Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research Report
RR-93-07

Concept Logics with Function Symbols

Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux

April 1993

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

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

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

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, SEMA Group, Siemens and Siemens-Nixdorf. 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
- Computer Linguistics
- Programming Systems
- Deduction and Multiagent Systems
- Document Analysis and Office Automation.

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.

Friedrich J. Wendl
Director

Concept Logics with Function Symbols

Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux

DFKI-RR-93-07

This work has been supported by a grant from The Federal Ministry for Research and Technology (FKZ ITW-8903 0 and ITW-9201).

Concept Logics with Function Symbols

Hans-Jürgen Bürckert Bernhard Hollunder
Armin Laux
Deutsches Forschungszentrum für künstliche Intelligenz (DFKI)
Stuhlsatzenhausweg 3
W-6600 Saarbrücken 11, Germany
e-mail: {buerckert, hollunder, laux}@dfki.uni-sb.de

Abstract

Constrained resolution allows the incorporation of domain specific problem solving methods into the classical resolution principle. Firstly, the domain specific knowledge is represented by a restriction theory. One then starts with formulas containing so-called restricted quantifiers, written as $\forall_{X:R} F$ and $\exists_{X:R} F$, where X is a set of variables and the restriction R is used to encode domain specific knowledge by filtering out some assignments to the variables in X. Formulas with restricted quantifiers can be translated into clauses which consist of a (classical) clause together with a restriction. In order to attain a refutation procedure which is based on such clauses one needs algorithms to decide satisfiability and validity of restrictions w.r.t. the given restriction theory.

Recently, concept logics have been proposed where the restriction theory is defined by terminological logics. However, in this approach problems have been assumed to be given as sets of clauses with restrictions and not in terms of formulas with restricted quantifiers. For this special case algorithms to decide satisfiability and validity of restrictions have been given.

In this paper we will show that things become much more complex if problems are given as sets of formulas with restricted quantifiers. The reason for this is due to the fact that Skolem function symbols are introduced when translating such formulas into clauses with restrictions. While we will give a procedure to decide satisfiability of restrictions containing function symbols, validity of such restrictions turns out to be undecidable. Nevertheless, we present an application of concept logics with function symbols, namely their use for generating (partial) answers to queries.

Contents

1 Introduction

Deductive systems which are based on the classical resolution principle in general do not allow the incorporation of methods for domain specific problem solving: Firstly, a set of (first-order) formulas is transformed into a set of clauses by a domain-independent transformation algorithm. Then these clauses are tested on unsatisfiability by a more or less blind search. Recently, a logic with restricted quantifiers has been introduced and, building upon this, constrained resolution allows the incorporation of domain specific knowledge into the resolution principle ([Bür91, Bür93, BBH+90, BHL93]). The main idea behind this approach is to represent domain-specific knowledge in a so-called restriction theory and to extend the classical quantifiers as follows. If R is a restriction, i.e. an open formula over the signature of the restriction theory, formulas $\forall_{\{x\}:R}F$ and $\exists_{\{x\}:R}F$ are allowed which can be read as "F holds for all elements satisfying R" and "F holds if there exists an element which satisfies R", respectively. When transforming such formulas into clauses one obtains so-called RQ-clauses which are of the form $C \mid\mid R$, where C is a clause and R is a restriction. In order to prove unsatisfiability of the obtained RQ-clauses set one can use the constrained resolution principle to derive empty RQ-clauses $\Box \mid\mid R_1, \ldots, \Box \mid\mid R_n$. This process is iterated until for each model of the restriction theory there is a an empty RQ-clause $C \mid\mid R$ whose restriction R is satisfied by that model.

In order to profit from the constrained resolution principle one has to select a restriction theory that provides both, a powerful language to represent domain specific knowledge and (efficient) algorithms to decide satisfiability and validity of restrictions. In this paper we investigate the use of terminological logics as restriction theory. To represent knowledge of a problem domain in this formalism one starts with given atomic concepts and roles, and defines new concepts using the operators provided by a so-called concept language. For several reasons, terminological logics seem to be a good candidate to define a restriction theory. On the one hand, they have widely been accepted to be a knowledge representation formalism applicable to a large class of problem domains. On the other hand, most terminological logics are a decidable and well-investigated fragment of first-order logics.

Indeed, the use of terminological logics in the constrained resolution principle has already been investigated in [BBH+90] and was called concept logics there. In this approach problems have been assumed to be given as a set of RQ-clauses together with a restriction theory, and algorithms for deciding satisfiability and validity of restrictions have been given. However, problems are usually not given by RQ-clauses but by formulas (with restricted quantifiers), and we will show that things become much more complicated in this case. The reason for this lies in the fact that function symbols may be introduced via Skolemization. We thus need algorithms to test satisfiability

and validity of restrictions containing function symbols, which was not addressed in [BBH+90].

This paper is organized as follows. In Section 2 syntax and semantics of a logic with restricted quantifiers are given and the constrained resolution principle is recalled. Besides from Skolem function symbols, which are introduced by Skolemization, we allow RQ-formulas to contain user-specified function symbols. Since all variables in RQ-formulas are constrained, it is appropriate to take restrictions into account when interpreting these function symbols and thus each of them may have a function declaration. Section 3 presents the concept language $\mathcal{ALC}$ and it is shown how constrained resolution can be instantiated if the restriction theory is given by the terminological logic based on this concept language. In [BHL93] we proposed an optimization of the constrained resolution provided that the restriction theory satisfies a certain condition which is satisfied when using $\mathcal{ALC}$. That means we do not need to consider RQ-clauses with arbitrary satisfiable restrictions, but only RQ-clauses whose restrictions satisfy an additional condition, called constraint unifiability.

In Section 4 we investigate algorithms for testing constraint unifiability and validity of restrictions containing function symbols. We will present an algorithm to decide constraint unifiability of a restriction. However, testing validity of a set of restrictions turns out to be undecidable. Though this result shows that there is no algorithm to decide whether the derived empty RQ-clauses are sufficient to prove unsatisfiability of an RQ-clause set, there is an interesting application of concept logics with function symbols which is presented in Section 5, namely using concept logics for query answering. The main idea of this approach is to translate facts and a negated query, both given as RQ-formulas, into a set $\mathcal{C}$ of RQ-clauses. Each restriction R of a derived empty RQ-clause can then be read as an answer "the query is successful whenever R is satisfied".

1.1 Related Work

The idea of clauses with restrictions has already been introduced by Höhfeld and Smolka [HS88] who did not aim at a refutation procedure, but at query answering for logic programming. The basis of our work is [Bür91], [Bür93] where a logic with restricted quantifiers and the constrained resolution principle have been introduced. Constrained resolution generalizes several approaches of building in theories into resolution based deduction systems. An important example for building in such theories are sorted logics (see, e.g., [Obe62], [Wal87], [Wal88], [Sch89], [WO90], [Coh92], [Wei92]).

The use of terminological logics as restrictions, so-called concept logics, has been discussed in [BBH+90]. In that approach, problems have been assumed to be given as a set of RQ-clauses without function symbols, together with a terminological restriction

theory. For this case, algorithms for deciding satisfiability and validity or restrictions have been given.

The transformation of formulas with restricted quantifiers into a set of RQ-clauses while preserving satisfiability has been given in [BHL93]. Building upon this transformation procedure a refutation procedure for formulas with restricted quantifiers is given there, which is instantiated for concept logics with function symbols in the present paper.

2 A Logic with Restricted Quantifiers

In this section we recall a logic with restricted quantifiers (RQ for short). Syntax and semantics of RQ-formulas are given in Subsections 2.1 and 2.2, respectively. Subsection 2.3 introduces a resolution principle for RQ-clauses.

2.1 Syntax

A **signature** Σ consists of three pairwise disjoint sets of symbols: a set F_Σ of function symbols, a set V_Σ of variables, and a set P_Σ of predicate symbols. The notions of (ground) terms and formulas are defined as usual. Given a formula F with exactly the free variables $x_1, \ldots, x_n$, then $\forall F$ denotes the **universal closure** $\forall x_1 \ldots \forall x_n F$ of F and $\exists F$ denotes the **existential closure** $\exists x_1 \ldots \exists x_n F$ of the formula F.

We now introduce restricted quantification systems (RQS) to represent domain specific (or background) knowledge and RQ-signatures which extend an RQS by foreground language symbols. An RQS consists of three parts, that is, a signature Δ, a set of (open) Δ-formulas, which define the syntactically allowed background formulas, and a restriction theory, which represents the possible interpretations of the restrictions.

A **restricted quantification system (RQS)** $\mathcal{R}$ consists of

- a signature Δ with equality,
- a set of (open) Δ-formluas, the **restriction formulas** or **restrictions** which are closed under conjunction and instantiation of variables, and
- a theory over Δ, the **restriction theory**.

The restriction theory can be given either as a set of axioms or as a set of Δ-structures. Note that in the latter case the restrictions need not have a first-order axiomatization.

A **signature with restricted quantifiers** or an **RQ-signature** Σ consists of an RQS $\mathcal{R}$ together with an additional set of predicate symbols $\mathcal{P}_\Sigma$ and an additional set of function symbols $\mathcal{F}_\Sigma$, both disjoint from the symbols of Δ of the RQS. In order to simplify our notation we will use the prefix "Σ-" if we denote objects—terms, atoms, formulas, etc.—that are built upon symbols out of $\mathcal{F}_\Sigma$ and $\mathcal{P}_\Sigma$, and variables out of V_Δ only.

Given such an RQ-signature Σ we now define formulas with restricted quantifiers w.r.t. Σ. Therefore we allow quantifiers to be indexed not only by variables, but by pairs of a variable set X and a restriction formula R. These extended quantifiers are written as $\forall_{X:R}$ and $\exists_{X:R}$, and we call them **restricted quantifiers**. Note that the restrictions represent background information and the Σ-formulas foreground information, respectively. We define **RQ-formulas over** Σ by

1. all Σ-atoms are RQ-formulas,
2. $\forall_{X:R}F$ and $\exists_{X:R}F$ are RQ-formulas, where F is an RQ-formula, R is a restriction, and X contains at least the free variables in R,
3. $F \wedge G$, $F \vee G$, $\neg F$, $F \rightarrow G$, $F \leftrightarrow G$ are RQ-formulas, where F, G are RQ-formulas, and x is a variable.

In the second definition the formula F may contain free variables of X that are now bounded by the restricted quantifiers $\forall_{X:R}$ or $\exists_{X:R}$. The formula R is called the restriction for the variables of X and can be seen as a sieve that filters out the possible assignments of elements to these variables.

An **RQ-clause** (or **constrained clause**) consists of a Σ-clause C, the so-called **kernel**, together with a restriction R. Such a clause, written as $C \,||\, R$, represents the RQ-formula $\forall_{X:R}C$, where X contains exactly the free variables in C and R. If C is empty we call it an **empty RQ-clause**, written as $\square \,||\, R$.

Without loss of generality we can assume that the set $\mathcal{F}_\Sigma$ of foreground function symbols is empty. We can always achieve this by modifying an RQS as follows: the first step is to extend the background signature Δ by the symbols in $\mathcal{F}_\Sigma$. But after doing this we are neither allowed to use these symbols in our foreground language (since $\mathcal{F}_\Sigma$ is empty now), nor to use them in a restriction, because the set of restrictions does not contain any formula over these function symbols up to now.[1] Of course, we want to be able to express the same facts before and after the extension of F_Δ. To guarantee this we use **unfolding**, i.e., we replace every Σ-term, e.g. $f(x)$, by a new variable z, and then we enlarge the set of restrictions by the equation $z = f(x)$. Therefore the second

[1] Note that the original signature Δ of the RQS did not contain any of these additional function symbols, and restrictions are (open) formulas over this original signature Δ.

step is to extend the set of restrictions such that it contains in addition all equations of the form $x = t$, where x is a variable and t is a Σ-term.

2.2 Semantics

We first recapitulate the semantics of first-order formulas (with equality) by using Σ-structures and Σ-assignments. Then we extend these Σ-structures to RQ-structures, which gives a semantics of RQ-formulas.

Let Σ be a signature. A **Σ-structure** $\mathcal{A}$ consists of a non-empty universe $U^{\mathcal{A}}$ and maps each n-ary function (predicate) symbol to an n-ary function (relation). A **Σ-assignment** α maps each variable $x \in V_\Sigma$ to an element $u \in U^{\mathcal{A}}$. This mapping is extended to terms as usual: if $t \equiv f(t_1, \ldots, t_n)$ is an arbitrary term, then we define $\alpha(f(t_1, \ldots, t_n)) := f^{\mathcal{A}}(\alpha(t_1), \ldots, \alpha(t_n))$.

Satisfiability of formulas without restricted quantifiers is defined as usual. We write $(\mathcal{A}, \alpha) \models F$ if F is satisfied by the Σ-structure $\mathcal{A}$ and the Σ-assignment α. A Σ-structure $\mathcal{A}$ is a **Σ-model** of a formula F, written $\mathcal{A} \models F$, if and only if $(\mathcal{A}, \alpha) \models F$ holds for every Σ-assignment α. A formula F is called **valid** if and only if every Σ-structure $\mathcal{A}$ is a Σ-model of F. Two formulas are **equivalent** iff they have exactly the same models.

To simplify our notation we will use some abbreviations: $F[x_1, \ldots, x_n]$ denotes a formula F that contains at least the free variables $x_1, \ldots, x_n$. With $F[x \leftarrow t]$ we denote the formula which is obtained from F by replacing every free occurrence of the variable x by the term t. Analogously, $F[x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n]$ denotes the replacement of every free variable x_i by the term t_i, $i = 1, \ldots, n$. If u is an element of the universe, then $\alpha_{[x \leftarrow u]}$ denotes the Σ-assignment α with the exception of the explicit assignment of u to x. As above, this abbreviation is extended to $\alpha_{[x_1 \leftarrow u_1, \ldots, x_n \leftarrow u_n]}$, where $x_1, \ldots, x_n$ and $u_1, \ldots, u_n$ are variables and elements of the universe, respectively.

The semantics of restricted quantifiers can be given by **relativization**, that is, one can transform any RQ-formula into an equivalent first-order formula by replacing

$$\begin{array}{lll} \forall_{X:R} F & \text{by} & \forall x_1 \ldots \forall x_n (R \rightarrow F) \\ \exists_{X:R} F & \text{by} & \exists x_1 \ldots \exists x_n (R \wedge F) \end{array}$$

where $X = \{x_1, \ldots, x_n\}$ is a set of variables.

We will use an alternative characterization which maintains the separation of foreground and background symbols. Let Σ be an RQ-signature over the RQS $\mathcal{R}$. An **RQ-structure over** Σ is a Σ-structure $\mathcal{A}$ such that the restriction of $\mathcal{A}$ to Δ, written $\mathcal{A}|_\Delta$, is one of the Δ-models in $\mathcal{R}$. As we assumed that Σ introduces only new predicate symbols but no function symbols, we obtain the different RQ-structures by

expanding every model of the restriction theory with all possible interpretations of theses new predicate symbols. If the restriction theory is given by a Δ-axiomatization, RQ-structures are exactly those structures that satisfy the axioms of $\mathcal{R}$, considered as formulas over the extended signature.

Let $\mathcal{A}$ be an RQ-structure over the RQ-signature Σ, α be a Σ-assignment, and $X = \{x_1, \ldots, x_n\}$ be a set of variables. **RQ-satisfiability** of an RQ-formula F is defined as an extension of the satisfiability of first-order formulas by:

$$
\begin{array}{lll}
(\mathcal{A}, \alpha) \models \forall_{X:R} F & \text{iff} & \text{for all } u_1, \ldots, u_n \in U^{\mathcal{A}} \text{ with} \\
 & & (\mathcal{A}|_\Delta, \alpha_{[x_1 \leftarrow u_1, \ldots, x_n \leftarrow u_n]}) \models R \text{ holds} \\
 & & (\mathcal{A}, \alpha_{[x_1 \leftarrow u, \ldots, x_n \leftarrow u_n]}) \models F \\
(\mathcal{A}, \alpha) \models \exists_{X:R} F & \text{iff} & \text{there are } u_1, \ldots, u_n \in U^{\mathcal{A}} \text{ such that} \\
 & & (\mathcal{A}|_\Delta, \alpha_{[x_1 \leftarrow u_1, \ldots, x_n \leftarrow u_n]}) \models R \text{ and} \\
 & & (\mathcal{A}, \alpha_{[x_1 \leftarrow u_1, \ldots, x_n \leftarrow u_n]}) \models F
\end{array}
$$

A closed RQ-formula F is **RQ-satisfiable** if and only if there is an RQ-structure $\mathcal{A}$ such that $(\mathcal{A}, \alpha) \models F$ for each Σ-assignment α. In this case, $\mathcal{A}$ is a **Σ-model** of F, written $\mathcal{A} \models F$. The RQ-formula F is called **RQ-valid** if and only if every RQ-structure $\mathcal{A}$ is a Σ-model of F.

Given a restriction R, we say R is **RQ-satisfiable** if and only if there exists an RQ-structure which satisfies the existential closure of R (that means iff there exists a Δ-model in $\mathcal{R}$ that satisfies $\exists\ R$). Analogously, a restriction R is called **RQ-valid** if and only if the existential closure of R is satisfied by each RQ-structure.

2.3 RQ-Resolution

Given a set $\mathcal{C}$ of RQ-clauses we need an appropriate resolution calculus, which allows one to check $\mathcal{C}$ on RQ-unsatisfiability. Such a calculus is given in [Bür91] and consists of two rules, called RQ-resolution and RQ-factor rule. Using a predicate *redundant* (cf. [BHL93]) generalized forms of these rules have been given, namely

RQ-resolution rule (RR)

$$
\frac{\begin{array}{l} p(x_1, \ldots, x_n) \vee C_1 \vee \ldots \vee C_k \,||\, R \\ \neg p(y_1, \ldots, y_n) \vee D_1 \vee \ldots \vee D_m \,||\, S \end{array}}{C_1 \vee \ldots \vee C_k \vee D_1 \vee \ldots \vee D_m \,||\, R \wedge S \wedge \Gamma} \quad \text{if not } \mathit{redundant}\,(R \wedge S \wedge \Gamma)
$$

where Γ is the conjunction of the equations $x_i = y_i$, $i = 1, \ldots, n$.

The inferred RQ-clause is called **RQ-resolvent**.

RQ-factor rule (FR)

$$\frac{p(x_1^1,\ldots,x_n^1) \vee \ldots \vee p(x_1^m,\ldots,x_n^m) \vee C_1 \vee \ldots \vee C_k \,||\, R}{p(x_1^1,\ldots,x_n^1) \vee C_1 \vee \ldots \vee C_k \,||\, R \wedge \Gamma} \quad \text{if not } \mathit{redundant}\ (R \wedge \Gamma)$$

where Γ is the conjunction of the equations $x_i^1 = x_i^j$, $i = 1,\ldots,n$ and $j = 2,\ldots,m$.

The inferred RQ-clause is called **RQ-factor**.

If $C \,||\, R$ is an RQ-clause, then *redundant* $(C \,||\, R)$ is true iff this RQ-clause is redundant to prove RQ-unsatisfiability of a set of RQ-clauses.[2] Thus, given a fixed RQS, the predicate *redundant* has to be instantiated in an appropriate manner in order to guarantee refutation completeness of the RQ-resolution principle. For example, one can instantiate this predicate by *redundant* $(C \,||\, R)$ iff R is RQ-unsatisfiable. In this case, refutation completeness is guaranteed for arbitrary restricted quantification systems (see [Bür91]). Another instantiation of this predicate will be presented in Section 3.

For sake of simplicity we will sometimes use constant symbols in the kernels of RQ-clauses, though we assumed that the foreground language introduces new predicate symbols only.[3] For example, we simply write

$$q(a,y) \,||\, p(y) \qquad \text{instead of} \qquad q(x,y) \,||\, x = a \wedge p(y).$$

An **RQ-resolution step** $\mathcal{C} \rightarrow \mathcal{C}'$ transforms a set $\mathcal{C}$ of RQ-clauses into a set $\mathcal{C}'$ by either choosing two suitable clauses in $\mathcal{C}$ and adding their RQ-resolvent, or by adding an RQ-factor to $\mathcal{C}$. An **RQ-derivation** is a possibly infinite sequence $\mathcal{C}_0 \rightarrow \mathcal{C}_1 \rightarrow \mathcal{C}_2 \rightarrow \ldots$ of RQ-resolution steps. An **RQ-refutation** of a set $\mathcal{C}_0$ of RQ-clauses is an RQ-derivation which starts with $\mathcal{C}_0$ and satisfies the following condition: For each model $\mathcal{A}$ of the restriction theory there is an RQ-clause set $\mathcal{C}_i$ in the derivation containing an empty clause $\Box \,||\, R$, whose restriction is satisfied by this model, i.e. $\mathcal{A} \models \exists R$. In contrast to the classical resolution principle we need in general more than one empty RQ-clause to prove the RQ-unsatisfiability of an RQ-clause set (cf. [BHL93]).

3 Concept Logics

Terminological logics have are used as a knowledge representation formalism in Artificial Intelligence. To represent knowledge of a problem domain in this formalism one

[2] Cf. the classical resolution principle where tautological clauses are redundant to prove unsatisfiability of a clause set.

[3] Note that in Subsection 2.1 we required the set $\mathcal{F}_\Sigma$ of additional foreground function symbols to be empty.

starts with given atomic concepts and roles, and defines new concepts using the operations provided by a so-called concept language. Thereby, concepts can be considered as unary predicates which are interpreted as sets of objects, and roles as binary predicates which are interpreted as binary relations between objects. Examples for atomic concepts may be *woman* or *queen*, and for roles *likes-the-same-clothes-as*. The use of terminological logics in restrictions of RQ-formulas is called **concept logic**.

In this section we will present the terminological logic which uses operators of a distinguished concept language, called $\mathcal{ALC}$. This logic has widely been accepted to be an adequate knowledge representation mechanism for a large class of problem domains, and it is a decidable and well-investigated subclass of first-order logics. In Subsection 3.1 we introduce syntax and semantics of the language $\mathcal{ALC}$. In 3.2 syntax and semantics of function declarations are given, while 3.3 presents a restricted quantification system over $\mathcal{ALC}$. A refutation procedure for concept logics, which is an instantiation of the general refutation procedure presented in [BHL93], is given in Subsection ??.

3.1 The Concept Language $\mathcal{ALC}$

The concept language $\mathcal{ALC}$ provides two formalisms to describe a particular problem domain: a terminological formalism to represent taxonomical knowledge by defining concepts, and an assertional formalism which can be used to describe concrete objects. Therefore we assume in the following a signature $\Delta = (F_\Delta, V_\Delta, P_\Delta)$ to be given, where

- F_Δ consists of a set of function symbols,
- V_Δ consists of a set of variables, and
- P_Δ consists of a set of unary predicates (atomic concepts), the symbols $\top$ and $\bot$, and a set of binary predicates (roles).

In $\mathcal{ALC}$, concepts can be built up from atomic concepts, the **top concept** $\top$, the **bottom concept** $\bot$, and roles with the help of the operators $\sqcap$ (concept conjunction), $\sqcup$ (concept disjunction), $\neg$ (concept negation), $\forall R.C$ (value-restriction), and $\exists R.C$ (exists-restriction) as follows:

1. Each atomic concept, $\top$, and $\bot$ are concepts.
2. If C and D are concepts, then $C \sqcap D$, $C \sqcup D$, and $\neg C$ are concepts.
3. If C is a concept and R is a role, then $\forall R.C$ and $\exists R.C$ are concepts.

Let $\mathcal{A}$ be a Δ-structure. Then the semantics of roles and concepts is given by

- $A^{\mathcal{A}} \subseteq U^{\mathcal{A}}$ for each atomic concept A in P_Δ.
- $R^{\mathcal{A}} \subseteq U^{\mathcal{A}} \times U^{\mathcal{A}}$ for each role R in P_Δ.
- $\top^{\mathcal{A}} = U^{\mathcal{A}}$ and $\bot^{\mathcal{A}} = \emptyset$.
- $[C \sqcap D]^{\mathcal{A}} = C^{\mathcal{A}} \cap D^{\mathcal{A}}$, $[C \sqcup D]^{\mathcal{A}} = C^{\mathcal{A}} \cup D^{\mathcal{A}}$, and $[\neg C]^{\mathcal{A}} = U^{\mathcal{A}} \setminus C^{\mathcal{A}}$
- The value- and exists-restriction are interpreted by

$$\begin{aligned} [\forall R.C]^{\mathcal{A}} &= \{u \in U^{\mathcal{A}} \mid \forall u' : (u, u') \in R^{\mathcal{A}} \rightarrow u' \in C^{\mathcal{A}}\} \\ [\exists R.C]^{\mathcal{A}} &= \{u \in U^{\mathcal{A}} \mid \exists u' : (u, u') \in R^{\mathcal{A}} \wedge u' \in C^{\mathcal{A}}\} \end{aligned}$$

where R is a role and C, D are concepts.

Using these operators, we can, e.g., define the concept of women who like the same clothes as a queen by : *woman* $\sqcap \exists$ *likes-the-same-clothes-as.queen*, if *woman* and *queen* are atomic concepts, and *likes-the-same-clothes-as* is a role. In 3.3 we will show how to use concepts to restrict the possible assignments to variables by quantifying over elements in a given concept only.

Note, that concepts can be seen as first-order formulas (without equality) with one free variable. For example, the concept $\forall R.C$ represents the formula $\forall y\ (R(x, y) \rightarrow C(y))$ where x is a free variable.

In the following we will sometimes need a concept C to be in **negation normal form**, i.e., negation signs in C only occur immediately in front of atomic concepts. It is easy to show that each concept C can be transformed into an equivalent concept in negation normal form. For example, $\neg\forall R.C$ is rewritten as $\exists R.\neg C$ (cf. [Hol90]).

The terminological knowledge of a problem domain can be defined by a **terminology (TBox)** which consists of a finite set of **terminological axioms**, i.e., expressions of the form $A = C$ where A is an atomic concept and C is a concept. For example, if *woman*, *person*, and *female* are atomic concepts we can define "men are persons who are not female" by the terminological axiom

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

If *child* is a role we then can describe "not female persons with only female children" by the expression *man* $\sqcap \forall$ *child.female*. That means, terminological axioms allows one to define abbreviations for concepts, and hence helps one to keep the definitions of concepts simple. However, for reason of simplicity of presentation we do not consider terminological axioms, i.e., we assume each concept only to be built up by atomic concepts and roles but not by abbreviations for concepts. This assumption does not influence expressive power. For technical details see, e.g., [Hol90].

The **assertional formalism** of $\mathcal{ALC}$ allows us to introduce concrete objects by stating that they are instances of concepts and roles. Thereby, each ground term over F_Δ is called an **object**. For example, if *John* is a constant, and *father* is a unary function symbol in F_Δ, then *John* as well as *father (John)* are objects. In general, only constants are allowed as objects in the concept language $\mathcal{ALC}$. But this view is not sufficient for us since, by Skolemization, function symbols may occur in the restrictions of RQ-clauses (cf. Section 3 in [BHL93]). The assertional formalism is given by concept instances and role instances which are defined as follows:

1. If o is an object and C a concept, then $o : C$ is a **concept instance**.
2. If o and o' are objects and R is a role, then oRo' is a **role instance**.

A Δ-structure $\mathcal{A}$ maps objects to elements of the universe $U^{\mathcal{A}}$ and satisfies $o : C$ iff $o^{\mathcal{A}} \in C^{\mathcal{A}}$, and oRo' iff $(o^{\mathcal{A}}, o'^{\mathcal{A}}) \in R^{\mathcal{A}}$. Concept instances and role instances are called **assertional axioms**. A finite set of assertional axioms is called an **ABox**. We say an ABox A is **consistent** iff there exists a Δ-structure $\mathcal{A}$ which satisfies every axiom in A, written as $\mathcal{A} \models \mathsf{A}$.

With these axioms we can, e.g., define that Elizabeth is a queen and that Mary likes the same clothes as her mother by

$$\textit{Elizabeth} : \textit{queen} \qquad \text{and}$$
$$\textit{Mary likes-the-same-clothes-as mother (Mary)}$$

respectively.

3.2 Function Declarations

By definition, RQ-formulas may contain n-ary function symbols. Consider, for example, the RQ-formula

$$\forall_{\{x\}:\textit{human}} \textit{male}(\textit{father}(x))$$

where *human* is a unary restriction, *male* is a predicate, and *father* a function symbol. Up to now we interpreted these function symbols free, i.e., we assumed a Σ-structure $\mathcal{A}$ to map each n-ary function symbol f to a function $f^{\mathcal{A}} : U^{\mathcal{A}} \times \ldots \times U^{\mathcal{A}} \mapsto U^{\mathcal{A}}$, where $U^{\mathcal{A}}$ is the universe of $\mathcal{A}$. Indeed, since all variables in RQ-formulas are constrained, it is appropriate to take restrictions into account when interpreting function symbols. In the above example it is more intuitive to define the unary function symbol *father* to map from *human* to *human* instead of mapping arbitrary elements of the universe to the universe.

Thus, we extend the restriction theory by function declarations for the function symbol occurring in RQ-formulas. If f is an n-ary function symbol and $R_1, \ldots, R_n, R$

are restrictions, a **function declaration** is of the form

$$f : R_1 \times \ldots \times R_n \mapsto R.$$

We assume that there is exactly one declaration for each function symbol. A straightforward semantics of these function declaration could be defined by $\forall x_1 \ldots \forall x_m \big(R_1(x_1) \wedge \ldots \wedge R_n(x_m)\big) \rightarrow R(f(x_1, \ldots, x_m))$. We additionally assume the range of a function to be non-empty what strongly simplifies the algorithm for testing satisfiability of $\mathcal{ALC}$ restrictions (cf. 4.3). More formally, a Σ-structure $\mathcal{A}$ satisfies the function declaration $f : R_1 \times \ldots \times R_n \mapsto R$ iff $\mathcal{A}$ satisfies

1. $\forall x_1 \ldots \forall x_m \big(R_1(x_1) \wedge \ldots \wedge R_n(x_m)\big) \rightarrow R(f(x_1, \ldots, x_m))$ and
2. $R \neq \emptyset$.

In [BHL93] a method for transforming a set of RQ-formulas into a set of RQ-clauses while preserving RQ-satisfiability has been described. In this transformation restricted existential quantifiers are eliminated via Skolemization. Thereby, for each n-ary Skolem function symbol f a Skolem declaration is added to the restriction theory. A **Skolem declaration** is of the form

$$(R \neq \emptyset) \rightarrow f : R_1 \times \ldots \times R_n \mapsto R$$

where $R_1, \ldots, R_n, R$ are restrictions (cf. 3.2 in [BHL93]). A Σ-structure $\mathcal{A}$ satisfies this Skolem declaration iff

1. $\mathcal{A}$ satisfies $R = \emptyset$ or
2. $\mathcal{A}$ satisfies $\forall x_1 \ldots \forall x_m \big(R_1(x_1) \wedge \ldots \wedge R_n(x_m)\big) \rightarrow R(f(x_1, \ldots, x_m))$.

When looking at the transformation of RQ-formulas into a set of RQ-clauses, the following property can easily be shown.[4] If $C \parallel S$ is an RQ-clause where S contains a Skolem function symbol f with the Skolem declaration $(R \neq \emptyset) \rightarrow f : R_1 \times \ldots \times R_n \mapsto R$, then S contains a conjunct $R \neq \emptyset$, where $R \neq \emptyset$ is an abbreviation for the formula $\exists x\ R(x)$. For example, the transformation of the RQ-formula

$$\forall_{\{x\}:R_1} \exists_{\{y\}:R_2}\ p(x, y)$$

results in the RQ-clause set

$$\begin{array}{rcl} p(x, y) & \parallel & R_1(x) \wedge y = f_{Skolem}(x) \wedge R_2 \neq \emptyset \\ \square & \parallel & R_1(z) \wedge R_2 = \emptyset \end{array}$$

[4] Cf. quantifier splitting (Subsection 3.1) and Skolemization procedures (Subsection 3.2) in [BHL93].

and an extension of the restriction theory by the Skolem declaration

$$(R_2 \neq \emptyset) \rightarrow f_{Skolem} : R_1 \mapsto R_2.$$

That means, whenever there is an RQ-clause whose restriction contains a function symbol or a Skolem function symbol with range R, we only have to take Σ-structures $\mathcal{A}$ with $R^{\mathcal{A}} \neq \emptyset$ into consideration when testing satisfiability or validity of this restriction.

3.3 A Restricted Quantification System over $\mathcal{ALC}$

In this subsection we show how the concept language $\mathcal{ALC}$ can be used to define a restricted quantification system RQS. We therefore have to say how the signature, how the restrictions, and how the restriction theory are to be defined.

Firstly, in order to use concepts as restrictions we allow restricted quantifiers of the form $\forall_{\{x\}:C}$ and $\exists_{\{x\}:C}$ where C is a concept, i.e., a unary predicate. This leads to restrictions of the form $x : C$ where x is a variable. Secondly, we assumed the set $\mathcal{F}_\Sigma$ of foreground function symbols to be empty (see Subsection 2). This can always be obtained by unfolding and leads to restrictions of the form $y = f(t_1, \ldots, t_n)$, where y is a variable, f is a function symbol, and $t_1, \ldots, t_n$ are terms. Finally, when transforming RQ-formulas into a set of RQ-clauses restrictions of the form $C = \emptyset$ and $C \neq \emptyset$ are introduced which are abbreviations for the closed formulas $\forall x\ \neg C(x)$ and $\exists x\ C(x)$, respectively. This is the third kind of restriction we take into consideration. In the following definition of an RQS over $\mathcal{ALC}$ these three types of restrictions are introduced.[5] We thus obtain the following definition of a **restricted quantification system over $\mathcal{ALC}$** which is given by

- A signature $\Delta = (F_\Delta, V_\Delta, P_\Delta)$ as described above.
- A set of restrictions which are Δ-formulas of the form

$$\begin{aligned} x : C \quad & \textbf{(containment)} \\ y = f(t_1, \ldots, t_n) \quad & \textbf{(equational restriction)} \\ C = \emptyset, C \neq \emptyset \quad & \textbf{(closed restriction)} \end{aligned}$$

  where C is a concept, x, y are variables, and $t_1, \ldots, t_n$ are Δ-terms. These restrictions are called **$\mathcal{ALC}$-restrictions**. Note, that we assumed restrictions to be closed under conjunction and instantiation of variables.

- A restriction theory which is given by an ABox A and a set F of **declarations**, i.e. function declarations or Skolem declarations, such that

[5] Note, that RQ-formulas must not obtain restrictions of the form $C = \emptyset$ or $C \neq \emptyset$. Furthermore, equations in restrictions of RQ-clauses can only be of the form $y = f(t_1, \ldots, t_n)$ where y is a *new* variable. This is due to the fact that equations are introduced only by unfolding.

1. each Skolem function symbol in F_Δ has exactly one Skolem declaration in F, and
2. each function symbol in F_Δ which is not a Skolem function symbol has exactly one function declaration in F.

Thus, a Δ-structure $\mathcal{A}$ is an RQ-structure iff it satisfies A and each declaration in F. We then write $\mathcal{A} \models \mathsf{A} \cup F$.

3.4 Constrained Resolution applied to Concept Logics

In this subsection we will show an important property restricted quantification systems over the concept language $\mathcal{ALC}$ have, namely that the *redundant* predicate of the RQ-resolution and the RQ-factor rule can be instantiated by constraint unifiability, which will be defined below.

In [BHL93], a general refutation procedure for a set of RQ-formulas has been presented. The main idea of this refutation procedure is as follows. Firstly, the RQ-clauses are transformed into a set of RQ-clauses while preserving RQ-satisfiability. To prove RQ-unsatisfiability of an RQ-clause set $\mathcal{C}$ then the RQ-resolution and the RQ-factor rule are used which successively add new RQ-clauses to $\mathcal{C}$. This process is iterated until a set of empty RQ-clauses $\Box \mid\mid R_1, \ldots, \Box \mid\mid R_n$ is derived such that $R_1 \vee \ldots \vee R_n$ is RQ-valid.

In the same paper it has been shown that constraint unification can be used to instantiate the *redundant* predicate of the RQ-resolution and the RQ-factor rule if the RQS satisfies a certain condition, called (TM). Let $R = E_1 \wedge \ldots \wedge E_n \wedge N_1 \wedge \ldots \wedge N_m$ be a restriction, where $E_1, \ldots, E_n$ are the equational restrictions in R. Then R is **constraint unifiable with substitution** σ iff there exists an RQ-structure $\mathcal{A}$ and a Σ-assignment α such that

1. $E_1 \wedge \ldots \wedge E_n$ is unifiable with σ, and
2. $(\mathcal{A}, \alpha) \models \sigma N_1 \wedge \ldots \wedge \sigma N_m$.

If the restriction R is constraint unifiable with σ we call σ a **constraint unifier** of R. If a constraint unifier σ is a most general unifier of the equational restrictions in R we call σ a **constraint mgu** of R. We say R is **constraint unifiable** iff there is a substitution σ such that R is constraint unifiable with σ.

Example 3.1 Let A and B be predicate symbols of a background signature Δ, and let R be the restriction $(y = f(x)) \wedge A(x) \wedge B(y)$, where f is a function symbol. Obviously, the only equational restriction in R, $y = f(x)$, is unifiable with mgu $\sigma = \{y \leftarrow f(x)\}$.

1. Let f have the function declaration $f : A \mapsto \neg B$. Since function declarations are part of the restriction theory, each RQ-structure $\mathcal{A}$ has to satisfy $f : A \mapsto \neg B$, i.e., $f^{\mathcal{A}}(u) \notin B^{\mathcal{A}}$ if $u \in A^{\mathcal{A}}$. By definition, R is constraint unifiable with σ iff there exists an RQ-structure $\mathcal{A}$ and a Δ-assignment α such that $(\mathcal{A}, \alpha) \models \sigma(A(x)) \wedge \sigma(B(y))$. This is the case iff $\alpha(x) \in A^{\mathcal{A}}$ and $f^{\mathcal{A}}(\alpha(x)) \in B^{\mathcal{A}}$. Because of the function declaration such a pair $(\mathcal{A}, \alpha)$ cannot exist, i.e., R is not constraint unifiable with σ.

2. If f has the function declaration $f : A \mapsto A$, then R is constraint unifiable with σ iff there exists an RQ-structure $\mathcal{A}$ and a Δ-assignment α such that $\alpha(x) \in A^{\mathcal{A}}$ and $f^{\mathcal{A}}(\alpha(x)) \in B^{\mathcal{A}}$. Because of the function declaration we know $f^{\mathcal{A}}(\alpha(x)) \in A^{\mathcal{A}}$ if $\alpha(x) \in A^{\mathcal{A}}$. Thus R is constraint unifiable with σ if there exists an RQ-structure $\mathcal{A}$ such that $A^{\mathcal{A}} \cap B^{\mathcal{A}} \neq \emptyset$. ■

In order to formulate the condition (TM) we need the notion of a term-model. If $\mathcal{S}$ is a set of formulas over some signature Δ, then $\mathcal{A}$ is a **term-model** of $\mathcal{S}$ over Δ iff $\mathcal{A} \models \mathcal{S}$, all elements in the universe $U^{\mathcal{A}}$ are interpretations of Δ-ground terms, and two different Δ-ground terms denote different elements in $U^{\mathcal{A}}$.

The property (TM) the RQS has to satisfy is given by: Let $\mathcal{T}$ be a satisfiable restriction theory and let $R_1, \ldots, R_n$ be a set of restrictions, then (TM) is defined by

(TM)
$$\mathcal{T} \models \exists(R_1 \vee \ldots \vee R_n)$$
iff
there exists a term-model $\mathcal{A}$ of $\mathcal{T}$ such that $\mathcal{A} \models \exists(R_1 \vee \ldots \vee R_n)$

The following theorem, proved in [BHL93], shows the connection between condition (TM) and constraint unifiability.

Theorem 3.2 *Let $\mathcal{T}$ be a satisfiable restriction theory, and let $R_1, \ldots, R_n$ be a set of restrictions such that*

1. *$\mathcal{T}$ does not contain (explicitly or implicitly) equations.*

2. *Each restriction R_i can be written as $E_i \wedge N_i$, where E_i is a conjunction of equations and N_i does neither contain (explicitly or implicitly) equations nor disequations.*

Then $\mathcal{T}$ and $R_1, \ldots, R_n$ satisfy condition (TM).

In an RQS over $\mathcal{ALC}$ the restriction theory is given by an ABox A together with a set F of declarations. Obviously, neither A nor F contain (explicitly or implicitly)

equations since A consists of concept and role instances and F of (Skolem) function declarations only.[6] Furthermore, each $\mathcal{ALC}$-restriction R is given as a conjunction of containments $x : C$, closed restrictions $C = \emptyset$, $C \neq \emptyset$, and equational restrictions $y = f(t_1, \ldots, t_n)$. Hence, each $\mathcal{ALC}$-restrcition can be written as $E \wedge N$ where E is a conjunction of equations and N does neither contain (explicitly or implicitly) equations nor disequations. Thus, by the above theorem, we know that an RQS over $\mathcal{ALC}$ satisfies condition (TM).

We therefore can apply the next theorem, also proved in [BHL93], which tells us that only RQ-clauses with a constraint unifiable restriction need to be derived in order to test RQ-unsatisfiability of an RQ-clause set with $\mathcal{ALC}$-restrictions.

Theorem 3.3 *Let Δ be a signature, let $\mathcal{T}$ be a satisfiable set of Δ-formulas, and let $R_1, \ldots, R_n$ be restrictions such that condition (TM) is satisfied. If each restriction R_i is given by $E_{i_1} \wedge \ldots \wedge E_{i_k} \wedge N_{i_1} \wedge \ldots \wedge N_{i_l}$, where $E_{i_1}, \ldots, E_{i_k}$ are the equational restrictions in R_i, and if each conjunction $E_{i_1} \wedge \ldots \wedge E_{i_k}$ is unifiable with the mgu σ_i, then $\mathcal{T} \models \exists(R_1 \vee \ldots \vee R_n)$ iff $\mathcal{T} \models \exists(\sigma_1 N_1 \vee \ldots \vee \sigma_n N_n)$.*

Summing up, if we use restricted quantifiers $\forall_{\{x\}:C}$ and $\exists_{\{x\}:C}$ where C is a concept, a set $\mathcal{S}$ of RQ-formulas can be tested on RQ-unsatisfiability as follows. Firstly, $\mathcal{S}$ is transformed into a set $\mathcal{C}$ of RQ-clauses while preserving RQ-satisfiability. An algorithm for this is described in Section 3 of [BHL93]. Then $\mathcal{C}$ can be tested on RQ-unsatisfiability via constrained resolution, where only RQ-clauses with a constraint unifiable restriction need to be considered. We thus obtain an instantiation of the general refutation procedure in Section 4 of [BHL93] which is given in Figure 1.

The problems of how to test constraint unifiability of an $\mathcal{ALC}$-restriction and how to check RQ-validity of a set of $\mathcal{ALC}$-restrictions will be investigated in the next section.

4 Testing Constraint Unifiability and RQ-Validity of $\mathcal{ALC}$-Restrictions

In order to give an algorithm which tests constraint unifiability of an $\mathcal{ALC}$-restriction we use the notions of containment sets and (admissible) containment combinations, which are given in Subsection 4.1. We will show that testing constraint unification can be reduced to a top consistency test of a given concept D_0, i.e., to a test whether there

[6] Usually, objects are assumed to satisfy the unique name assumption, i.e., different objects are mapped to different elements of the universe. However, if $\mathcal{A}$ is a model of an ABox A, there exists a model $\mathcal{A}'$ of A which satisfies the unique name assumption (and vice versa). This is due to the fact that cardinality of $C^{\mathcal{A}}$ cannot be restricted in $\mathcal{ALC}$ for a concept C. Thus, if $o^{\mathcal{A}} = o'^{\mathcal{A}} = u \in U^{\mathcal{A}}$, one can extend $U^{\mathcal{A}}$ by a "duplicate" u' of u and define $o^{\mathcal{A}'} = u$ and $o'^{\mathcal{A}'} = u'$.

Input: A set $\mathcal{S}$ of RQ-formulas and a restriction theory $\mathcal{T}$ which is given by an ABox A and a set of declarations

Output: *RQ-unsatisfiable* iff $\mathcal{S}$ is not RQ-satisfiable
RQ-satisfiable or the algorithm does not terminate, else

Initializing

Transform $\mathcal{S}$ into a set $\mathcal{C}$ of RQ-clauses while preserving RQ-satisfiability as described in Section 3 of [BHL93], and let $\mathcal{T}'$ be the modified restriction theory. Remove an RQ-clause $C \,||\, R$ from $\mathcal{C}$ if R is not constraint unifiable.

Testing

1. If $\Box \,||\, R_1, \ldots, \Box \,||\, R_n$ are empty RQ-clauses in $\mathcal{C}$ such that $R_1 \vee \ldots \vee R_n$ is RQ-valid w.r.t. $\mathcal{T}'$, then return *RQ-unsatisfiable*.

2. If there is an RQ-clause to which the RQ-factor rule (FR) is applicable, but has not yet been applied, then apply the RQ-factor rule to this RQ-clause and add the RQ-factor to $\mathcal{C}$.

3. Find two RQ-clauses which can be resolved against each other by the RQ-resolution rule (RR) (of course the two RQ-clauses have to be chosen by a fair strategy). If there does not exist such a pair of RQ-clauses, return *RQ-satisfiable*. Otherwise, add the RQ-resolvent to $\mathcal{C}$ (after an appropriate variable renaming) and goto 1.

Figure 1: The refutation procedure.

exists a Σ-structure $\mathcal{A}$ such that $D_0^{\mathcal{A}} = \top^{\mathcal{A}}$ $(= U^{\mathcal{A}})$. An algorithm for this test is given in Subsection 4.2. Building upon this, an algorithm for testing constraint unifiability of an $\mathcal{ALC}$-restriction is given in 4.3, and its termination, correctness, and completeness is proved. Finally, in 4.4, we show that validity of a set of $\mathcal{ALC}$-restrictions is undecidable.

4.1 Admissible Containment Combinations

Suppose we want to test constraint uniafibility of an $\mathcal{ALC}$-restriction R. By definition, R can be written as $E_1 \wedge \ldots \wedge E_n \wedge N_1 \wedge \ldots \wedge N_m$ where $E_1, \ldots, E_n$ are the equational restrictions in R. We then have to test whether (i) the equations $E_1, \ldots, E_n$ are unifiable and (ii) if $E_1, \ldots, E_n$ are unifibale and σ is the most general unifier of $E_1, \ldots, E_n$ we furthermore have to test whether there exists an RQ-structure $\mathcal{A}$ such that $\mathcal{A} \models \sigma N_1 \wedge \ldots \wedge \sigma N_m$.

Algorithms for testing unifiability of a set of equations $E_1, \ldots, E_n$ and for computing the most general unifier σ of $E_1, \ldots, E_n$ are well-known. Thus, we still need an algorithm for testing RQ-satisfiability of the restriction $\sigma N_1 \wedge \ldots \wedge \sigma N_m$. That means, we need an algorithm for testing the existential closure of conjunctions of Δ-formulas on RQ-satisfiability which have the form

- $t : C$
- $C = \emptyset, C \neq \emptyset$

where t is a Δ-term, and C is a concept. Restrictions of this form are called **equation free restrictions.**[7] Thereby, restrictions of the form $t : C$ arise from containments $x : C$ by unifying the variable x with a term t, e.g., in $x : C \wedge x = f(y)$.

There are two straightforward possibilities to simplify equation free restrictions. Firstly, for testing RQ-satisfiability of the existential closure $\exists x R$ of an equation free restriction R it is sufficient to test RQ-satisfiability of the restriction R' which arises from $\exists x R$ by removing the quantifier $\exists x$, replacing each occurrence of x in R by a new constant a, and adding the function declaration $a :\mapsto \top$ to the set F of function declarations. The restriction R' is then called the **ground version of** R.

Analogously, we can assume that the ground version of an equation free restriction does not contain closed restrictions of the form $C \neq \emptyset$. Note, that a restriction of the form $R \wedge C \neq \emptyset$ is equivalent to $\exists x(R \wedge x : C)$ and thus to $R \wedge a : C$ where a is a new constant.

For example, testing RQ-satisfiability of the restriction $f(x, y) : \neg D \wedge x : D \wedge B \neq \emptyset \wedge E = \emptyset$ is equivalent to testing RQ-satisfiability of the restriction $f(a, b) : \neg D \wedge a : D \wedge c : B \wedge E = \emptyset$ where a, b, c are new constants.

Thus, we assume in the following restrictions to be given by conjunctions of Δ-formulas which have the form $t : C$ or $C = \emptyset$ where t is a ground term and C is a concept. For testing constraint unifiability we now need an algorithm which tests such restrictions on RQ-satisfiability.

In order to guarantee that the the range of each function which has a function declaration in F is non-empty (cf. semantics of function declarations), we assume in the following that an ABox A contains a containment $a : C$ for each function declaration $f : C_1 \times \ldots \times C_n \mapsto C$ in F. If this is not the case, we extend A by $a : C$ where a is a new object. Because of the semantics of function declarations this does not influence RQ-satisfiability.

[7] Note that $C = \emptyset$ and $C \neq \emptyset$ are abbreviations for the Δ-formulas $\forall x \neg C(x)$ and $\exists x C(x)$, respectively.

We are now going to give such an RQ-satisfiability algorithm, the main idea of which is as follows. Suppose a containment $f(t_1, \ldots, t_n) : D$ and a function declaration $f : C_1 \times \ldots \times C_n \mapsto C$ to be given. Then, a Δ-structure $\mathcal{A}$ satisfies both, the function declaration and the containment iff

1. $[f(t_1, \ldots, t_n)]^{\mathcal{A}} \in D^{\mathcal{A}}$ and
2. $t_i^{\mathcal{A}} \in C_i^{\mathcal{A}}$ for $i = 1, \ldots, n$ implies $[f(t_1, \ldots, t_n)]^{\mathcal{A}} \in C^{\mathcal{A}}$.

Thus, in order to find a Δ-structure $\mathcal{A}$ which satisfies $f(t_1, \ldots, t_n) : D$ w.r.t. the function declaration $f : C_1 \times \ldots \times C_n \mapsto C$, we can do the the following. Firstly, for each argument t_i of f we choose non-deterministically whether $t_i^{\mathcal{A}} \in C_i^{\mathcal{A}}$ or $t_i^{\mathcal{A}} \in [\neg C_i]^{\mathcal{A}}$ holds. Analogously, we decide non-deterministically whether or not $[f(t_1, \ldots, t_n)]^{\mathcal{A}} \in C^{\mathcal{A}}$ holds. The only restriction on these decisions is: If we choose $t_i^{\mathcal{A}} \in C_i^{\mathcal{A}}$ for $i = 1, \ldots, n$, then we choose $[f(t_1, \ldots, t_n)]^{\mathcal{A}} \in C^{\mathcal{A}}$.

Now suppose, we made such decisions $t_i^{\mathcal{A}} \in \tilde{C}_i^{\mathcal{A}}$ where $\tilde{C}_i$ is either C_i or $\neg C_i$, and $f(t_1, \ldots, t_n) : \tilde{C}$ where $\tilde{C}$ is either C or $\neg C$. Then there exists a Δ-structure which satisfies both, the containment $f(t_1, \ldots, t_n) : D$ and the function declaration $f : C_1 \times \ldots \times C_n \mapsto C$, iff the restriction

$$t_1 : \tilde{C}_1 \wedge \ldots \wedge t_n : \tilde{C}_n \wedge f(t_1, \ldots, t_n) : \tilde{C} \wedge f(t_1, \ldots, t_n) : D$$

is satisfiable.

Let us define this more formally. If A is an ABox and R is a restriction, then the **term set** of A and R is given by the set all (sub)terms in $\mathsf{A} \cup \{R\}$ with a leading function symbol. For example, if R is given by $a : A \wedge f(b, g(c)) : B$, the term set of R is given by $\{a, f(b, g(c)), b, g(c), c\}$.

For each term t in a function term set S we now construct a **non-deterministic concept set** $ncs(t)$ of t (w.r.t. S). The intuitive idea of a set $ncs(t)$ is to determine all concepts C for which we non-deterministically have to decide whether $t \in C$ or $t \in \neg C$ (restrictions on the possible decisions are formulated below). These sets are defined by

1. If t is a constant with the function declaration $t :\mapsto C$ or with the Skolem declaration $(C \neq \emptyset) \rightarrow t :\mapsto C$, then $ncs(t) := \{C\} \cup \{D_i \mid g(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_n)$ is a term in S and g has the function declaration $g : D_1 \times \ldots \times D_n \mapsto D$ or the Skolem declaration $(D \neq \emptyset) \rightarrow g : D_1 \times \ldots \times D_n \mapsto D\}$.
2. If t is a term $f(t_1, \ldots, t_n)$ where f has the function declaration $f : C_1 \times \ldots \times C_n \mapsto C$ or the Skolem declaration $(C \neq \emptyset) \rightarrow f : C_1 \times \ldots \times C_n \mapsto C$, then $ncs(t) := \{C\} \cup \{D_i \mid g(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_m)$ is a term in S and g has the function declaration $g : D_1 \times \ldots \times D_m \mapsto D$ or the Skolem declaration $(D \neq \emptyset) \rightarrow g : D_1 \times \ldots \times D_n \mapsto D\}$.

Obviously, the non-deterministic concept set of each term t in a term set is not empty. Let now S be a term set and let for each term t in S the non-deterministic concept set be given by $ncs(t)$. Now we represent the non-deterministic decisions for t in so-called containment sets. If, for some term t in S, $ncs(t) = \{A_1, \ldots, A_n\}$ then each set $\{t : \tilde{A}_1, \ldots, t : \tilde{A}_n\}$ is called a **containment set** of t (w.r.t. S) if each $\tilde{A}_i$ is either A_i or $\neg A_i$. For example, if $ncs(f(a)) = \{B, C\}$, then $\{f(a) : B, f(a) : C\}$, $\{f(a) : \neg B, f(a) : C\}$, $\{f(a) : B, f(a) : \neg C\}$, $\{f(a) : \neg B, f(a) : \neg C\}$ are all the containment sets of $f(a)$.

A set which consists for each term t in a term set S of one containment set is called containment combination of S. More formally, if $t_1, \ldots, t_m$ are exactly the terms in S and $cont\ (t_i)$ is a containment set of t_i, then the set $\{cont(t_1), \ldots, cont(t_m)\}$ is called a **containment combination** of S. Finally, we are not interested in all possible containment combinations, but only in those combinations which are compatible with the declarations in F: A containment combination $\mathcal{C}$ is an **admissible containment combination** iff for each term $f(t_1, \ldots, t_n)$ in S holds: if f has the function declaration $f : D_1 \times \ldots \times D_n \mapsto D$ and $t_i : D_i$ is in $\mathcal{C}$ for $i = 1, \ldots, n$, then $f(t_1, \ldots, t_n) : D$ is in $\mathcal{C}$. Observe that if a is a constant with the function declaration $a :\mapsto C$, each admissible containment combination contains $a : C$. Furthermore, if a Skolem function symbol f_S occurs in a term set of an ABox A and a restriction R, this Skolem function symbol occurs in R but not in A. Let now the Skolem declaration of f_S be given by $(D \neq \emptyset) \rightarrow f_S : D_1 \times \ldots \times D_n \mapsto D$ $(n \geq 0)$. Then the restriction R contains a conjunct $D \neq \emptyset$ (cf. Subsection 3.2). Thus, for testing satisfiability of R w.r.t. A and F we only have to consider Σ-structures $\mathcal{A}$ such that $D^{\mathcal{A}} \neq \emptyset$.[8]

Example 4.1 Let the set F contain the function declarations

$$f : A \times B \mapsto C$$
$$g : \neg B \mapsto D$$
$$a :\mapsto \top$$
$$b :\mapsto \top.$$

Furthermore, let an ABox A be given by $\{a : A, f(b, g(b)) : C\}$ and a restriction R by

$$f(b, g(b)) : B \wedge g(b) : E$$

Then the term set S of A and R is $\{a, f(b, g(b)), b, g(b)\}$. The non-deterministic concept set of the terms in S are given by:

$$\begin{aligned} ncs(a) &= \{\top\} \\ ncs(b) &= \{\top, A, \neg B\} \\ ncs(g(b)) &= \{B, D\} \\ ncs(f(b, g(b))) &= \{C\} \end{aligned}$$

[8]Thus, as an optimization, admissible containment combinations could be defined such that they contain $f_S(t_1, \ldots, t_n) : D$ whenever f_S appears in a term set and has the Skolem declaration $(D \neq \emptyset) \rightarrow f_S : D_1 \times \ldots \times D_n \mapsto D$.

Now the set

$$\{a : \top, b : \top, b : A, b : \neg B, g(b) : B, g(b) : D, f(b, g(b)) : C\}$$

is an admissible containment combination, while the set

$$\{a : \top, b : \top, b : A, b : \neg B, g(b) : B, g(b) : D, f(b, g(b)) : \neg C\}$$

is *not* an admissible containment combination, since $f(b, g(b)) : C$ has to be in an admissible containment combination $\mathcal{C}$ if $b : A$ and $g(b) : B$ are in $\mathcal{C}$ (cf. the function declaration of f). ■

In order to check RQ-satisfiability of a restriction R w.r.t. to a given ABox A and a set F of declarations we will use admissible containment combinations as follows. If S is the term set of A and R, and $\mathcal{C}$ is an admissible containment combination of S, then we will test whether or not there exists a Δ-structure $\mathcal{A}$ which satisfies both the ABox which is given by $\mathsf{A} \cup \mathcal{C}$, and R. In other words, we test whether R is RQ-satisfiable w.r.t. $\mathsf{A} \cup \mathcal{C}$. We will show that R is RQ-satisfiable (w.r.t. A and F) iff there exists an admissible containment combination $\mathcal{C}$ of S such that R is RQ-satisfiable w.r.t. $\mathsf{A} \cup \mathcal{C}$.

Since the ABox which is given by $\mathsf{A} \cup \mathcal{C}$ may contain role instances, an algorithm which tests RQ-satisfiability of a restriction R w.r.t. $\mathsf{A} \cup \mathcal{C}$ must test satisfiability of a set of

- concept instances $t : C$, where t is a ground term and C is a concept,
- role instances sRt, where s, t are ground terms and R is a role, and
- closed restrictions $C = \emptyset$, where C is a concept.

Thereby, concept instances and role instances may occur in the ABox $\mathsf{A} \cup \mathcal{C}$, and closed restrictions and concept instances may occur in the ground version of the equation free restriction.

Example 4.2 Consider the ABox $\mathsf{A} = \{a : C, f(a, a) : B\}$, the set $F = \{f : A \times B \mapsto C, a :\mapsto \top, b :\mapsto \top, c :\mapsto \top, d :\mapsto \top\}$ of function declarations, and the restriction R, given by

$$f(b, c) : \neg D \wedge b : D \wedge c : D \wedge d : B \wedge E = \emptyset.$$

The term set of A and R is $\{a, f(a, a), f(b, c), b, c, d\}$. Obviously, the set

$$\mathcal{C} = \{a : \top, a : A, a : B, b : \top, b : A, c : \top, c : B, d : \top, f(a, a) : \neg C, f(b, c) : \neg C\}$$

is an admissible containment combination. In order to test RQ-satisfiability of R w.r.t. $\mathsf{A} \cup \mathcal{C}$ we have to check whether there exists a Δ-structure $\mathcal{A}$ such that

1. $\mathcal{A}$ satisfies the ABox which is given by $\mathsf{A} \cup \mathcal{C}$, i.e., $\mathcal{A}$ satisfies the concept instances $\{a : \top, a : A, a : B, b : \top, b : A, c : \top, c : B, d : \top, f(a,a) : \neg C, f(b,c) : \neg C\}$, and

2. $\mathcal{A}$ satisfies R, i.e., $\mathcal{A} \models f(b,c) : \neg D \wedge b : D \wedge c : D \wedge d : B \wedge E = \emptyset$.

Note that the test does no longer take the function declarations into account. ■

An algorithm for testing RQ-satisfiability of a restriction R w.r.t. $\mathsf{A} \cup \mathcal{C}$ is given in the next subsection.

4.2 Testing Top Consistency

Algorithms for testing satisfiability of an ABox, i.e., of a set of variable-free concept instances and role instances are well-known (see, e.g., [Hol90]). For additionally testing satisfiability of closed restriction of the form $C = \emptyset$, we introduce the notion of top consistency: a concept D_0 is **top consistent** w.r.t. an ABox A iff there exists a Δ-structure $\mathcal{A}$ such that $\mathcal{A} \models \mathsf{A}$ and $D_0^{\mathcal{A}} = \top^{\mathcal{A}} (= U^{\mathcal{A}})$. The next lemma shows that testing satisfiability of an ABox together with a set of closed restrictions of the form $C = \emptyset$ can be reduced to a top consistency test.

Lemma 4.3 *Let A be an ABox, and let $C_1 = \emptyset, \ldots, C_n = \emptyset$ be a set of closed restrictions. Then there exists a Δ-structure $\mathcal{A}$ such that $\mathcal{A}$ satisfies A and $\mathcal{A} \models C_1 = \emptyset \wedge \ldots \wedge C_n = \emptyset$ iff the concept $\neg C_1 \sqcap \ldots \sqcap \neg C_n$ is top consistent w.r.t. A.*

Proof: The closed restriction $C_i = \emptyset$ means that $\neg C_i$ is equivalent to the top concept $\top$. Thus, $C_1 = \emptyset \wedge \ldots \wedge C_n = \emptyset$ is satisfied iff each of the concepts $[\neg C_1]^{\mathcal{A}}, \ldots, [\neg C_n]^{\mathcal{A}}$ is equivalent to $\top^{\mathcal{A}}$, and hence iff $[\neg C_1 \sqcap \ldots \sqcap \neg C_n]^{\mathcal{A}}$ is equivalent to $\top^{\mathcal{A}}$. □

An algorithm for testing top consistency of a concept D_0 w.r.t. a given ABox A is given in [Lau92]. In this algorithm only constants are allowed as objects. However, since we only want to handle ground terms as objects and since equations between these ground terms cannot be expressed by an ABox, we can handle these ground terms exactly like constants in this algorithm.

The top consistency algorithm is based on the notion of a constraint system. A **constraint system** is finite non-empty set of constraints of the form $a : C$ or aRb, where C is a concept, R is a role, and a, b are constants. A constraint system S contains a **clash** iff (i) S contains two concept instances $a : A$ and $a : \neg A$ where a is an constant and A is an atomic concept or (ii) S contains a constraint $a : \bot$ for some constant a. We say S is **clash-free** iff S does not contain a clash. A constraint system S is **satisfiable** iff there exists a Δ-structure $\mathcal{A}$ such that $\mathcal{A} \models s$ for each constraint s in S.

1. $S \rightarrow_{\sqcap} \{a : C_1, a : C_2\} \cup S$

   if $a : C_1 \sqcap C_2$ is in S
   and S does not contain both constraints $a : C_1$ and $a : C_2$.

2. $S \rightarrow_{\sqcup} \{a : D\} \cup S$

   if $a : C_1 \sqcup C_2$ is in S,
   neither $a : C_1$ nor $a : C_2$ is in S, and D is C_1 or C_2.

3. $S \rightarrow_{\forall} \{b : C\} \cup S$

   if $a : \forall R.C$ and aRb are in S
   and $b : C$ is not in S.

4. $S \rightarrow_{\exists_1} \{aRb, b : C, b : D_0^*\} \cup S$

   if $a : \exists R.C$ is in S,
   $D_1, \ldots, D_n$ are exactly the constraints of the form $a : \forall R.D_i$ in S,
   there exists no c such that $c : C, c : D_1, \ldots, c : D_n, c : D_0^*$ are in S,
   and b is a new constant.

5. $S \rightarrow_{\exists_2} \{aRc\} \cup S$

   if $a : \exists R.C$ is in S,
   $D_1, \ldots, D_n$ are exactly the constraints of the form $a : \forall R.D_i$ in S,
   the constraints $c : C, c : D_1, \ldots, c : D_n, c : D_0^*$ are all in S,
   and aRc is not in S.

Figure 2: Propagation rules of the top consistency test.

Given an ABox A and a concept D_0, we say the constraint system S is **induced** by A and D_0 iff $S = \mathsf{A} \cup \{a_0 : D_0^*, a_1 : D_0^*, \ldots, a_n : D_0^*\}$ where a_0 is a new constant, D_0^* is the negation normal form of D_0, and $a_1, \ldots, a_n$ are exactly the objects in A. The top consistency algorithm has a concept D_0 and an ABox A as input and starts with the constraint system S which is induced by A and D_0. It then successively adds new constraints to S by the five propagation rules given in Figure 2 until no more propagation rule is applicable. A constraint system S to which no more rules are applicable is called **complete**.

The following theorem has been proved in [Lau92].

Theorem 4.4 *Let A be an ABox, and let D_0 be a concept. Then D_0 is top consistent w.r.t. A iff there exists a chain $S_0 \rightarrow_1 S_1 \rightarrow_2 \ldots \rightarrow_n S_n$ where*

1. *S_0 is the constraint system which is induced by A and D_0,*

2. $\rightarrow_i$ *is the leftmost propagation rule in the sequel* $\rightarrow_\sqcap$, $\rightarrow_\sqcup$, $\rightarrow_\forall$, $\rightarrow_{\exists_1}$, $\rightarrow_{\exists_2}$ *which is applicable to* S_{i-1},

3. S_n *is complete and clash-free.*

Thus, a concept D_0 is top consistent w.r.t. an ABox A iff a complete and clash-free constraint system S can be obtained from the constraint system which is induced by A and D_0. We assumed each occurrence of a term t to be replaced by a new constant a_t. Obviously, replacing each occurrence of a_t by t in the resulting constraint system S preserves satisfiability and hence we assume S to contain the object t instead of the constant a_t. Analogously to [Lau92] one can show that the following Δ-structure $\mathcal{A}$ satisfies S:

- $U^{\mathcal{A}}$ is the set of objects in S,
- $A^{\mathcal{A}} := \{o \mid o : A \text{ is in } S\}$ for each atomic concept A in S,
- $R^{\mathcal{A}} := \{(o,p) \mid oRp \text{ is in } S\}$ for each role R in S,
- $o^{\mathcal{A}} := o \in U^{\mathcal{A}}$ for each object o in S.

We will call this Δ-structure the **free Δ-structure** of S.

4.3 Testing Constraint Unifiability

Now we are able to give an algorithm for testing RQ-satisfiability of an equation free restriction R w.r.t. an ABox A and a set F of declarations. This algorithm has as input an ABox A, a set F of declarations, and an equation free restriction R. The algorithm is given in Figure 3 and tests whether there exists an admissible containment combination $\mathcal{C}$ of the term set of A and R such that R is RQ-satisfiable w.r.t. $\mathsf{A} \cup \mathcal{C}$. Note that the set F of function symbols is implicitly represented in the containment combination $\mathcal{C}$.

We will now show that the RQ-satisfiability algorithm returns "RQ-satisfiable" iff the restriction R is RQ-satisfiable w.r.t. the ABox A and the set F of function declarations. Firstly, we will show that the algorithm always terminates.

Lemma 4.5 *Let* A *be an ABox,* F *be a set of declarations, and* R *an equation free restriction. The RQ-satisfiability algorithm with input* A, F, *and* R *terminates.*

Proof: Let S be the function term set of A and R, and let $ncs(t)$ be the non-deterministic concept set of t w.r.t. F. Obviously, the set $ncs(t)$ is finite for each term t. Therefore, only a finite number of containment combinations (and thus of admissible containment

1. Let S be the function term set of A and R.
2. Let R' be the ground version of R.
3. For each term t in S let $ncs(t)$ be the non-deterministic concept set of t.
4. Let $C_1 = \emptyset, \ldots, C_n = \emptyset$ be the closed constraints in R', and let $\mathsf{A}_{R'}$ be the ABox which consists of the containments in R'.
5. Check whether there is an admissible containment combination $\mathcal{C}$ of S such that $\neg C_1 \sqcap \ldots \sqcap \neg C_n$ is top consistent w.r.t. the ABox which is given by $\mathsf{A} \cup \mathcal{C} \cup \mathsf{A}_{R'}$.
6. If there exists such an admissible containment combination $\mathcal{C}$, then return *RQ-satisfiable*, else return *not RQ-satisfiable*.

Figure 3: The RQ-satisfiability algorithm for equation free restrictions.

combinations) exists. In [Lau92] has already been shown that the top consistency algorithm terminates. □

The next lemma states that the RQ-satisfiability algorithm is sound.

Lemma 4.6 *Let A be an ABox, F be a set of declarations, R be an equation free restriction, and S be the term set of A and R. Furthermore, let R' be the ground version of R, $C_1 = \emptyset, \ldots, C_k = \emptyset$ be the closed restrictions in R', and $\mathsf{A}_{R'}$ be the set of containments in R'. If there exists an admissible containment combination $\mathcal{C}$ of S such that $\neg C_1 \sqcap \ldots \sqcap \neg C_k$ is top consistent w.r.t. $\mathsf{A} \cup \mathcal{C} \cup \mathsf{A}_{R'}$, then R is RQ-satisfiable (w.r.t. A and F).*

Proof: Since $\neg C_1 \sqcap \ldots \sqcap \neg C_k$ is top consistent w.r.t. $\mathsf{A} \cup \mathcal{C} \cup \mathsf{A}_{R'}$, the top consistency algorithm with input $\neg C_1 \sqcap \ldots \sqcap \neg C_k$ and $\mathsf{A} \cup \mathcal{C} \cup \mathsf{A}_{R'}$ constructs a constraint system, say S^*, which is complete and clash-free (cf. [Lau92]). Thus, if $\mathcal{A}^*$ is the free Δ-structure of S^*, then $\mathcal{A}^* \models s$ for each constraint s in S^*.

Let now $\mathcal{A}$ be a Δ-structure which is identical to $\mathcal{A}^*$, but interprets the function symbols in F as follows:

$$[f(t_1, \ldots, t_n)]^{\mathcal{A}} := \begin{cases} u_1, & \text{if } f(t_1, \ldots, t_n) : C_i \ (i = 1, \ldots, m) \text{ are exactly the} \\ & \text{containments of the form } f(t_1, \ldots, t_n) : C \text{ in } S^* \text{ and} \\ & u_1 \text{ is some element in } [C_1 \sqcap \ldots \sqcap C_m]^{\mathcal{A}} \\ u_2, & \text{if there is no containment of the form} \\ & f(t_1, \ldots, t_n) : C \text{ in } S^* \text{ and } u_2 \text{ is some element in } D^{\mathcal{A}} \end{cases}$$

if f is not a Skolem function symbol and has the function declaration $f : D_1 \times \ldots \times D_n \mapsto D$. Furthermore, if f_S is a Skolem function symbol with the Skolem declaration $(D \neq \emptyset) \rightarrow f : D_1 \times \ldots \times D_n \mapsto D$, $\mathcal{A}$ interprets f_S by

$$[f_S(t_1, \ldots, t_n)]^{\mathcal{A}} := \begin{cases} u_1, & \text{if } f(t_1, \ldots, t_n) : C_i \ (i = 1, \ldots, m) \text{ are exactly the} \\ & \text{containments of the form } f(t_1, \ldots, t_n) : C \text{ in } S^* \text{ and} \\ & u_1 \text{ is some element in } [C_1 \sqcap \ldots \sqcap C_m]^{\mathcal{A}} \\ u_2, & \text{if there is no containment of the form} \\ & f(t_1, \ldots, t_n) : C \text{ in } S^*, \ D^{\mathcal{A}} \neq \emptyset, \\ & \text{and } u_2 \text{ is some element in } D^{\mathcal{A}} \\ u_3, & \text{if there is no containment of the form} \\ & f(t_1, \ldots, t_n) : C \text{ in } S^*, \ D^{\mathcal{A}} = \emptyset, \\ & \text{and } u_3 \text{ is some element in } U^{\mathcal{A}} \end{cases}$$

We already know that $\mathcal{A}^* \models \mathsf{A}$ and $\mathcal{A}^* \models \mathsf{A}_{R'}$ since $\mathcal{A}^*$ satisfies S^*. We still have to show that $\mathcal{A}$ satisfies each function declaration in F.

Firstly, let f be a function symbol which is not a Skolem function symbol and which has the function declaration $f : D_1 \times \ldots \times D_n \mapsto D$. We have to show that (i) $D^{\mathcal{A}} \neq \emptyset$ and (ii) whenever $t_i^{\mathcal{A}} \in D_i^{\mathcal{A}}$ for $i = 1, \ldots, n$, then $[f(t_1, \ldots, t_n)]^{\mathcal{A}} \in D^{\mathcal{A}}$. For (i), remember that for the function declaration of f there is a concept instance $a : D$ in A where a is a new constant. Since $\mathcal{A}$ satisfies S^* and $a : D$ is in S^* (since $a : D$ is in A), $D^{\mathcal{A}} \neq \emptyset$ holds. For (ii) we distinguish two cases: if f is a function symbol which does not occur in S^*, then, by definition of $\mathcal{A}$, $[f(t_1, \ldots, t_n)]^{\mathcal{A}} = u_2 \in D^{\mathcal{A}}$. Suppose, on the other hand, $f(t_1, \ldots, t_n)$ occurs in S^* and $t_i^{\mathcal{A}} \in D_i^{\mathcal{A}}$ for $i = 1, \ldots, n$. By definition of $\mathcal{A}$, $t_i \in D_i^{\mathcal{A}}$ holds iff $t_i : D_i$ is a constraint in S^*. Furthermore, since $f(t_1, \ldots, t_n)$ occurs in S^*, $f(t_1, \ldots, t_n)$ and $t_1, \ldots, t_n$ are elements in the function test set of A and R. Thus, the admissible containment combination $\mathcal{C}$ contains either $t_i : D_i$ or $t_i : \neg D_i$ for each $i \in \{1, \ldots, n\}$. Since we assumed $t_i : D_i$ to be in S^* and since S^* is clash-free, $t_i : \neg D_i$ cannot occur in $\mathcal{C}$ $(i = 1, \ldots, n)$. Finally, because $\mathcal{C}$ is admissible, $f(t_1, \ldots, t_n) : D$ is in $\mathcal{C}$. That means, $f(t_1, \ldots, t_n) : D$ occurs in S^* and thus $[f(t_1, \ldots, t_n)]^{\mathcal{A}} \in D^{\mathcal{A}}$.

Secondly, let f be a Skolem function symbol which has the Skolem declaration $(D \neq \emptyset) \rightarrow f : D_1 \times \ldots \times D_n \mapsto D$. We have to show that either $D^{\mathcal{A}} = \emptyset$, or that $D^{\mathcal{A}} \neq \emptyset$ and $[f(t_1, \ldots, t_n)]^{\mathcal{A}} \in D^{\mathcal{A}}$ whenever each $t_i^{\mathcal{A}} \in D_i^{\mathcal{A}}$. If $D^{\mathcal{A}} = \emptyset$ there is nothing to show. If, on the other hand, $D^{\mathcal{A}} \neq \emptyset$ the argumentation is the same as in the non-Skolem case above.

Summing up, $\mathcal{A}$ satisfies both A and F, i.e., $\mathcal{A}$ is an RQ-structure. Furthermore, $\mathcal{A}$ satisfies $C_i = \emptyset$ for $i = 1, \ldots, k$ since $\neg C_i^{\mathcal{A}^*} = \top^{\mathcal{A}^*}$. Finally, $\mathcal{A}$ satisfies each containment in R' since $\mathcal{A}$ satisfies $\mathsf{A}_{R'}$. That means, $\mathcal{A} \models R'$ and therefore $\mathcal{A} \models \exists R$, i.e., R is RQ-satisfiable (w.r.t. A and F). □

Completeness of the RQ-satisfiability algorithm is shown by the following lemma.

Lemma 4.7 *Let* A *be an ABox, let* F *be a set of declarations, and let* R *be an equation free restriction. Furthermore, let* R' *be the ground version of* R, $C_1 = \emptyset, \ldots, C_k = \emptyset$ *be the closed restrictions in* R', $\mathsf{A}_{R'}$ *the set of containments in* R', *and* S *the term set of* A *and* R. *If there exists a* Δ*-structure* $\mathcal{A}$ *such that* $\mathcal{A} \models \mathsf{A} \cup F$ *and* $\mathcal{A} \models R'$, *then there exists an admissible containment combination* $\mathcal{C}$ *of* S *such that* $\neg C_1 \sqcap \ldots \sqcap \neg C_k$ *is top consistent w.r.t.* $\mathsf{A} \cup \mathcal{C} \cup \mathsf{A}_{R'}$.

Proof: We will show that there exists an admissible containment combination $\mathcal{C}$ of S such that $\mathcal{A} \models \mathcal{C}$. Therefore, let $\{s_1, \ldots, s_m\}$ be the terms in S and for each term s_i in S let $ncs(s_i)$ be the non-deterministic containment set of s_i. Obviously, for each concept D in $ncs(s_i)$ either $s_i^{\mathcal{A}} \in D^{\mathcal{A}}$ or $s_i^{\mathcal{A}} \in [\neg D]^{\mathcal{A}}$ holds. Now, if D is a concept in an arbitrary non-deterministic containment set $ncs(s_i)$, let $\dot{D}$ be D if $s_i^{\mathcal{A}} \in D_i^{\mathcal{A}}$ and let $\dot{D}$ be $\neg D$ if $s_i^{\mathcal{A}} \notin D^{\mathcal{A}}$. Then the set $\mathcal{C} = \{s_i : \dot{D} \mid s_i \text{ is a term in } S \text{ and } D \text{ is a concept in } ncs(s_i)\}$ is a containment combination of S. Furthermore, let $f(t_1, \ldots, t_n)$ be an arbitrary term in S, where f has the function declaration $f : D_1 \times \ldots \times D_n \mapsto D$. Then, if $t_i^{\mathcal{A}} \in D_i^{\mathcal{A}}$ for $i = 1, \ldots, n$, we know $[f(t_1, \ldots, t_n)]^{\mathcal{A}} \in D^{\mathcal{A}}$ because $\mathcal{A}$ satisfies F. Thus, $\mathcal{C}$ is an admissible containment combination of S. Finally, let f_S be a Skolem function symbol with the function declaration $(D \neq \emptyset) \rightarrow f_S : D_1 \times \ldots \times D_n \mapsto D$. If $D^{\mathcal{A}} = \emptyset$ there is nothing to show, and if $D^{\mathcal{A}} \neq \emptyset$ the argumentation is the same as in the case above.

Hence, we can conclude that $\mathcal{A} \models \mathcal{C}$. We already know that $\mathcal{A}$ satisfies A and, because of $\mathcal{A} \models R'$, both $\mathcal{A} \models C_1 = \emptyset \wedge \ldots \wedge C_k = \emptyset$ and $\mathcal{A} \models \mathsf{A}_{R'}$ holds. Summing up, $[\neg C_1]^{\mathcal{A}} = \top^{\mathcal{A}}, \ldots, [\neg C_k]^{\mathcal{A}} = \top^{\mathcal{A}}$, and $\mathcal{A}$ satisfies A, $\mathcal{C}$ and $\mathsf{A}_{R'}$. That means, $\neg C_1 \sqcap \ldots \sqcap \neg C_k$ is top consistent w.r.t. $\mathsf{A} \cup \mathcal{C} \cup \mathsf{A}_{R'}$. □

Summing up the above results, we obtain the following theorem.

Theorem 4.8 *Let* A *be an ABox, let* F *be a set of declarations, and let* R *be an equation free restriction. Then* R *is RQ-satisfiable (w.r.t.* A *and* F*) iff the RQ-satisfiability algorithm with input* A, F, *and* R *returns "RQ-satisfiable".*

Proof: Because of Lemma 4.5 the RQ-satisfiability algorithm with input A, F, and R terminates. By definition, R is RQ-satisfiable iff there exists a RQ-structure $\mathcal{A}$ such that $\mathcal{A} \models \exists R$, i.e., iff there exists a Δ-structure $\mathcal{A}$ which satisfies A and F such that $\mathcal{A} \models \exists R$. Let S be the term set of A and R. Firstly, if $\mathcal{A}$ is an RQ-structure such that $\mathcal{A} \models \exists R$, then $\mathcal{A} \models R'$ where R' is the ground version of R. Then, because of Lemma 4.7, there exists an admissible containment combination $\mathcal{C}$ of S such that $\neg C_1 \sqcap \ldots \sqcap \neg C_k$ is top consistent w.r.t. $\mathsf{A} \cup \mathcal{C} \cup \mathsf{A}_{R'}$ (where $C_1 = \emptyset, \ldots, C_k = \emptyset$ are the closed restrictions in R' and $\mathsf{A}_{R'}$ is the set of containments in R'). In this case, the RQ-satisfiability algorithm returns "RQ-satisfiable".

1. Let $E_1, \ldots, E_n$ and $N_1, \ldots, N_m$ the equational and non-equational restrictions in R, respectively.

2. If $E_1, \ldots, E_n$ is not unifiable, return *not constraint unifiable*, else let σ be the most general unifier of $E_1, \ldots, E_n$.

3. If the equation free restriction $\sigma N_1 \wedge \ldots \wedge \sigma N_m$ is RQ-satisfiable w.r.t. A and F, return *constraint unifiable*, else return *not constraint unifiable*.

Figure 4: The constraint unifiability algorithm.

Conversely, suppose the RQ-satisfiability algorithm returns "RQ-satisfiable". Then there exists an admissible containment combination $\mathcal{C}$ of S such that $\neg C'_1 \sqcap \ldots \sqcap \neg C'_k$ is top consistent w.r.t. $\mathsf{A} \cup \mathcal{C} \cup \mathsf{A}_{R'}$. Thus, R is RQ-satisfiable because of Lemma 4.7. □

Now it is straightforward to give an algorithm for testing constraint unifiability of a restriction R (w.r.t. a given ABox A and a set F of declarations). If R is a restriction and $E_1, \ldots, E_n$ and $N_1, \ldots, N_m$ are the equational and the non-equational restrictions in R, respectively, we firstly have to compute the most general unifier σ of $E_1, \ldots, E_n$ (provided these equations are unifiable). Then we can apply the RQ-satisfiability algorithm to $\sigma N_1 \wedge \ldots \wedge \sigma N_m$ since this restriction is equation free. The constraint unifiability algorithm is given in Figure 4. It has an ABox A, a set F of declarations, and a restriction R as input and returns "constraint unifiable" iff R is constraint unifiable (w.r.t. A and F).

4.4 Testing RQ-validity of $\mathcal{ALC}$-Restrictions

Let us now recall the refutation procedure of Figure 1. In the "testing part" of this algorithm it is tested whether or not some derived empty RQ-clauses are sufficient to prove RQ-unsatisfiability of the input RQ-formulas. More technically, if $\square || R_1, \ldots, \square || R_n$ are derived empty RQ-clauses we have to test $R_1 \vee \ldots \vee R_n$ on RQ-validity w.r.t. the given restriction theory. However, the following theorem shows that this test is undecidable for $\mathcal{ALC}$-restrictions.

Theorem 4.9 *RQ-validity of a set $R_1, \ldots, R_n$ of $\mathcal{ALC}$-restrictions is undecidable.*

Proof: We will show that an algorithm which decides RQ-validity of a set of $\mathcal{ALC}$-restrictions could also be used to decide satisfiability of an arbitrary clause set, what is known to be undecidable.

Let $\mathcal{C} = \{C_1, \ldots, C_n\}$ be a set of clauses over some signature Σ. Furthermore, let A be a new unary predicate and f_p be a new m-ary function symbol for each m-ary predicate symbol p in $\mathcal{C}$. We firstly use the following translation:

$$\begin{array}{rlr} p(t_1, \ldots, t_m) & \text{is mapped to the formula} & A(f_p(t_1, \ldots, t_m)) \\ \neg p(t_1, \ldots, t_m) & \text{is mapped to the formula} & \neg A(f_p(t_1, \ldots, t_m)) \end{array}$$

for each literal $p(t_1, \ldots, t_m)$ or $\neg p(t_1, \ldots, t_m)$ occurring in $\mathcal{C}$. We denote the application of this reduction to $\mathcal{C}$ by $\mathcal{C}^* = \{C_1^*, \ldots, C_n^*\}$. It is easy to verify that $\mathcal{C}$ is unsatisfiable iff $\mathcal{C}^*$ is unsatisfiable: Let $\mathcal{M}$ be a model of $\mathcal{C}$ and let $\mathcal{M}^*$ be defined such that such $\mathcal{M}^* \models A(f_p(t_1, \ldots, t_m))$ iff $\mathcal{M} \models p(t_1, \ldots, t_m)$. Then $\mathcal{M}^*$ obviously satisfies $\mathcal{C}^*$ iff $\mathcal{M}$ satisfies $\mathcal{C}$.

Furthermore, since the clause set $\mathcal{C}^*$ represents the formula $\forall\, C_1^* \wedge \ldots \wedge C_n^*$, we obtain that $\mathcal{C}^*$ is unsatisfiable iff $\exists \neg C_1^* \vee \ldots \vee \neg C_n^*$ is valid. Let now $\mathcal{C}^+$ be the clause set $\{C_1^+, \ldots, C_n^+\}$ where C_i^+ is obtained from C_i^* by replacing $A(f_p(t_1, \ldots, t_m))$ and $\neg A(f_p(t_1, \ldots, t_m))$ by $f_p(t_1, \ldots, t_m) : A$ and $f_p(t_1, \ldots, t_m) : \neg A$, respectively. Then each C_i^+ represents an $\mathcal{ALC}$- restriction since A is unary predicate, i.e., a concept. Obviously, testing validity of $\exists \neg C_1^* \vee \ldots \vee \neg C_n^*$ is equivalent to testing RQ-validity of $\neg C_1^+ \vee \ldots \vee \neg C_n^+$ w.r.t. the following restricted quantification system over $\mathcal{ALC}$:

- the signature Δ is given by the concept A and the set of function symbols occurring in $\mathcal{C}^+$,
- restrictions are of the form $t : A$ only, where t is a Δ-term,
- the restriction theory is given by an empty ABox and a declaration

$$f : \underbrace{\top \times \ldots \times \top}_{m} \mapsto \top$$

  for each m-ary function symbol in Δ.

Summing up, if we had an algorithm for deciding RQ-validity of a set of $\mathcal{ALC}$-restrictions we could decide RQ-validity of $\mathcal{C}^+$ and thus of the clause set $\mathcal{C}$. □

That means, we cannot decide whether or not a given set of $\mathcal{ALC}$-restrictions is RQ-valid. The reason for this lies in the fact that we allowed function symbols. If we restrict ourselves such that no function symbols occur, neither explicitly in RQ-formulas nor implicitly in restricted existential quantifiers which are eliminated by introducing Skolem function symbols, RQ-validity is known to be decidable (cf. [BBH+90]).

5 An Application: Query Answering

Though the result of the previous subsection shows that one cannot obtain a decidable refutation procedure for concept logics, we will now show that concept logics can be applied, e.g., in query answering or in abductive systems.

Let us firstly have a look at the query answering capabilities of concept logics. If we use the classical resolution principle to test unsatisfiability of a clause set, we obtain answers **yes** or **no** (provided that the algorithm terminates at all). For example, let us apply classical resolution to the following problem: In a knowledge base it is explicitly represented that the supermarket is open each day except from sunday, i.e., the clause set

$$\begin{array}{c} \textit{day (monday)} \\ \vdots \\ \textit{day (sunday)} \\ \textit{supermarket-open (monday)} \\ \vdots \\ \textit{supermarket-open (saturday)} \\ \neg\textit{supermarket-open (sunday)} \end{array}$$

is stored. Obviously, the query Q_1, "is the supermarket open on wednesday?", can be answered with **yes**, what is intuitively adequate, by a single resolution step. But, on the other hand, the query Q_2, "is the supermarket closed some day?", will be answered with **yes** as well. Though this answer is logically correct, it is an unsatisfying answer if one wonders whether to go to the supermarket on saturday or on sunday.

There exist extensions of the classical resolution principle which can answer the query Q_2 in such a way that "the supermarket is closed on sunday" can be generated from this answer. One example is the use of PROLOG (e.g., [Llo84]) where the variable bindings, which have been made to generate a refutation, are stored explicitly. Thus, queries containing unbound variables are not only answered by **yes** or **no** but, additionally, by an appropriate variable binding of their free variables. As a disadvantage of this approach one might consider the use of **negation as failure**. That means, if PROLOG fails to prove a fact p, it considers $\neg p$ as proved. For example, if it was not stored in the above database whether or not the supermarket is open on thursday, PROLOG would give two answers to query Q_2, namely thursday and sunday. Another approach has been presented in Section 4.7 of [GN87]. There, a special answer literal $Ans\,(\nu_1, \ldots, \nu_n)$ is introduced where $\nu_1, \ldots, \nu_n$ are the free variables of the query which are bounded to values during the refutation process. Unfortunately, the problem whether or not one has found all possible answers of the query is undecidable.

Now, how can we use concept logics for query answering and what advantages does this approach have? Firstly, we can assume a knowledge base to be given by a set of

RQ-formulas, and a restriction theory by an ABox A and a set F of declarations. A query, given as RQ-formula, can then be answered by constrained resolution as follows. The RQ-formulas and the negated query are translated into a set $\mathcal{C}$ of constrained clauses. Then we start deriving empty RQ-clauses $\square \,||\, R$ from $\mathcal{C}$, where each of these empty RQ-clauses tells us that the query is a logical consequence from the knowledge base whenever R is satisfied. In the above supermarket example we can represent some part of the knowledge in an ABox A, e.g.,

$$\begin{array}{c} \mathit{monday} : \mathit{Day} \\ \vdots \\ \mathit{sunday} : \mathit{Day} \end{array}$$

where *Day* is a concept.[9] Which days the supermarket is open can be stored as follows (already translated into a set of RQ-formulas)

$$\begin{array}{c} \mathit{supermarket\text{-}open}\ (x_1) \,||\, x_1 = \mathit{monday} \\ \vdots \\ \mathit{supermarket\text{-}open}\ (x_6) \,||\, x_6 = \mathit{saturday} \\ \neg \mathit{supermarket\text{-}open}\ (x_7) \,||\, x_7 = \mathit{sunday}. \end{array}$$

The negated query Q_2 can be represented by the RQ-clause $\mathit{supermarket\text{-}open}\ (x) \,||\, x : \mathit{Day}$, and by a single RQ-resolution step we obtain the empty RQ-clause

$$\square \,||\, x : \mathit{Day} \wedge x = \mathit{sunday}.$$

From this empty RQ-clause the constructive answer "the supermarket is closed on sunday" can be obtained immediately.

The approach of using concept logics for query answering has two advantages. Firstly, logical negation is used instead of negation as failure. That means, even if none of the facts "the supermarket is open on thursday" nor "the supermarket is closed on thursday" would be represented, RQ-resolution only gives a single answer to query Q_2, namely sunday. Secondly, a part of the knowledge base can be represented in an ABox such that one can reason on this part of the knowledge base by using specialized algorithms (e.g., [Hol90]).

Let us now reconsider the above generated empty RQ-clause $\square \,||\, x : \mathit{Day} \wedge x = \mathit{sunday}$. This empty RQ-clause tells us that the query is answered in each model of the restriction theory which satisfies $\mathit{sunday} : \mathit{Day}$. As shown in subsection 4.4, the problem whether the restrictions of a set of empty RQ-clauses are RQ-valid is undecidable (provided there are function symbols in some part of the complete knowledge base). That means, if empty RQ-clauses $\square \,||\, R_1, \ldots, \square \,||\, R_n$ are derived from a clause set $\mathcal{C}$, we can in general not decide whether these empty RQ-clauses are sufficient to prove

[9] For sake of readability we omit function declarations here (e.g., $\mathit{monday} :\mapsto \top, \ldots$).

RQ-unsatisfiability of $\mathcal{C}$. However, given the above empty RQ-clause it is easy to verify that the restriction $Day \wedge x = sunday$ is satisfied by each model of the restriction theory since $sunday : Day$ is contained in the ABox A.

But even if we cannot decide whether a given set of empty RQ-clauses represents a contradiction in all or only in some models of the restriction theory, there are interesting applications of this kind of query answering. We will now show how to use concept logics in abductive reasoning components. As an example, suppose the following part of a (colloquial language) knowledge base to be given

(A) If there is high water at place x, then x is wet
(B) If it is raining at place x, then x is wet.

Furthermore, assume we have observed that the 4th Avenue is wet and are interested in a possible explanation for this fact. In order to solve this (abductive) reasoning task we can represent (A), (B), and the negated observation by one RQ-clause each. Thus, using an appropriate restriction theory we may obtain the following constrained RQ-clause set

$$(1) \quad wet\ (x_1) \,||\, x_1 : highwater$$
$$(2) \quad wet\ (x_2) \,||\, x_2 : raining$$
$$(3) \quad \neg\ wet\ (x) \,||\, x = 4th\ Avenue.$$

By applying one constrained resolution step to (1) and (2) we obtain the empty RQ-clause

$$(4) \quad \square \,||\, x : highwater \wedge x = 4th\ Avenue.$$

This empty RQ-clause solves our reasoning problem by giving a possible explanation, namely, there is high water at the 4th Avenue. This is due to the fact that the RQ-clause set $\{(1), (2), (3)\}$ is RQ-unsatisfiable in all models of the restriction theory which satisfy $4th\ Avenue : highwater$ and can be seen as an abductive reasoning step (cf., e.g., [BN92]).

6 Conclusion

In this paper we presented an instantiation of the general refutation procedure given in [BHL93] for restricted quantification systems over the concept language $\mathcal{ALC}$. We showed that such an RQS satisfies condition (TM) and thus, as an optimization, $\mathcal{ALC}$-restrictions can be tested on constraint unifiability instead of RQ-satisfiability (cf. [BHL93]), and an algorithm for this constraint unifiability test has been given. In contrast to concept logics without function symbols [BBH+90] it turned out that RQ-validity of $\mathcal{ALC}$-restrictions becomes undecidable when allowing function symbols in these restrictions. Thus, as a noteworthy result, describing problems in terms of RQ-clauses without function symbols or in terms of RQ-formulas is more than syntactical

sugar. The reason for this is due to the fact that Skolemization may introduce Skolem function symbols into restrictions. We proved that allowing function symbols in $\mathcal{ALC}$-restrictions together with disjunction (which is needed to test a set of restrictions on RQ-validity) is as expressive as ordinary clause logic which is known to be undecidable.

However, it turned out that there are interesting applications of concept logics with function symbols. Firstly, we presented a query answering approach based on concept logics. For giving (partial) answers to a query we only need to test constraint unifiability of $\mathcal{ALC}$-restrictions, whereby each empty RQ-clause $\square \,||\, R$ with a constraint unifiable restriction R represents an answer in all models of the restriction theory which satisfy R. Testing validity of a set of restrictions is only needed if we want to decide whether the derived empty RQ-clauses give an exhaustive answer to the query. Secondly, it turned out that concept logics with function symbols can be used within abductive reasoning systems in order to generate possible explanations for an observation.

References

[BBH+90] F. Baader, H.-J. Bürckert, B. Hollunder, W. Nutt, and J. H. Siekmann. Concept logics. In Lloyd, editor, *Proceedings of Symposium on Computational Logic*, ESPRIT Basic Research Series, pages 177–201. Springer, 1990.

[BHL93] H.-J. Bürckert, B. Hollunder, and A. Laux. On skolemization in logics with constraints. Research Report RR-93-06, DFKI Saarbrücken, 1993.

[BN92] H.-J. Bürckert and W. Nutt. On abduction and answer generation through constrained resolution. Research Report RR-92-51, DFKI Saarbrücken, 1992.

[Bür91] H.-J. Bürckert. *A Resolution Principle for a Logic with Restricted Quantifiers*, volume 568 of *LNAI*. Springer, 1991.

[Bür93] H.-J. Bürckert. A Resolution Principle for Constrained Logics. *Artificial Intelligence*, 1993. To appear.

[Coh92] A.G. Cohn. A many sorted logic with possibly empty sorts. In *Proceedings of the 11th Conference on Automated Deduction*, volume 607 of *LNAI*, pages 633–647. Springer, 1992.

[GN87] M. R. Genesereth and N. J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufman Publishers, Inc., 1987.

[Hol90] B. Hollunder. Hybrid inferences in KL-ONE-based knowledge representation systems. In *14th German Workshop on Artificial Intelligence*, volume 251 of *Informatik-Fachberichte*, pages 38–47, Ebingerfeld, Germany, 1990. Springer.

[HS88] M. Höhfeld and G. Smolka. Definite relations over constraint languages. LILOG–Report 53, IBM Deutschland, 1988.

[Lau92] A. Laux. Integrating a modal logic of knowledge into terminological logics. Research Report RR-92-56, DFKI Saarbrücken, 1992.

[Llo84] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 1984.

[Obe62] A. Oberschelp. Untersuchungen zur mehrsortigen Quantorenlogik (in german). *Mathematische Annalen*, 145:297–333, 1962.

[Sch89] M. Schmidt-Schauß. *Computational Aspects of an Order-Sorted Logic with Term Declarations*, volume 395 of *LNAI*. Springer, 1989.

[Wal87] C. Walther. *A Many-Sorted Calculus Based on Resolution and Paramodulation.* Research Notes in Artificial Intelligence. Pitman Publishing, London, 1987.

[Wal88] C. Walther. A many-sorted unification. *JACM*, 35(1):1–17, 1988.

[Wei92] C. Weidenbach. A new sorted logic. In *Proceedings of the GWAI-92*, 1992.

[WO90] C. Weidenbach and H.-J. Ohlbach. A resolution calculus with dynamic sort structures and partial functions. In *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 688–693. Pitman Publishing, London, August 1990.

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
D-6750 Kaiserslautern
FRG

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse bezogen werden.
Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far can be ordered from the above address.
The reports are distributed free of charge except if otherwise indicated.

DFKI Research Reports

RR-92-21
Jörg-Peter Mohren, Jürgen Müller
Representing Spatial Relations (Part II) -The Geometrical Approach
25 pages

RR-92-22
Jörg Würtz: Unifying Cycles
24 pages

RR-92-23
Gert Smolka, Ralf Treinen:
Records for Logic Programming
38 pages

RR-92-24
Gabriele Schmidt: Knowledge Acquisition from Text in a Complex Domain
20 pages

RR-92-25
Franz Schmalhofer, Ralf Bergmann, Otto Kühn, Gabriele Schmidt: Using integrated knowledge acquisition to prepare sophisticated expert plans for their re-use in novel situations
12 pages

RR-92-26
Franz Schmalhofer, Thomas Reinartz, Bidjan Tschaitschian: Intelligent documentation as a catalyst for developing cooperative knowledge-based systems
16 pages

RR-92-27
Franz Schmalhofer, Jörg Thoben: The model-based construction of a case-oriented expert system
18 pages

RR-92-29
Zhaohui Wu, Ansgar Bernardi, Christoph Klauck:
Skeletel Plans Reuse: A Restricted Conceptual Graph Classification Approach
13 pages

RR-92-30
Rolf Backofen, Gert Smolka:
A Complete and Recursive Feature Theory
32 pages

RR-92-31
Wolfgang Wahlster:
Automatic Design of Multimodal Presentations
17 pages

RR-92-33
Franz Baader: Unification Theory
22 pages

RR-92-34
Philipp Hanschke: Terminological Reasoning and Partial Inductive Definitions
23 pages

RR-92-35
Manfred Meyer:
Using Hierarchical Constraint Satisfaction for Lathe-Tool Selection in a CIM Environment
18 pages

RR-92-36
Franz Baader, Philipp Hanschke:
Extensions of Concept Languages for a Mechanical Engineering Application
15 pages

RR-92-37
Philipp Hanschke: Specifying Role Interaction in Concept Languages
26 pages

RR-92-38
Philipp Hanschke, Manfred Meyer:
An Alternative to H-Subsumption Based on Terminological Reasoning
9 pages

RR-92-40
Philipp Hanschke, Knut Hinkelmann: Combining Terminological and Rule-based Reasoning for Abstraction Processes
17 pages

RR-92-41
Andreas Lux: A Multi-Agent Approach towards Group Scheduling
32 pages

RR-92-42
John Nerbonne:
A Feature-Based Syntax/Semantics Interface
19 pages

RR-92-43
Christoph Klauck, Jakob Mauss: A Heuristic driven Parser for Attributed Node Labeled Graph Grammars and its Application to Feature Recognition in CIM
17 pages

RR-92-44
Thomas Rist, Elisabeth André: Incorporating Graphics Design and Realization into the Multimodal Presentation System WIP
15 pages

RR-92-45
Elisabeth André, Thomas Rist: The Design of Illustrated Documents as a Planning Task
21 pages

RR-92-46
Elisabeth André, Wolfgang Finkler, Winfried Graf, Thomas Rist, Anne Schauder, Wolfgang Wahlster:
WIP: The Automatic Synthesis of Multimodal Presentations
19 pages

RR-92-47
Frank Bomarius: A Multi-Agent Approach towards Modeling Urban Traffic Scenarios
24 pages

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

DFKI Technical Memos

TM-91-12
Klaus Becker, Christoph Klauck, Johannes Schwagereit: FEAT-PATR: Eine Erweiterung des D-PATR zur Feature-Erkennung in CAD/CAM
33 Seiten

TM-91-13
Knut Hinkelmann: Forward Logic Evaluation: Developing a Compiler from a Partially Evaluated Meta Interpreter
16 pages

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

DFKI Documents

D-92-11
Kerstin Becker: Möglichkeiten der Wissensmodellierung für technische Diagnose-Expertensysteme
92 Seiten

D-92-12
Otto Kühn, Franz Schmalhofer, Gabriele Schmidt: Integrated Knowledge Acquisition for Lathe Production Planning: a Picture Gallery (Integrierte Wissensakquisition zur Fertigungsplanung für Drehteile: eine Bildergalerie)
27 pages

D-92-13
Holger Peine: An Investigation of the Applicability of Terminological Reasoning to Application-Independent Software-Analysis
55 pages

D-92-14
Johannes Schwagereit: Integration von Graph-Grammatiken und Taxonomien zur Repräsentation von Features in CIM
98 Seiten

D-92-15
DFKI Wissenschaftlich-Technischer Jahresbericht 1991
130 Seiten

D-92-16
Judith Engelkamp (Hrsg.): Verzeichnis von Softwarekomponenten für natürlichsprachliche Systeme
189 Seiten

D-92-17
Elisabeth André, Robin Cohen, Winfried Graf, Bob Kass, Cécile Paris, Wolfgang Wahlster (Eds.): UM92: Third International Workshop on User Modeling, Proceedings
254 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-92-18
Klaus Becker: Verfahren der automatisierten Diagnose technischer Systeme
109 Seiten

D-92-19
Stefan Dittrich, Rainer Hoch: Automatische, Deskriptor-basierte 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-artigen Syntax von RELFUN
51 Seiten

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

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

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

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

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

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

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

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

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

D-93-06
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-$).

Concept Logics with Function Symbols

Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux

RR-93-07
Research Report