successful instance of T.

Proof: Easy modification of the proof of Proposition 4.2. ❑

Proposition 4.7: (Soundness)

Let C_0 be a concept description and let T be the extended concept tree computed by Satisfiability(C_0). *If there exists a successful instance of T, then C_0 is satisfiable.*

Proof: Let C_0 be a concept description and let S be a successful instance of the extended concept tree computed by Satisfiability(C_0). Consider the canonical interpretation I induced by S and $N \in \top^I$. We will show that C_0^I is a nonempty set. To that purpose we will use the following claim:

Claim: Let I be the canonical interpretation induced by S and let $N \in \top^I$. Then $N \in (N.\text{concept-description})^I$. As mentioned above, if N and M are elements of $\top^I$, then $\tau(N.\text{concept-description}) > \tau(M.\text{concept-description})$ if N is a predecessor of M. Thus one can easily prove the claim by **induction on the depth** $\tau(N.\text{concept-description})$. The proof of this claim is similar to the proof of the claim in Proposition 4.3. However it is easier for the following reason. In the proof of Proposition 4.3 we had to formulate a stronger induction hypothesis, since if N is predecessor of M, then $\tau(N.\text{concept-description})$ is not necessarily greater than $\tau(M.\text{concept-description})$.

Let $C_1 \sqcup \ldots \sqcup C_n$ be a disjunctive normal form of C_0. Then there exists a node in S such that N.type = "$\sqcap$" and N.value ≠ "cycle". Hence $N \in \top^I$. We conclude that $N \in C_i^I$ and $N \in C_0^I$. Thus we have shown that C_0 is satisfiable. ❑

5 Conclusion

In the previous two sections we described algorithms for deciding satisfiability and universal satisfiability. These algorithms were presented for theoretical purposes, i.e. for proving soundness and completeness, rather than for an actual implementation. The algorithms as presented above suffer from two main sources of unnecessary complexity: The first point is that we construct a complete concept tree before testing for the existence of a successful instance. As soon as a successful instance is found the remaining unsearched part of the tree was constructed in vain. Thus an actual implementation should combine the generation of the tree with searching for a successful instance. This can be realized by a simple depth first strategy with backtracking to a previous "or"-node if a clash is encountered. Consequently, only one path has to be stored instead of keeping the whole tree in memory. As a second point, the size of a disjunctive normal form $C_1 \sqcup \ldots \sqcup C_n$ for a concept description C may be exponential in the size of C. In addition, if C_i leads to a successful instance, the computation of $C_{i+1}, \ldots, C_n$ was superfluous. However, by choosing an appropriate data structure one can enumerate the disjuncts C_i one after the other using polynomial space.

A constrained resolution prover (CORE) has been implemented along these lines in the WINO-project of the German national institute for artificial intelligence (DFKI) by our research team. CORE will be the heart of a first prototype of a knowledge representation and reasoning system (KRIS) to be developed in the WINO-project. Thereby, our future research is concentrated on following problems:

- ➤ As mentioned in section 2, the input formulae have to be in clause form. Since this is a strong restriction, we examine how to transform arbitrary constrained formulae into constrained clauses in the case of our application with concept constraints.
- ➤ In this paper we considered the concept language $\mathcal{ALC}$ as constraint theory. However, other language constructs such as number restrictions, intersection of roles etc., are used to describe taxonomical knowledge. Hence we will use such an enriched concept language as constraint theory. There are already some results how to devise algorithms for checking satisfiability and universal satisfiability in these languages.
- ➤ Besides concept languages, so-called assertional languages are employed in KL-ONE systems to represent knowledge about individuals. Thus, we are going to amalgamate our constraint theory by allowing such assertions.

Acknowledgements. We would like to thank Erich Achilles, Armin Laux, Jörg Peter Mohren, and Thomas Steinbach for their implementational work.
This research was supported by the German Bundesministerium für Forschung und Technologie under grant ITW 8903 0 and by the Esprit Basic Research Project 3012, COMPULOG.

References

Baader, F.: "*Regular Extensions of KL-ONE*", DFKI Research Report, forthcoming.

Beierle, C., Hedtstück, U.,Pletat, U., Schmitt, P. H., Siekmann, J.: "*An Order-Sorted Logic for Knowledge Representation Systems*", IWBS Report 113, IBM Deutschland, 1990.

Brachman, R. J., Schmolze, J. G.: "An Overview of the KL-ONE Knowledge Representation System", *Cognitive Science*, 9(2), pp. 171-216, 1985.

Brachman, R.J., Levesque, H.J.: *Readings in Knowledge Representation.* Morgan Kaufmann Publishers, 1985.

Brachman, R.J., Pigman Gilbert, V., Levesque, H.J.: "An Essential Hybrid Reasoning System: Knowledge and Symbol Level Account in KRYPTON", in *Proceedings of the 9th IJCAI*, pp. 532-539, Los Angeles, Cal., 1985.

Bürckert, H.-J.: "*A Resolution Principle for a Logic with Restricted Quantifiers*", Dissertation, Universität Kaiserslautern, Postfach 3049, D-6750 Kaiserslautern, West-Germany, 1990.

Bürckert, H.-J.: “A Resolution Principle for Clauses with Constraints”, in *Proceedings of 10th International Conference on Automated Deduction*, Springer LNAI 449, pp. 178-192, 1990.

Cohn, A. G.: “ A More Expressive Formulation of Many-Sorted Logic”, *JAR* 3,2, pp. 113-200, 1987.

Frisch, A.: “A General Framework for Sorted Deduction: Fundamental Results on Hybrid Reasoning”, in *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning*, pp. 126-136, 1989.

Frisch, A.: “An Investigation into Inference with Restricted Quantification and Taxonomic Representation”, *Logic Programming Newsletters*, 6, pp. 5-8, 1986.

Höhfeld, M., Smolka, G.: “*Definite Relations over Constraint Languages*”, LILOG-Report 53, IBM Deutschland, 1988.

Hollunder, B.: “Hybrid Inferences in KL-ONE-based Knowledge Representation Systems”, DFKI Research Report RR-90-06, DFKI, Postfach 2080, D-6750 Kaiserslautern, West-Germany, 1990. Also in the *Proceedings of the 14th German Workshop on Artificial Intelligence*, Springer-Verlag, 1990.

Hollunder, B., Nutt, W.: “Subsumption Algorithms for Concept Languages”, DFKI Research Report RR-90-04, DFKI, Postfach 2080, D-6750 Kaiserslautern, West-Germany, 1990. Also in the *Proceedings of the 9th European Conference on Artificial Intelligence*, Pitman Publishing, 1990.

Huet, G.: “*Constrained Resolution - A Complete Method for Higher Order Logic*”, Ph.D. Thesis, Case Western University, 1972.

Jaffar, J., Lassez, J.-L.: “Constrained Logic Programming”, *Proceedings of ACM Symp. on Principles of Programming Languages*, 111-119, 1987.

Levesque, H. J., Brachman, R. J.: “Expressiveness and Tractability in Knowledge Representation and Reasoning”, *Computational Intelligence*, 3, pp. 78-93, 1987.

MacGregor, R., Bates, R.: “*The Loom Knowledge Representation Language*”, Technical Report ISI/RS-87-188, University of Southern California, Information Science Institute, Marina del Rey, Cal., 1987.

Nebel, B.: “*Reasoning and Revision in Hybrid Representation Systems”*, Lecture Notes in Artificial Intelligence, LNAI 422, Springer-Verlag, 1990.

Oberschelp, A.: “Untersuchungen zur mehrsortigen Quantorenlogik”, *Mathematische Annalen*, 145:297-333,1962.

Quillian, R. M.: “Semantic Memory”, in *Semantic Information Processing* (ed. M. Minsky), pp. 216-270, MIT-Press, 1968.

Robinson, J.A.: “A Machine Oriented Logic Based on the Resolution Principle”, *J. ACM*, 12(1), pp. 23-41, 1965.

Schmidt, A.: “Ueber deduktive Theorien mit mehreren Sorten von Grunddingen”, *Mathematische Annalen*, 115:485-506,1938.

Schmidt, A.: “Die Zulässigkeit der Behandlung mehrsortiger Theorien mittels der üblichen einsortigen Prädikatenlogik”, *Mathematische Annalen*, 123:187-200,1951.

Schmidt-Schauß, M., Smolka, G.: "*Attributive Concept Descriptions with Unions and Complements*", SEKI Report SR-88-21, FB Informatik, University of Kaiserslautern, West Germany, 1988. To appear in Artificial Intelligence.

Schmidt-Schauß, M.: "*Computational Aspects of an Order-sorted Logic with Term Declarations"*, Lecture Notes on Artificial Intelligence, LNAI 395, Springer, 1989

Smolka, G.: "*Logic Programming over Polymorphically Order-sorted Types*", Dissertation, Universität Kaiserslautern, 1989.

Stickel, M. E.: "Automated Deduction by Theory Resolution", *Journal of Automated Reasoning*, 1:333-355, 1985.

Vilain., M. B.: "The Restricted Language Architecture of a Hybrid Representation System", in *Proceeding of the 9th IJCAI*, pp. 547-551, Los Angeles, Cal., 1985.

Walther, C.: "A Many-sorted Calculus Based on Resolution and Paramodulation", *Research Notes in Artificial Intelligence,* Pitman, Morgan Kaufman Publishers, 1987.

Walther, C.: "Many-Sorted Unification", *J. ACM*, 35(1), pp. 1-17, 1988.

Weidenbach, C., Ohlbach H.-J.: "A Resolution Calculus with Dynamic Sort Structures and Partial Functions", *Proceedings of the 9th European Conference on Artificial Intelligence*, Pitman Publishing, 1990.