Concept Logics

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

September 1990
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
Concept Logics

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

DFKI-RR-90-10
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 ........................................................................................................ 3
2. Constrained Resolution ......................................................................................... 4
3. Concept Languages ............................................................................................ 8
4. Proving Universal Satisfiability and Satisfiability of Concept Descriptions. 12
   4.1 Basic Definitions ............................................................................................ 13
   4.2 An Algorithm for Deciding the Universal Satisfiability of Concept Descriptions ....................................................................................................... 15
   4.3 Deciding the Satisfiability of Concept Descriptions ........................................ 21
5. Conclusion ............................................................................................................ 23
References ................................................................................................................. 24
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:

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

or in a sorted logic as

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

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 $B \lor C$, from the parent formulae $A \lor B$ and $\neg A \lor C$. 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:
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 $s_i = t_i$ ($1 \leq i \leq n$) to the resolvent (instead of instantiating it with the substitution)\(^3\).

A slight modification of this approach transforms 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

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

is replaced by

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

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 $\Gamma$ and $\Gamma'$ denote constraints that are separated from their corresponding clause by $\ || $, then we denote constrained resolution as:

$$P(x_1, \ldots, x_n) \lor C \ || \ \Gamma$$

$$-P(y_1, \ldots, y_n) \lor C' \ || \ \Gamma'$$

$$C \lor C' \ || \ \Gamma \land \Gamma'\{y_i = x_i \ (1 \leq i \leq n)\} \text{ is satisfiable}$$

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 \( \Gamma \) comes with a finite set of variables \( \text{Var}(\Gamma) = \{x_1, \ldots, x_n\} \). 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 \( \Gamma \) it contains a set \( \Gamma^\mathcal{A} \) of \( \mathcal{A} \)-solutions, i.e., of assignments \( \sigma: \mathcal{V} \to D^\mathcal{A} \). 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 \( \Gamma_1 \) and \( \Gamma_2 \) there exists their conjunct constraint \( \Gamma_1 \land \Gamma_2 \) such that \( (\Gamma_1 \land \Gamma_2)^\mathcal{A} = \Gamma_1^\mathcal{A} \cap \Gamma_2^\mathcal{A} \) for every constraint model \( \mathcal{A} \); and we assume that for every constraint \( \Gamma \) and for every (renaming or identifying) variable substitution \( \varphi: \mathcal{V} \to \mathcal{V} \) there exists the (variant or instance) constraint \( \varphi\Gamma \) such that \( (\varphi\Gamma)^\mathcal{A} = \{\sigma^* | \sigma \in \Gamma^\mathcal{A} \text{ with } \sigma_x = \sigma_y \text{ if } \varphi x = \varphi y \} \) for every constraint model \( \mathcal{A} \), where \( \sigma^* \) 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 $\Gamma$. Constrained clauses are written $C \parallel \Gamma$.

The semantics of constrained clauses are defined as follows.

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

$$\begin{align*}
(A, \alpha) \not\models P(x_1, \ldots, x_n) & \iff (\alpha x_1, \ldots, \alpha x_n) \not\in P^A \\
(A, \alpha) \not\models \neg P(x_1, \ldots, x_n) & \iff (\alpha x_1, \ldots, \alpha x_n) \not\in P^A \\
(A, \alpha) \not\models C & \iff (A, \alpha) \not\models L \text{ for some literal } L \text{ in the set } C
\end{align*}$$

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

$$A \models C \parallel \Gamma \iff (A, \alpha) \not\models C \text{ for each } A\text{-solution } \alpha \text{ of } \Gamma.$$ 

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

We call a set of constrained clauses $R$-satisfiable iff there is an $R$-structure $A$ such that each of the clauses is satisfied by the structure; otherwise the clause set is called $R$-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 $\Gamma$ is called satisfiable or solvable iff there is at least one model $A$, where the $A$-solution set of the constraint is not empty. We have the following constrained resolution rule for a pair of variable disjoint constrained clauses:

$$\begin{align*}
\{P(x_{11}, \ldots, x_{1n}), \ldots, P(x_{k1}, \ldots, x_{kn})\} \cup C \parallel \Gamma \\
\{-P(y_{11}, \ldots, y_{1n}), \ldots, -P(y_{m1}, \ldots, y_{mn})\} \cup C' \parallel \Gamma' \\
\hline
C \cup C' \parallel \varphi(\Gamma \land \Gamma')
\end{align*}$$

if $\varphi(\Gamma \land \Gamma')$ is satisfiable

where $C$ and $C'$ are the remaining parts of the two clauses and the new constraint $\varphi(\Gamma \land \Gamma')$ is a conjunction instantiated by a variable substitution $\varphi$ that identifies corresponding variables of the complementary literals: $\varphi x_{1i} = \ldots = \varphi x_{ki} = \varphi y_{1i} = \ldots = \varphi y_{mi} \ (1 \leq i \leq n)$. 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 \( C \) of constrained clauses is \( \mathcal{R} \)-unsatisfiable iff there exists a (finite) refutation of \( C \).*

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 \((\text{person} \cap \neg \text{female})\), where \(\cap\) 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 \(\exists \text{child}.\text{female}\) and \(\forall \text{child}.\text{female}\), 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, \((\geq 3 \text{ friend}) \cap (\leq 2 \text{ child})\) can be read as “all individuals having at least three friends and at most two children.”

Concept languages, e.g., \(FL\) and \(FL^-\) (Levesque & Brachman, 1987), \(T\overline{F}\) and \(N\overline{TF}\) (Nebel, 1990), or the \(AL\)-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 \(ALC\) (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 \(ALC\), e.g. \(ALC\) 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 \(T\) (top symbol) and \(\bot\) (bottom symbol). The set of concept descriptions of \(ALC\) is inductively defined:

\[
\begin{align*}
\text{every concept symbol is a concept description (atomic description).}
\end{align*}
\]

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

\[
\begin{align*}
\text{\(C \cap D\) (conjunction), \(C \cup D\) (disjunction) and \(\neg C\) (negation) are concept descriptions, and}
\end{align*}
\]

\(^5\text{In this paper we do not consider the definition of complex roles. See (Baader, 1990; Hollunder & Nutt, 1990) for a discussion of role-forming constructs such as the transitive closure or intersection of roles.}\)
∀R.C (value restriction) and ∃R.C (exists restriction) are concept descriptions.

An interpretation \( I \) of a concept description consists of a nonempty set \( T^I \) (the domain of \( I \)), and a function \( \cdot^I \) (the interpretation function of \( I \)). The interpretation function maps every concept symbol \( A \) to a subset \( A^I \) of \( T^I \) and every role symbol \( R \) to a subset \( R^I \) of \( T^I \times T^I \). 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

- \( (C \cap D)^I := C^I \cap D^I \)
- \( (C \cup D)^I := C^I \cup D^I \)
- \( (\neg C)^I := T^I \setminus C^I \)
- \( (\forall R.C)^I := \{ a \in T^I \mid \forall b: (a, b) \in R^I \Rightarrow b \in C^A \} \)
- \( (\exists R.C)^I := \{ a \in T^I \mid \exists b: (a, b) \in R^I \land b \in C^A \} \).

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 \( \text{person} \cap \neg\text{female} \). If we want to use this expression in other concept descriptions it is appropriate to define the terminological axiom

\[ \text{man} = \text{person} \cap \neg\text{female} \]

where \( \text{man} \) is a concept. If \( \text{child} \) is a role, we can describe “not female persons with only female children” by the expression \( \text{man} \cap \forall \text{child}\cdot\text{female} \). 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 \( C^I = T^I \). 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 so-called 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_i:C_i, \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 $I$ is an $I$-assignment $\alpha: \mathcal{V} \to T_I$ such that $\alpha x_i \in C_i (1 \leq i \leq n)$. A concept constraint $\Gamma$ is satisfiable iff there exists an interpretation $I$ and a solution of $\Gamma$ in $I$.

Let $\Gamma$ be a concept constraint. Without loss of generality we can assume that the variables occurring in the atomic concept constraints of $\Gamma$ 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 $\Gamma^\prime := \Gamma \setminus \{x_i:C_i, x_j:C_j\} \cup \{x_i:C_i \cap C_j\}$. Obviously, $\Gamma$ is satisfiable if and only if $\Gamma^\prime$ 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 $I$ such that every $C_i$ 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 $I_i$ such that $C_i$ is nonempty. Without loss of generality we can assume that $T_i \cap T_j = \emptyset$ for $i \neq j$. Now, let $T^I := \bigcup_{i \leq n} T_i$, $A^I := \bigcup_{i \leq n} A_i$ for every concept symbol $A$, and let $R^I := \bigcup_{i \leq n} R_i$ for every role symbol $R$. Then $I$ is an interpretation and it is easy to see that $\Gamma$ has a solution in $I$. Hence $\Gamma$ is satisfiable. $\square$

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 \( I \), some \( C_i \) has a solution in \( I \).

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

\( (\Leftarrow) \) 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 \( I \) such that \( C_i^I = \emptyset \) for each \( i \) \( (1 \leq i \leq n) \). We show that \( \{ \Gamma_1, \ldots, \Gamma_n \} \) cannot be valid. Assume by contradiction that some \( \Gamma_{i_0} \) has a solution in \( I \). Then by Proposition 3.1 all its concept descriptions are satisfiable in \( I \), and hence especially \( (C_{i_0})^I \neq \emptyset \), which yields a contradiction. \( \Box \)

Now, let \( C_1, \ldots, C_n \) be concept descriptions and let \( I \) be any interpretation. Then obviously \( (\neg C_1 \lor \ldots \lor \neg C_n)^I = T^I \) iff \( C_i^I = \emptyset \) 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 \( \neg C_1 \lor \ldots \lor \neg C_n \) 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 \( \neg C_1 \lor \ldots \lor \neg C_n \) 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 \( \Gamma_i \). 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 $C = \exists R.A \land \forall R.B$ is valid, we want to construct a finite interpretation $I$ such that $C^I$ is a nonempty set, i.e., there exists an element in $T^I$ which is an element of $C^I$. The algorithm generates such an element $a$ and imposes the constraint $a \in C^I$ on it. Obviously, this yields that $a \in (\exists R.A)^I$ and $a \in (\forall R.B)^I$. From $a \in (\exists R.A)^I$ we can deduce that there has to exist an element $b$ such that $(a, b) \in R^I$ and $b \in A^I$. Thus the algorithm introduces a new element $b$ and constrains it as mentioned. Since $a \in (\forall R.B)^I$ and $(a, b) \in R^I$, we also get the constraint $b \in B^I$. We have now satisfied all the constraints in the example without getting a contradiction. This shows that $C^I$ is satisfiable. We have generated an interpretation $I$ which interprets $C$ as a nonempty set: $T^I = \{a, b\}$, $A^I = B^I = \{b\}$, and $R^I = \{(a, b)\}$. 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 $a \in (C \land D)^I$ forces to generate two new constraints $a \in C^I$ and $a \in D^I$. On the other hand, if we have a constraint $a \in (C \lor D)^I$, then we have to choose either $a \in C^I$ or $a \in D^I$.

Now let us consider an algorithm which checks the universal satisfiability of concept descriptions. Suppose $C = \exists R.A \land \forall R.B$ is universally satisfiable. We try to construct an interpretation $I$ such that $C^I = T^I$. Since $T^I$ is a nonempty set, there has to exist an element in $T^I$, and hence in $C^I$. Thus the algorithm generates such an element $a$ and imposes the constraint $a \in C^I$ on it. Furthermore, as argued above, the algorithm creates the constraints $(a, b) \in R^I$, $b \in A^I$, $b \in B^I$ where $b$ is a newly introduced element. Since $b$ is an element of $T^I$, we have to impose the additional constraint $b \in C^I$ on $b$. Again, because $b \in C^I$, there exists an element $c \in T^I$ which satisfies the constraints $(b, c) \in R^I$, $c \in A^I$, $c \in B^I$, $c \in C^I$. Obviously, the algorithm that tries to construct an interpretation $I$ with $C^I = T^I$ would run forever creating more and more new elements. However, if we set $c = b$, then we have $T^I = \{a, b\}$, $A^I = B^I = \{b\}$, $R^I = \{(a, b), (b, b)\}$, and hence $C^I = T^I$. 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 $i$, $1 \leq i < k$. 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”,

---

14
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 \( N\text{.component} \). Figure 4.1 shows a concept tree.

A concept tree \( T \) is called extended if for every node \( N \) in \( T \) \( N\text{.extended} = \text{“yes”} \).

---

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

The function \textbf{Universal-Satisfiability} takes a concept description \( C \) as input and creates a concept tree \( T \). This concept tree consists of the node root with

\begin{itemize}
  \item [\textit{root.type}] = “Ω”
  \item [\textit{root.extended}] = “no”
  \item [\textit{root.value}] = “null”
\end{itemize}

and \( \textit{root.concept-description} \) contains a disjunctive normal form for \( C \). Then the function \textit{Extend-concept-tree} is called with \( T \) as argument.

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

\begin{verbatim}
Algorithm Expand-concept-tree (T)
if T is extended
  then return T
elsif T contains a node N such that N.type = “Ω” and N.extended = “no”
  then Expand-or-node(T,N)
elsif T contains a node N such that N.type = “®” and N.extended = “no”
  then Expand-and-node(T,N)
else let N be a node in T such that N.type = “∃R” and N.extended = “no”
  Expand-∃R-node(T,N)
end Expand-concept-tree.
\end{verbatim}

The function \textbf{Expand-or-node} takes a concept tree \( T \) and a node \( N \) occurring in \( T \) as arguments and returns a concept tree \( T' \). Suppose \( C_1 \cup C_2 \cup \ldots \cup C_n \) is a disjunctive normal form of \( \textit{N.concept-description} \). We modify \( T \) (and obtain \( T' \)) such that \( N\textit{.extended} = \text{“yes”} \) and the (newly created) nodes \( N_i, 1 \leq i \leq n \), with

\begin{itemize}
  \item [\textit{N_i.type}] = “®”
  \item [\textit{N_i.extended}] = “no”
  \item [\textit{N_i.concept-description}] = \( C_i \)
  \item [\textit{N_i.value}] = “null”
\end{itemize}

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 \( N.\text{extended} = \text{"yes"} \) and \( N.\text{value} \) is

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

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

- \( N_i.\text{type} = \text{"∃R_i"} \)
- \( N_i.\text{extended} = \text{"no"} \)
- \( N_i.\text{concept-description} = A_1 \sqcap ... \sqcap A_n \sqcap \exists R_i.C_i \sqcap \forall S_1.D_1 \sqcap ... \sqcap \forall S_k.D_k \)
- \( N_i.\text{value} = \text{"null"} \)

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 \( N.\text{concept-description} = A_1 \sqcap ... \sqcap A_n \sqcap \exists R.C \sqcap \forall S_1.D_1 \sqcap ... \sqcap \forall S_k.D_k \). We modify \( T \) (and obtain \( T' \)) such that \( N.\text{extended} = \text{"yes"} \) and the (newly created) node \( N' \) with

- \( N'.\text{type} = \text{"∪"} \)
- \( N'.\text{extended} = \text{"no"} \)
- \( N'.\text{concept-description} = \text{root.\text{concept-description}} \sqcap R \sqcap \sqcap \bigcup_{1 \leq j \leq k} \exists R_i.C_i \bigcup \forall S_j.D_j \)
- \( N'.\text{value} = \text{"null"} \)

is a successor of \( N \).

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

**Proposition 4.1:** (Termination)

*Let \( C \) be a concept description. Then \text{Universal-Satisfiability}(C) terminates.*

**Proof:** Assume that the algorithm \text{Universal-Satisfiability} does not terminate. Then an infinite concept tree is generated since each call of **Extend-or-node**, **Expand-and-node**, or **Expand-∃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 \( N_1, N_2, \ldots \) with \( N_i.\text{type} = "\Omega" \) and \( N_i.\text{concept-description} \) is not equal to \( N_j.\text{concept-description} \) modulo idempotency and commutativity for \( i < j \); or otherwise we would have then \( N_j.\text{value} = "\text{cycle}" \) 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 \( N.\text{type} = "\Omega" \) is of the form \( C_1 \lor \ldots \lor C_n \) where the \( C_i \) are subterms of \( \text{root.concept-description} \). 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 \( N.\text{type} = "\forall" \) or \( N.\text{type} = "\exists R" \) all direct successors, and by keeping for a node \( N \) with \( N.\text{type} = "\Omega" \) 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.](image-url)

An instance \( T \) is successful iff for every leaf \( N \) in \( T \) \( N.\text{value} = "\text{solved}" \) or \( N.\text{value} = "\text{cycle}" \).

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

Suppose \( a \in T^I \). The root is a node of type "\lor" and we have an element \( a \in C^I \). If \( C_1 \lor \ldots \lor C_n \) is a disjunctive normal form for \( C \), then there exist direct successors \( N_1, \ldots, N_n \) of root such that \( N_i.\text{type} = "\Omega" \) and \( N_i.\text{concept-description} = C_i \) for \( i, 1 \leq i \leq n \). Since \( a \in (C_1 \lor \ldots \lor C_n)^I \)
... \cup C_n)^l$ there exists an $i$ such that $a \in C_i$. We choose the node $N_i$ as direct successor for the root in our instance. Since $a \in C_i$, there is no clash in $C_i$ and hence $N_i$.value \neq \text{“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 $A_1 \cap ... \cap A_m \cap \exists R_1.E_1 \cap ... \cap \exists R_k.E_k \cap \forall S_1.D_1 \cap ... \cap \forall S_m.D_m$. Since $a \in C_i$, we have $a \in \exists R_1.E_1$ for $i$, $1 \leq i \leq l$. There exists $b_i \in T_l$ such that $(a, b_i) \in R_i$ and $b_i \in E_i$ $(1 \leq i \leq l)$ Furthermore $b_i \in D_j$, if $R_i = S_j$ $(1 \leq j \leq k)$ since $a \in (\forall S_j.D_j)$. Since $b_i \in T_l$, we also have $b_i \in C_l$. Hence $b_i \in (C \cap E_i \cap \bigcap R_i = S_j D_j)^l$. By construction of the concept tree the node $N_i$ has exactly $l$ direct successors $M_1, ..., M_l$. Every $M_i$ has exactly one direct successor $M_i'$ with

- $M_i'.\text{concept-description} = C \cap E_i \cap \bigcap R_i = S_j D_j$
- $M_i'.\text{type} = \text{“}\cup\text{”}$

and for each $i$ we have an element $b_i \in (C \cap E_i \cap \bigcap R_i = S_j D_j)^l$. We can now proceed with these nodes of type “\cup” 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 $T_l$ are the nodes $N_1, ..., N_n$ in $S$ such that $N_i$.type = “\cap” and $N_i$.value \neq “cycle”.

(2) Interpretation of role symbols: Let $N$ be an element of $T_l$. Then $N$.type = “\cap” 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 $M_1, ..., M_n$ 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 $M_i'.\text{type} = \text{“}\cap\text{”}$, and every $M_i'$ has exactly one direct successor $M_i''$ with $M_i''.\text{type} = \text{“}\forall\text{”}$. If $M_i''.\text{value} = \text{“solved”}$ or $M_i'.\text{value} = \text{“null”}$, then we set $(N, M_i'') \in R_i$. Note that $M_i'' \in T_l$. Otherwise $M_i''.\text{value} = \text{“cycle”}$ and there exists a predecessor $P$ of $M_i''$ such that $P$.type = “\forall”, $P$.value \neq “cycle”, and $P$.concept-description is equal to $M_i''$.concept-description modulo associativity, commutativity, and idempotency. In this case we set $(N, P) \in R_i$. Note that $P \in T_l$.

(3) Interpretation of concept symbols: Suppose $N \in T_l$. Then $N$.concept-description is of the form $A_1 \cap ... \cap A_m \cap \exists R_1.E_1 \cap ... \cap \exists R_k.E_k \cap \forall S_1.D_1 \cap ... \cap \forall S_m.D_m$. If $A_i$ $(1 \leq i \leq m)$ is a non-negated concept symbol, then we set $N \in A_i$. Note that $N$.value \neq “clash”, and hence there does not exist an $A_j$ with $A_j = \neg A_i$. Thus $N \in (A_1 \cap ... \cap A_m)^l$.

Before we can show that this canonical interpretation satisfies $T_l = C_l$, we need one more definition and an observation concerning concept descriptions of nodes with type “\cap”.

19
The depth \( \tau \) 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 = “\( \sqcap \)” , then \( N\).concept-description has the form \( C \sqcap C_{\text{rest}} \). Let \( C_I \sqcup \ldots \sqcup C_n \) be a disjunctive normal form of \( C \sqcap C_{\text{rest}} \). There exist direct successors \( N_1, \ldots, N_n \) of \( N \) with \( N_i\).type = “\( \sqcap \)” and \( N_i\).concept-description = \( C_i \). Obviously, \( C \) subsumes \( C \sqcap C_{\text{rest}} \), and \( C \sqcap C_{\text{rest}} \) subsumes each \( C_i \). Thus, if \( N_i \) is a node in \( T \) with \( N_i\).type = “\( \sqcap \)” , then \( \text{root.concept-description} \) (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 \( T^I \). First we prove the following claim.

**Claim**: Let \( I \) be the canonical interpretation induced by \( S \) and let \( N \in T^I \). If \( D \) is a subconjunction of \( N\).concept-description, then \( N \in D^I \).

**Proof**: (by induction on the depth \( \tau(D) \).) Let \( N \in T^I \). 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 \( A_1 \sqcap \ldots \sqcap A_m \sqcap \exists R_1.E_1 \sqcap \ldots \sqcap \exists R_l.E_l \sqcap \forall S_1.D_1 \sqcap \ldots \sqcap \forall S_k.D_k \). Now let \( D \) be a subconjunction of \( N\).concept-description.

- If \( \tau(D) = 0 \), then \( D \) is of the form \( A_{i_1} \sqcap \ldots \sqcap A_{i_n} \). By definition of the interpretation of concept symbols we have \( N \in (A_{i_1} \sqcap \ldots \sqcap A_m)^I \), which implies \( N \in (A_{i_1} \sqcap \ldots \sqcap A_{i_n})^I \).
- If \( \tau(D) > 0 \), let \( D = D_{i_1} \sqcap \ldots \sqcap D_{i_n} \). We have to show for all \( i, 1 \leq i \leq n \), that \( N \in D_i^I \). If \( \tau(D_j) < \tau(D) \) 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 \( M \in T^I \), \( (N, M) \in R^I \) 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 = “\( \sqcap \)” . Furthermore, \( N''\).concept-description = \( C \sqcap E \sqcap \bigcap R = S_j D_j \). Suppose \( E_1 \sqcup \ldots \sqcup E_n \) is a disjunctive normal form of \( E \). It is easy to see that, if \( C \sqcup \ldots \sqcup C_m \) 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 = “\( \sqcap \)” , \( N'' \) 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 = "®" 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 $\pi(E_j) \leq \pi(E) < \pi(\forall R.E)$ 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 $M \in T^I$ 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 $N''.type = "\Omega"$ and $N''.concept-description = C_1 \bigwedge \ldots \bigwedge C_n$. Suppose $E_1 \bigvee \ldots \bigvee E_n$ 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 $M$.value = "cycle", then $M \in T^I$ and by definition of $I$ we have $(N, M) \in R^I$. Since $\pi(E_j) < \pi(\forall R.E)$ 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 $M'.type = "\Omega"$ and $M'.concept-description$ is equal to $M$.concept-description modulo idempotency and commutativity. By definition of $I$ we know that $M' \in T^I$ 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 $N \in (\exists R.E)^I$. $\square$

Let $I$ be the canonical interpretation induced by $S$ and let $N \in T^I$. 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 $N \in (N$.concept-description$)^I$. Since $N$.type = "®", we know that $C$ subsumes $N$.concept-description (as mentioned above). Hence for every $N \in T^I$ we have $N \in C^I$. Thus $T^I = C^I$. $\square$

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 \( T^I = C_0^I \) and \( T^I \neq \emptyset \). To do this the function generates an element which is in \( T^I \). In general, constraints imposed on this element force the function to generate further elements which are in \( T^I \). Since we want to construct an interpretation with \( T^I = C_0^I \), 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 = "∃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 = \( \forall S_1.D_1 \). Recall that the initialization step of the algorithm was done in a way such that \( \text{root.concept-description} = C_0 \). Thus we just have to omit \( \text{root.concept-description} \) in the definition of \( N'\).concept-description, i.e., we modify Extend-∃R-node such that \( N'\).concept-description = \( C \sqcap \exists R.C \sqcap \forall S_j.D_j \). 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 \( \pi(N\).concept-description\) > \( \pi(M\).concept-description\) if \( N \) is a predecessor of \( M \). As a consequence, \( N\).value ≠ "cycle" for every node \( N \) in \( T \). Furthermore, if \( \pi(C_0) = n \), the longest path in \( S \) contains at most \( n \) nodes of type "\( \sqcap \)", and hence at most \( 3 \times n \) 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 $\text{Satisfiability}(C_0)$. 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 $\text{Satisfiability}(C_0)$. Consider the canonical interpretation $I$ induced by $S$ and $N \in T^I$. 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 $N \in T^I$. Then $N \in (N.\text{concept-description})^I$. As mentioned above, if $N$ and $M$ are elements of $T^I$, then $\pi(N.\text{concept-description}) > \pi(M.\text{concept-description})$ if $N$ is a predecessor of $M$. Thus one can easily prove the claim by induction on the depth $\pi(N.\text{concept-description})$. 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 $\pi(N.\text{concept-description})$ is not necessarily greater than $\pi(M.\text{concept-description})$.

Let $C_1 \cup \ldots \cup C_n$ be a disjunctive normal form of $C_0$. Then there exists a node in $S$ such that $N.\text{type} = "\odot"$ and $N.\text{value} \neq \text{"cycle"}$. Hence $N \in T^I$. 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 $C_1 \cup \ldots \cup C_n$ 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


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.

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
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 Bürckert: 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 Klückner, 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
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 via Terminological 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: Conflict 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-92-19
Stefan Dittrich, Rainer Hoch: Automatische, Deskriptorbasierte Unterstützung der Dokumentanalyse zur Fokussierung und Klassifizierung von Geschäftsbriefen
107 Seiten

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-arten 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, Gernrod 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
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
264 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).