**Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH**

**Research
Report**
RR-92-51

On Abduction and Answer Generation through Constrained Resolution

Hans-Jürgen Bürckert, Werner Nutt

October 1992

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 by the shareholder companies Daimler-Benz, IBM, Insiders, Fraunhofer Gesellschaft, GMD, Atlas Elektronik, Digital-Kienzle, Philips, Sema Group Systems, 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
- Intelligent Communication Networks
- Intelligent Cooperative Systems.

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

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

Prof. Dr. Gerhard Barth
Director

On Abduction and Answer Generation through Constrained Resolution

Hans-Jürgen Bürckert, Werner Nutt

DFKI-RR-92-51

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

On Abduction and Answer Generation through Constrained Resolution

Hans-Jürgen Bürckert and Werner Nutt

DFKI, Stuhlsatzenhausweg 3, 6600 Saarbrücken, Germany
e-mail: {hjb, nutt}@dfki.uni-sb.de

Abstract: Recently, extensions of constrained logic programming and constrained resolution for theorem proving have been introduced, that consider constraints, which are interpreted under an open world assumption. We discuss relationships between applications of these approaches for query answering in knowledge base systems on the one hand and abduction-based hypothetical reasoning on the other hand. We show both that constrained resolution can be used as an operationalization of (some limited form of) abduction and that abduction is the logical status of an answer generation process through constrained resolution, ie., it is an abductive but not a deductive form of reasoning.

Keywords: Constrained resolution, query answering, intensional answers, abduction.

Contents

1 Introduction 2

2 Constrained Resolution..... 4

3 Query Answering..... 9

4 Abductive Framework 15

5 Concluding Remarks 17

References.....18

1 Introduction

Quite recently a extension of logic programming and theorem proving by socalled open constraints that is of great interest for knowledge representation applications has been developped. Open constraints are symbolic constraints, which are no longer interpreted with respect to a single domain (ie., under closed world semantics) but over classes of models. In this paper we will show how these approaches are related to other techniques known from knowledge representation, logic programming, and deductive databases.

In particular, we will discuss some relationships between constrained resolution for theorem proving (Bürckert 1990, 1991, Frisch & Scherl 1990), query answering in constraint-based extensions of logic programming (Jaffar & Lassez 1987, van Hentenryck 1989, Höhfeld & Smolka 1988, Smolka 1989, Ait-Kaci & Podelski 1991), extensions of deductive database systems with intensional answers (Cholvy & Demolombe 1986, Imielsky 1987, Motro & Yuan 1990, Chu et al. 1991), and abduction-based hypothetical reasoning (Pople 1973, Poole 1988, Eshghi & Kowalski 1989, Konolige 1990, O'Rourke 1990, Denecker & De Schreye 1992, Kakas et al. 1992, Merziger 1992).

Hypothetical reasoning is the process of finding hypotheses that explain observed facts with respect to some knowledge base. Abductive inferencing is a way to treat this within a logical framework, where abduction is the following rule of inference: if we observe B and we know that A always implies B then we may infer that A holds. Clearly that this kind of inference is not sound, since logically "A implies B" means "not A or B". The latter is in particular true, if A is false, and hence inferring A may be incorrect.

The proper idea of hypothetical reasoning lies in finding possible *causes* for a given observation, which seems not to be adequately modelled by logical implication. Thus it would be more adequate to use only "relevant" implications as a source for abductive reasoning: If we have B and we know "A implies B" we infer A, only if A is not generally false. This means, we abduce only explanations that are *consistent* with our knowledge.

One application for abductive reasoning as pointed out by (Poole 1988) lies in default reasoning (Reiter 1980): Default reasoning is generating possibly unsound, but

justified inferences, where the justification is given by any plausible argumentation, eg., a default rule. Poole argues that default reasoning, for instance by the well-known default rule "all birds fly", is from the logical point of view much better modelled by abduction: Given a distinguished bird, say Tweety, the default inference that Tweety flies (as long as there is no evidence that it doesn't) may be seen as *abducing* a justification "bird-so-fly" for the inference "tweety flies".

A different, but – as we will see in a minute – similar view is obtained when we use McCarthy's approach of circumscribing "abnormality" predicates (McCarthy 1980): Default rules like "all birds fly" are coded into a logical implication "if something is a bird and if it is not an abnormal one then it flies" or in predicate logic notation

$$\forall x\ bird(x) \wedge \neg abnormal(x) \Rightarrow fly(x).$$

Circumscription (ie., minimization of the extension of the *abnormal*-predicate) is used in order to obtain for a distinguished bird the default inference that it flies: The model theory is restricted to those models where the set of abnormal birds is as small as possible, ie., where given birds are normal birds and hence fly, as long as there are no reasons to assume the contrary.

Now, taking a more "positive" approach, ie., using "normality" predicates instead of abnormality, we can rewrite the formula above into

$$\forall x\ bird(x) \wedge normal(x) \Rightarrow fly(x).$$

If we distinguish these normality predicates from all other predicates in that we allow application of such implication rules only if their normality precondition is consistent (eg., with some underlying theory of normality), we get the same kind of inference as above: We can infer that Tweety flies provided it is justified (ie., it is consistent to assume) that Tweety is a normal bird.

With this view of hypothetical reasoning we can compare it with recent work, that generalizes logic programming or theorem proving by integrating "open" constraints. In these approaches constraints are interpreted under an open world assumption, ie., we have more than one model for the constraint theory (Höhfeld & Smolka 1988, Bürckert 1990, 1991, Frisch & Scherl 1990, Ait-Kaci & Podelski 1991).[1] Our normality predicates can be considered as open constraints that have to be interpreted with respect to some constraint theory of "normality": As we will see, a formula

[1]This is in contrast to CLP-scheme of Jaffar and Lassez, where we have "closed" constraints that are interpreted over a fixed single model, the domain of computation (Jaffar & Lassez 1987, Maher 1987).

constrained´, eg., by normality predicates can only be used in the inference process, if its constraint is consistent with that constraint theory.

Constraint logic programming approaches use a constraint based version of resolution as an answer generation procedure that transforms any (constrained) query to a constrained logic program (ie., a set of constrained definite clauses) into a consistent answer constraint (Jaffar & Lassez 1987, Höhfeld & Smolka 1988, Smolka 1989). Such a constraint answers the query in that it logically entails the query. Recall that in classical Horn clause programming this is exactly the status of an answer substitution. In addition, finding an answer substitutions proves the existential closure of the query to be a logical consequence of the program. However, as we will see, the second property no longer holds in the case of general constrained logic programming: There an answer constraint only proves the query to be true in those models that satisfy the answer.

If we compare this with abductive reasoning, we can state that answer generation is an abductive process: It generates consistent assumptions, that together with the given program entail the query (as an existential statement), but, in general, it does not prove the query to be a consequence of the program.

So our aim in this paper is to demontrate both that constrained resolution (section 2) may be used as an operationalization of some form of abduction (section 4) and that logically an answer generation process through constrained resolution (section 3) is an abductive but not a deductive form of reasoning. We will conclude with a brief discussion of that abductive view of question answering (section 5).

2 Constrained Resolution

In this section we will recall the notion of constrained resolution (Bürckert 1990, 1991). We assume the reader to be familiar with some logical and model theoretical background as provided by textbooks on mathematical logics (eg., Shoenfield 1967). We also assume some acquaintance with foundations of automated theorem proving and logic programming (eg., Chang & Lee 1973, Kowalski 1979, Lloyd 1984, Gallier 1986).

For constrained resolution we abstract from the classical view of resolution in that we replace the unification process by a constraint solving procedure.[2] Therefore we will consider constrained clauses over a constraint system.

A ***constraint system*** $\mathfrak{R}$ consists of a signature $\Delta_{\mathfrak{R}}$ of predicate and function sysmbols, a set $M_{\mathfrak{R}}$ of structures over this signature, an infinite set V of variables, and a set $C_{\mathfrak{R}}$ of open formulae over the signature. The set $C_{\mathfrak{R}}$ is closed under conjunction and variable substitution. We call $\Delta_{\mathfrak{R}}$ the ***constraint signature***, $M_{\mathfrak{R}}$ the ***constraint theory***, its elements are the ***constraint models***, and the elements of $C_{\mathfrak{R}}$ are the ***constraints***. As a special case, the constraint theory might be specified by a consistent set of closed formulae over $\Delta_{\mathfrak{R}}$, called ***constraint declarations***, such that $M_{\mathfrak{R}}$ is just the set of all models of this axiomatization.

A constraint Γ is called ***solvable*** or ***satisfiable*** iff its existential closure $\exists \Gamma$ is satisfied by some constraint model $\mathfrak{I}$. The pair $(\mathfrak{I}, \alpha)$ of a constraint model $\mathfrak{I}$ and a variable assignment $\alpha: V \to \mathfrak{I}$ satisfying the constraint Γ in $\mathfrak{I}$ is called a ***solution*** of Γ, written $(\mathfrak{I}, \alpha) \models \Gamma$. In that case the assignment α is also called an ***$\mathfrak{I}$-solution*** of the constraint Γ, and the constraint is called $\mathfrak{I}$-solvable. The set of $\mathfrak{I}$-solutions α of Γ is denoted by $SOL(\mathfrak{I}, \Gamma)$ and the set of all solutions $(\mathfrak{I}, \alpha)$ of Γ is denoted by $SOL(\Gamma)$.

Now, given a constraint system and a set Σ of predicate symbols (disjoint from $\Delta_{\mathfrak{R}}$), a ***constrained formula*** is a pair $F \parallel \Gamma$, where F is any formula over Σ and Γ is any constraint. Both F and Γ may be open formulae and they may have common free variables; all free variables will be treated as implicitly universally quantified. If necessary we therefore will sometimes write the above constrained formula as $\forall_{X: \Gamma(X)} F$, where X is the set (or sequence) of the free variables of Γ and F. The free variables of Γ are also called the constrained variables of the constrained formula $F \parallel \Gamma$. Except for ease of presentation in some examples we we will not make use of formulae with constrained existential quantification (ie., formulae of the form $\exists_{X: \Gamma(X)} F$) as in (Bürckert 1991).

A ***constrained clause*** is a pair $C \parallel \Gamma$, where C is a (possibly empty) finite set of literals (atoms and negated atoms) over Σ and Γ is a constraint. We call C the ***kernel*** and Γ the ***constraint*** of the clause. Considered as a formula C is a disjunction of literals. If C contains only one positive literal H and a (possibly empty) set of

---

[2] As shown in (Siekmann 1990, Bürckert 1991, Jouannaud & Kirchner 1991) unification can be considered as solving equations in certain algebras, and hence it can also be seen as a constraint solving method.

negative literals, we have a constrained ***definite clause*** and we also write $(H \Leftarrow B) \parallel \Gamma$[3]. We call H the ***head***, B the ***body*** of the clause. If the body of a constrained definite clause is empty we have a constrained ***fact clause***, otherwise it is a constrained ***rule clause***. If a clause has only negative literals we call it a constrained negative clause or a ***goal clause***; such a clause can be seen as a rule clause with empty head and hence is often written $\Leftarrow B \parallel \Gamma$. If the kernel of a constrained clause is empty we have a constrained ***empty clause*** $\square \parallel \Gamma$.

The model theory of constrained formulae is given by free expansions of the constraint models with the symbols of Σ: An ***$(\mathfrak{R},\Sigma)$-structure*** is defined as a constraint model, where in addition the predicate symbols of Σ are interpreted as relations on the carrier. Notice, that in the case where the constraint theory is given by constraint declarations, the $(\mathfrak{R},\Sigma)$-structures are exactly those $(\Delta_{\mathfrak{R}} \cup \Sigma)$-structures that satisfy the constraint declarations. Atoms, disjunctions, conjunctions, implications, negations, and quantification are interpreted as usual. Constrained clauses are interpreted as disjunctions, their constraints play the role of preconditions and all free variables are universally quantified. This means that a constrained formula is interpreted as universally quantified implication and that constrained formulae with unsolvable constraint are tautologies w.r.t. the constraint theory.

Obviously we have the following special cases: A constrained clause is always satisfiable by those $(\mathfrak{R},\Sigma)$-structures $\mathfrak{I}$, for which the constraint has no $\mathfrak{I}$-solution. This especially applies to empty clauses, ie., constrained empty clauses may be satisfiable.

Example: (1) If we take CLP(R), the CLP-language over the real numbers R, we have a constraint system consisting of the single constraint model R of real numbers, where the constraints are arithmetical equations and inequations. As constrained formulae we have Horn formulae with arithmetic constraints. Solutions are pairs (R, α), where α maps the variables to such real numbers that solve the constraints.

(2) As an example for a constraint system with open constraints one can take a knowledge base in a (decidable) terminological language of the KL-ONE family (Brachman & Schmolze 1985, Nebel 1989, Baader et al. 1990). These are concept description lan-

---

[3] In this notation B is the set $\{A : \neg A$ is one of the negative literals of the definite clause$\}$. Notice, that in the sequel we will mostly drop the parantheses and write $H \Leftarrow B \parallel C$. This is justified, since on the one hand, a constraint formula can be read as an implication, i.e., $(H \Leftarrow B) \Leftarrow C$. On the other hand this douple implication is equivalent to $(H \Leftarrow (B \wedge C)$. The two writings $(H \Leftarrow B) \parallel C$ and $H \Leftarrow (B \parallel C)$ were equivalent, if in the first form we interpreted the bars $\parallel$ as an implication, and in the second form as a conjunction. But notice, that the second one is not really a constrained formula, as we do not allow recursive forms of constrained formulae and, of course, as C might constrain variables of H.

guages which are essentially equivalent to sublanguages of predicate logic. They allow in a socalled TBox the definition of terminology, a concept hierarchy (semantically a subset hierarchy), and in an ABox the assertion of facts about instance relationships between objects and concepts: (atomic) concepts are unary predicates, concept descriptions are certain open formulae built up with concepts and binary relations (roles) by conjunction, disjunction, negation and restricted forms of quantification (in order to bind the second arguments of roles, such that the resulting concept description contains exactly one free variable – ie. semantically they denote sets); objects are constants that can be used to instantiate the open formulae in order to assert membership relationships. The constraint models are all models of the terminological knowledge base. As constraints one can take sets of concept descriptions constraining the variables of our constrained formulae (cf. Baader et al 1991, Bürckert 1991).

For the proof theory of constrained formulae we consider a resolution based refutation calculus. Therefore sets of constrained formulae have to be transformed into constrained clause form, ie., into sets of constrained clauses. This is a non-trivial task, as the constraints may be unsolvable over some of the constraint models, which leads to "empty" quantification (see Bürckert et al. 1992 for more details about skolemization of constrained formulae).

For sets of constrained clauses we then have a resolution and a factoring rule where unification is replaced by a constrained solvability test. Thus, as mentioned in the introduction, constrained clauses are used in the inference process only if their constraints are solvable in the underlying constraint theory.

Constrained resolution rule:[4]

$$\frac{P(x) \vee C_1 \parallel \Gamma_1 \quad \neg P(y) \vee C_2 \parallel \Gamma_2}{C_1 \vee C_2 \parallel \Gamma_1 \wedge \Gamma_2 [x = y]} \qquad \text{if } \Gamma_1 \wedge \Gamma_2 [x = y] \text{ is solvable}$$

Constrained factoring rule (positive case)**:**

$$\frac{P(x) \vee P(y) \vee C \parallel \Gamma}{P(y) \vee C \parallel \Gamma [x = y]} \qquad \text{if } \Gamma [x = y] \text{ is solvable}$$

---

[4] The reading of the rules is as follows: Given the clause schemas above the line, infer the clause schema below the line, provided the condition holds. The focussed literals with n-ary predicate symbol P are considered as having n-ary vectors of variables x or y, while the C's that are separated from them by the $\vee$ are the remaining possibly empty sets of literals. The equation $[x = y]$ in square brackets means that the x's are to be replaced by the corresponding y's simultaneously at every occurrence in the whole clause or constraint, respectively (variable substitution). By the way, this demonstrates why we required the constraints to be closed under conjunction and variable substitution.

Constrained factoring rule (negative case)**:**

$$\frac{\neg P(x) \vee \neg P(y) \vee C \parallel \Gamma}{\neg P(y) \vee C \parallel \Gamma[x = y]} \qquad \text{if } \Gamma[x = y] \text{ is solvable}$$

Given a set of constrained clauses S_0, a derivation is any (possibly infinite) sequence (S_n) of clause sets such that S_{n+1} is obtained from S_n by a resolution or a factoring step that selects variants of clauses of S_n matching the schemes above the lines of our rules and adds the corresponding clauses from below the lines to the clause set. A refutation is any derivation such that for each constraint model $\mathfrak{I} \in M_{\mathfrak{R}}$ there exists a clause set S_n in the derivation which contains an empty clause whose constraint is $\mathfrak{I}$-solvable. Or, to phrase it differently: A refutation is a (possibly infinite) derivation (S_n), such that $\bigcup_n S_n$ contains for every constraint model $\mathfrak{I}$ an empty clause, whose constraint is $\mathfrak{I}$-solvable. From (Bürckert 1990, 1991) we have the following soundness and completeness result for constrained resolution.

Theorem: (Refutation Completeness of Constrained Resolution)

A set of constrained clauses is unsatisfiable w.r.t. a constraint theory iff there is (possibly infinite) refutation starting with that clause set.

By the compactness theorem of first order logics we can simplify this result for constraint theories with first order axiomatization in that a refutation is already given by some finite derivation. In this case we can add a collection rule for the constraints of empty clauses as disjunction of that constraints.

Constraint collection rule:

$$\frac{\square \parallel \Gamma_1 \quad \square \parallel \Gamma_2 \quad \ldots}{\square \parallel \Gamma_1 \vee \Gamma_2 \vee \ldots}$$

With that rule we have the following corollary.

Corollary: *Let the constraint theory $M_{\mathfrak{R}}$ be given by a set of constraint declarations. For every unsatisfiable clause set there exists a* finite *refutation, such that the final clause set contains an empty clause whose disjunctive constraint is $\mathfrak{I}$-solvable for every constraint model $\mathfrak{I} \in M_{\mathfrak{R}}$, ie., it is a logical consequence of the constraint theory.*

The following example shows that completeness indeed requires the derivation of several empty clauses.

Example: Suppose that we have a constraint theory M given by the following two constraint declarations

$$\{\Gamma(a),\ \Gamma(b) \vee \Gamma(c)\}$$

Let us consider the following set of three constrained clauses:

$$\begin{aligned} &(1) && P(x,x) \parallel \Gamma(x) \\ &(2) \neg && P(y,v) \parallel \Gamma(y) \wedge v = b \\ &(3) \neg && P(z,w) \parallel \Gamma(z) \wedge w = c \end{aligned}$$

We can derive two constrained resolvents (we simplified the constraints slightly)

from clauses (1) and (2) $\quad \square \parallel \Gamma(b)$

from clauses (1) and (3) $\quad \square \parallel \Gamma(c)$

Of course, none of the two empty clauses provides a refutation, but with the constrained collection rule we get

$$\square \parallel \Gamma(b) \vee \Gamma(c).$$

Obviously, that disjunction of constraints is a logical consequence of our constraint theory and hence is solvable in every constraint model. Thus constrained resolution provides a case distinction by separating the constraint models with respect to the constraints of empty clauses saying that we have reached a contradiction in every model that solve the constraint of a derived empty clause. The case distinction is complete, when the collection rule has collected enough constraints such that for every model there is at least one constraint that is solved by that model.

3 Query Answering

In the last section we gave resolution rules and a refutation completeness result for showing unsatisfiability of constrained clause sets. However, if we address knowledge representation, logic programming or logic-based information systems, we need a positive approach in the sense that we allow queries to a given knowledge base and we expect answers to our queries (Green 1969, Luckham & Nilsson 1971, Kowalski 1979, Lloyd 1984, Frost 1986, Genesereth & Nilsson 1987). We contrast the different assumptions underlying answer generation in logic programming on the one hand and reasoning with open constraints on the other hand.

Let us call a satisfiable set of constrained clauses together with the constraint theory a ***knowledge base*** (KB). If we have only definite clauses, we also call it a ***constrained logic program*** (CLP), and if we have only definite clauses without constraints we will call it a ***logic program*** (LP).[5] The structures satisfying a KB are called ***models*** of the KB.

If we want to retrieve (implicit) knowledge from a KB, there are the following two important forms of queries (cf. Frost 1986, Genesereth & Nilsson 1987), "yes-or-no" questions (eg., *Is John one of Sarah's parents?*) and "fill-in-the-blank" or "who" questions (eg., *Is there a parent of Susan?* or *Who is Sarah's parent?*). Yes-or-no questions can be answered by standard theorem proving methods: Rewrite the yes-or-no question as a theorem and try to prove either the theorem or its negation. Depending on which of the two proofs succeeds the answer is yes or no, respectively, otherwise there is no answer. The problem of course is the undecidability of first order logics, which leads to non-termination of these processes in the cases where no answer exists. Fill-in-the-blank questions, however, still need additional techniques.

Formally, a fill-in-the-blank question is an open formula, where the free variables play the role of blanks to be filled in by an answer. Logically, we can consider such a query as an existentially closed theorem that has to be proved with respect to the given KB. In order to get answers that fill in the blanks we have to generate through the proof candidates that satisfy the conditions required in the query. What are such candidates?

One view is that the candidates have to be designated objects. That is they should be named within the language of the KB: They have to be constants or, more generally, ground terms of the signature of that KB.[6] Hence such an answer is every substitution of the variables by ground terms, such that the query instantiated by the answer is a logical consequence of the KB, is true in *all* models of the KB. Such *definite* answers, however, need not exist in general, even if the existential closure of the query is a theorem of the KB: Take $P(a) \vee P(b)$ as a KB and take the query $\Leftarrow P(x)$. Then

---

[5] This is not directly compatible with our definition of constrained formulae, where we allowed only predicate symbols for the clause kernels, but no function symbols. However, classical LPs become CLPs, when we unfold the argument terms t_i occurring in a clause by replacing them with new variables x_i and adding as constraints the term equations $x_i = t_i$. These constraints have to be interpreted over a single constraint model, the term algebra or equivalently the ground term algebra, ie., the Herbrand universe (cf. Jaffar & Lassez 1987, Buntine & Bürckert 1989, Bürckert 1991). This is, what we will call a logic program.

[6] Such candidates are often called witnesses (for the existential requirements of teh query).

obviously $\exists x\ P(x)$ is a logical consequence of the KB, but of course there is no ground term that satisfies the query in the required way. On the other hand, it is well-known that for KBs of definite clauses (logic programs), such an answer always exists, if the query is a goal clause (Lloyd 1984).

The other view – eg., for *indefinite* KBs – is that the candidates are objects of models of the KB assigned to the existential variables of the query. This means that answers are pairs consisting of a model of the KB and an assignment that satisfies the query in that model.

The second approach is a basis of CLP-schemes with "open" constraints like in (Höhfeld & Smolka 1988), while the first approach is the usual one for deductive databases and common logic programming (Kowalski 1979, Lloyd 1984, Frost 1986).

SLD-resolution is known to provide a powerful technique to answer fill-in-the-blank questions in the definite case: Given a logic program LP and a query (ie., a goal clause $\Leftarrow Q$) the answers can be computed through SLD-resolution. This is a complete answer generation method in the following sense: For every answer there is an SLD-derivation computing a more general answer. Here an answer is a substitution σ of terms – not necessarily of ground terms as specified above, but representing a whole set of such ground substitutions, its ground instances – for the (free) variables of the query, such that the universal closure of σB is a logical consequence of LP: $LP \models \forall \sigma B$. It is easy to see, that this is equivalent to the formulation that the universal closure of the implication $[\sigma] \Rightarrow B$ is a logical consequence of LP: $LP \models \forall([\sigma] \Rightarrow B)$.[7] If we remind that classical LPs are CLPs, where the constraints are sets of term equations (over a single constraint model, the term algebra), $LP \models \forall([\sigma] \Rightarrow B)$ can equivalently be rewritten as $LP \models B \| [\sigma]$. Computing a more general answer means that through SLD-resolution an answer substitution is generated such that a given answer is an instance, ie., it can be obtained from the computed one by further instantiating the variables. It is well-known that this means that every ground instance of the given answer is also a ground instance of the computed answer. Comparing this with the constraint view this in turn means that over the single constraint model given as the ground term algebra all solutions of the given answer substitutions (considered as an equational constraint) are also solutions of the computed answer substitution.

---

[7] Here $[\sigma]$ is an equational representation of the substitution, ie., the conjunction of the substitution components considered as equations.

In constrained logic programming approaches instead of substitutions solvable constraints have been chosen to play the role of answers: Given a *CLP* we call a goal clause $\Leftarrow B \parallel \Gamma$ a ***query*** to the *CLP*. A *solvable* constraint Δ is called an ***answer*** to this query iff all solutions of that answer constraint satisfy the goal: $CLP \models \forall(\Delta \Rightarrow (B \wedge \Gamma))$.[8]

Such answer constraints are seen as *intensional answers* that represent *extensional answers*, ie., solutions $(\mathfrak{I}, \alpha)$ of the answer constraint, by their common property, the constraint, instead of enumerating all those solutions (cf. Cholvy & Demolombe 1986, Imielsky 1987, Motro & Yuan 1990, Chu et al. 1991). The free variables of $\Leftarrow B \parallel \Gamma$ are the blanks to be filled in. Thus we could verbalize the query as the question:

Are there any objects with property Γ, that satisfy B?

As an answer we expect to receive for every model of our KB a set of objects[9], of which we can be sure that they all have the required property Γ and that they satisfy F.

Thus let us define answers in that sense more precisely. Given a KB over some constraint system $\mathfrak{R}$ we call a constraint Δ an ***answer*** to the query $\Leftarrow B \parallel \Gamma$ iff the following three conditions hold:

(1) $SOL(\Delta) \neq \emptyset$ (ie., $\exists\Delta$ is $M_{\mathfrak{R}}$-satisfiable)

(2) $SOL(\mathfrak{I}, \Delta) \subseteq SOL(\mathfrak{I}, \Gamma)$ for each constraint model $\mathfrak{I}$ (ie., $M_{\mathfrak{R}} \models \forall(\Delta \Rightarrow \Gamma)$)

(3) $KB \models F \parallel \Delta$ (ie., $KB \models \forall_{X:\Delta} B$)

Thus according to the query above answers can be verbalized as: *All objects with property Δ – such objects exist* (1)*– satisfy B* (3) *(and they have the property Γ* (2)*).*

By the above corollary to the completeness theorem constrained resolution provides a terminating process (for a first order KB) if we know that there is such an answer for every constraint model. However as for common logic programming, we would like to have a procedure that is able to generate the answers. (Höhfeld & Smolka 1988) provides such an answer generation procedure for constraint logic programs, which is based on constrained SLD-resolution. Notice therefore, that standard SLD-

---

[8] Again, this notation is not directly compatible with our constrained formulae, but it shows the analogy to the notion of an answer in the case of an LP. In order to come to a notation that is more directly related to constraints, an easy transformation of the formulae shows, that $CLP \models \forall(\Delta \Rightarrow (B \wedge \Gamma))$ is equivalent to the requirement that for each constraint model $\mathfrak{I}$ all $\mathfrak{I}$-solutions of the answer constraint Δ are also $\mathfrak{I}$-solutions of the query's constraint Γ and that the constraint formula $B \parallel \Delta$ (remember that B is a conjunction of atoms) is a logical consequence of *CLP*, ie. $CLP \models B \parallel \Delta$, or rewritten with constrained quantification: $CLP \models \forall_{X:\Delta} B$.

[9] It should be guaranteed that a non-empty set of such objects exists in at least one of the models.

resolution can be seen as a goal reduction process starting with a CLP, whose constraints are term equations, and a query. In every step the goal clause is transformed into new goal clause using a suitable clause of the CLP. The process terminates when an empty goal is reached. During this process the equational constraints are collected and transformed into their unifiers. The resulting final unifier is the computed answer.

For arbitrary constraints the only necessary modification is that we collect the constraints (via conjunction) and perhaps transform them into equivalent, simplified forms. The computed answer is then the (simplified) constraint of the final empty goal.

Constrained SLD-resolution rule:

$$\frac{\Leftarrow P(y),\ B_2 \parallel \Gamma_2 \qquad\qquad P(x) \Leftarrow B_1 \parallel \Gamma_1}{\Leftarrow B_1,\ B_2 \parallel \Gamma_1 \wedge \Gamma_2\ [x = y]} \quad \text{if } \Gamma_1 \wedge \Gamma_2\ [x = y] \text{ is solvable}$$

A constrained SLD-derivation is a sequence of constrained SLD-resolution steps starting with some goal clause, such that in every step the resolvent of the direct predecessor step is used as one parent and a variant of a suitable program clause as the other parent. This means that the derivation can be seen as a goal transformation process that transforms a goal clause into the resolvent goal clause with program clauses. An SLD-refutation of a query is a finite SLD-derivation starting with the query and terminating with a constrained empty clause. That means that a refutation reduces the query to a constrained empty clause, whose constraint is an answer to the query.

As we see in the example below we may, however, need several empty goals, such that their constraints together provide a more general answer for any given answer, in the sense that every solution of the given answer is also a solution of at least one of the computed answers. Thus the following theorem of strong completeness or answer completeness of constrained SLD-resolution says essentially that for every given answer there exists a (possibly infinite) number of answers that can be computed by SLD-resolution, which are more general as the given answer in that they cover all solutions of the given one. The theorem is due to (Maher 1987) and has been worked out and generalized in (Höhfeld & Smolka 1988). We give a slightly modified reformulation of their results.

Theorem: (Answer Completeness of Constrained SLD-resolution)

a) *For each solution* $(\mathfrak{I}, \alpha)$ *of a given answer there is an SLD-refutation of the query, such that* $(\mathfrak{I}, \alpha)$ *is an* $\mathfrak{I}$*-solution of the computed answer constraint*

b) *If the constraint theory is first order, then there are finitely many SLD-refutations, such that every solution of the given answer is a solution of some of the computed answers.*

If we recall the definition of an answer, we see that the above theorem says, that given any answer Δ to a query $\Leftarrow B \parallel \Gamma$ constrained SLD-resolution computes answers Λ_i $(1 \leq i \leq n)$ such that

(1) $SOL(\Lambda_i) \neq \emptyset$ $(1 \leq i \leq n)$

(2) $\bigcup_{1 \leq i \leq n} SOL(\mathfrak{I}, \Lambda_i) \subseteq SOL(\mathfrak{I}, \Gamma)$ for each constraint model $\mathfrak{I}$

(3) $CLP \models B \parallel \Lambda_i$ $(1 \leq i \leq n)$

(4) $SOL(\mathfrak{I}, \Delta) \subseteq \bigcup_{1 \leq i \leq n} SOL(\mathfrak{I}, \Lambda_i)$ for each constraint model $\mathfrak{I}$

If we consider examples for such CLPs we see, that there is a problem with this form of query answering. In contrast to classical LP or to CLP with constraints over a single model (what we called "closed" constraints earlier) when we get an answer constraint, this need not prove the existentially closed query to be a logical consequence. In fact the refutation completeness result for constrained resolution shows that we need the derivation of "enough" answers (ie., empty clauses with constraints), in order to have a proof: The refutation completeness theorem says that for each constraint model we have to derive an answer constraint that has solutions in that model. In contrast to this the answer completeness result does not say anything about whether the query is a consequence of the KB or not: The answer completeness theorem works also for queries that are not theorems of the KB, but have answers in some models. [10]

Example: Let us again take the constraint theory of the example in the last section given by the two constraint declarations (a possible reading is given in parantheses):

$\Gamma(a)$	("*a* got a position")
$\Gamma(b) \vee \Gamma(c)$	("either *b* or *c* got a position")

[10] However, the cases the answer completeness theorem has been proved for are constraint theories with exactly one model (Jaffar & Lassez 1987, Maher 1987) and hence it follows trivially from the theorem that here the computation of an answer provides a proof of the query.

Let us then consider a CLP over this constraint theory given by the single constrained fact clause

$$P(x, x) \parallel \Gamma(x) \qquad \text{(“everybody with a position promotes himself”)}$$

and let us have the following query

$$\Leftarrow P(y,z) \parallel \Gamma(y) \wedge z = b \qquad \text{(“does somebody with a position promote } b\text{?”)}$$

The existential closure of this query would be true, if b got the position, since then he promotes himself. Exactly this will also be the answer $\Gamma(x) \wedge x = b$ that will be derived by constrained SLD-resolution.[11] However, from our KB we do not know, whether b or c is the one who got a position, and hence the existential closure of the query (ie. the proposition “somebody with a position promotes b”) does not follow from the KB. That this may cause problems becomes more apparent, when we rephrase the query as a who question: “who with a position promotes b?” Now, an answer “b provided b got the position” might not be extremely useful.

Thus our answer generation process might produce answers, although the given set of constrained clauses is not unsatisfiable (over the constraint theory). Instead the generated answer is a condition, under which the query would become true. Hence, answer generation through constrained resolution is a form of hypothetical reasoning. It is not a proof or refutation procedure, and as we see with the example it cannot be considered as a sound deduction process for existential sentences. Thus the question is, what is the logical status of this form of answer generation?

4 Abductive Framework

In order to find an answer to the question closing the last section, we will consider abductive frameworks for first order logics. Abductive frameworks have been introduced for example in (Poole 1988) or (Eshghi & Kowalski 1989) as a way of formalizing hypothetical reasoning within first order logics and they have been generalized to non-classical logics in (Konolige 1990).

---

[11] Notice, that if we call a set of answers *complete* iff it covers the solutions of any possible answer, it is easy to verify that the set consisting just of the answer of our example forms a complete set for the query.

An ***abductive framework*** is given by a first order signature, a consistent set of formulae over this signature (the KB) and a set of open formulae of the signature (the abducibles). Now, given any closed formula Q, called ***observation***, we say that a ground instance (or a set of ground instances) C of the abducibles is an ***explanation*** for Q iff $KB \cup C \models Q$. In order to avoid trivial explanations, one usually requires in addition that the explanation has to be consistent with the KB (and sometimes also with additional integrity constraints, Eshghi & Kowalski 1989). In (Poole 1989) there is a short discussion of explanations with free variables and a generalization is mentioned that is of interest for us: Here an explanation can be any instance of the abducibles, where free variables have to be existentially closed.

If we compare this with query answering as introduced in the last section we can see a very close connection. The reason is that by the deduction theorem of first order logics C is an explanation for Q iff $KB \models C \Rightarrow Q$.[12] The difference that remains to query answering lies in the quantifiers, since we had there that C is an answer to Q[13] iff $KB \models \forall(C \Rightarrow Q)$.

However, when we consider examples of existentially quantified observations, we see that there is something wrong with Poole's definition. Suppose we observe that *some antelope is running away*:

$$\exists_{x:antelope(x)} \; run\text{-}away(x).$$

Assume further that we know (i.e. we have a KB saying) that *an antelope runs away, if it notices a lion*:

$$\forall_{x:antelope(x)} \; \forall_{y:lion(y)} \; notice(x, y) \Rightarrow run\text{-}away(x).$$

---

[12] By the way there is in fact a very strong connection between abduction and deduction at least in the case of first order logics, which follows with the deduction theorem: If we remember that $C \Rightarrow Q$ is equivalent to $\neg Q \Rightarrow \neg C$ and if we again apply the deduction theorem, we obtain that C is an explanation for Q iff $KB \cup \neg Q \models \neg C$. With this view abduction can be reduced to deduction: In order to *abduce* C from the KB and the observation Q one can equivalently *deduce* $\neg C$ from the KB and the negated observation $\neg Q$. This is one of the main reasons why abduction works that well in the first order case and of course this explains also the operationalization problems with abduction for more expressive logics, where the deduction theorem is no longer valid (eg. in epistemic logics based on modal logic approaches). The other problems with abduction that lie in the requirements that the abduced explanations must be consistent with the KB and the search for good or even best explanations (eg. in the sense of *minimal* explanations) are also not specific to abduction. They occur in a similar manner for deduction, if we use it for deriving new theorems. Here we also want to have *interesting* sentences to be derived, or sentences that are as general as possible.

[13] Now C and Q are open formulae, while for abduction Q was a closed formula and C is ground or existentially closed.

Then one would not be extremely happy with the explanation that Poole's form of abduction would result in, namely that *some antelope notices a lion*:

$$\exists x{:}antelope(x)\ \exists y{:}lion(y)\ notice(x, y).$$

Instead, we would expect that the explanation links the antelope that we observed running away with the one that noticed the lion. Now, this link will be established when explanations for existentially quantified observations are defined in the same way as we did for answers to existential queries.

Thus an adaption of Poole's approach for explanations with free variables as follows will be necessary: For observations Q with free variables that are to be read as existentially quantified we require that in an explanation C the same free variables occur and that the universally closed implication $\forall(C \Rightarrow Q)$ is a consequence of the KB.

With that modification of the notion of an explanation in an abductive framework we see that query answering through constrained resolution is in fact hypotheses generation through abduction: Taking constraints as abducibles and the constraint theory with the constrained clauses as KB, we have an abductive framework, where query answering through constrained resolution generates explanations, the answer constraints, for existentially quantified observations, the existential closures of queries.

5 Concluding Remarks

Of course there is another very close connection between query answering and hypothetical reasoning completely apart from what we discussed before: Finding explanations is by its definition a form of query answering, namely answering why-questions. However, as we have seen in the last section there is also that more technical relationship between abduction of explanations to observations and generating answers to queries by a constrained resolution approach. The question remains now, whether this is really the same?

A closer analysis shows, however, that there remains an important difference. If we want to use constrained resolution for generating explanations we only can collect direct causes as explanations. What constrained resolution cannot provide is explaining chains of causes and effects like if $A \Rightarrow B$ and $B \Rightarrow C$ and we observe C then A is an explanation for C. The reasons lie in the syntactical restrictions that we put on

constrained formulae. Constraints are preconditions, hence they are candidates for explanations. But they are syntactically restricted to be only preconditions, hence they cannot model chains of causes and effects.

If we want to apply theorem proving or logic programming techniques for knowledge retrieval in KBs, we are confronted with the problem that for general KBs it is to strong to allow only designators – ie., term substitutions – as answers to fill-in-the-blank questions. The reason is that for indefinite KBs – differently as for LPs (Horn clause KBs) – there need not exist designators for the witnesses of an existential query (cf. Gallier & Raatz 1985). A solution is to take intensional answers, which, however, need to be restricted, as otherwise any formula implying the query can be taken as an answer. Open constraint theories and (simplified) constraints – eg., terminological languages – could serve as such a restricted *answer language* as we have seen.

However, it is not yet clear how query answering in general KBs should be formalized. And, as we have seen, open constraints provide only abductive answers. The question remains whether it is an adequate definition of an answer, when it comes out that an answer to a fill-in-the-blank question is only a hypothesis under which the question will be true. To see this let us reconsider the antelope example again. When we read the observation as a question: *Is there any antelope running away?* (or as a who question: *Which antelope is running away?*) then it might not be an adequate answer to say: *Yes, there is a running antelop, but only if there exists a lion and that antelope has noticed it* (or in case of the who question: *the antelope that has noticed the lion, if there were one*.)

In order to get complete answers that guarantee that the query is logically entailed by the KB one needs to derive *enough* answers, that provide a complete case distinction. However, this need no longer be decidable, even for constraint systems, where solvability of constraints is decidable (cf. Baader et al. 1992). Thus, for knowledge representtaion applications we need still more investigations providing adequate and operationalizable constraint systems.

References

Ait-Kaci, H., Podelski, A.: *Towards a meaning of LIFE*. Proc. of Int. Symp. on Programming Language Implementation and Logic Programming, Springer LNCS 528, 255-274, 1991.

Baader, F., Bürckert, H.-J., Hollunder, B., Nutt, W., Siekmann, J.H.: *Concept Logics*. In: Computational Logic (Ed. J.W. Lloyd), Springer Basic Research Series, 177-202, 1990.

Baader, F., Bürckert, H.-J., Nebel, B., Nutt, W., Smolka, G.: *On the Expressivity of Feature Logics with Negation, Functional Uncertainty, and Sort Equations*. To appear in *J. of Logic, Language, and Information*, 1992.

Buntine, W., Bürckert, H.-J.: *Solving Equations and Disequations*. SEKI-Report SR-89-03, Universität Kaiserslautern, 1989.

Bürckert, H.-J.: *A Resolution Principle for Clauses with Constraints*. Proc. of Int. Conf. on Automated deduction, Springer LNAI 449, 178-192, 1990.

Bürckert, H.-J.: *A Resolution Principle for a Logic with Restricted Quantifiers*. Springer LNAI 568, 1991.

Bürckert, H.-J., Hollunder, B., Laux, A.: *A Refutation Procedure for Concept Logics*. DFKI-Research Report, forthcoming.

Chang, C.-L., Lee, R.C.-T.: *Symbolic Logic and Theorem Proving*. Academic Press, 1973.

Cholvy, L., Demolombe, R.: *Querying a Rule Base*. Proc. of Intern. Conf. on Expert Database Systems, 365-371, 1986.

Chu, W.W., Lee, R.-C., Chen, Q.: *Using Type Inference and Induced Rules to Provide Intensional Answers*. Proc. of Intern. Conf. on Data Engineering, 396-403, 1991.

Denecker, M., De Schreye, D.: *A family of abductive procedures for normal abductive programs, the soundness and completeness wrt. Completion semantics*. CW-Report 136, K.U. Leuwen, 1992.

Eshghi, K., Kowalski, R.: *Abduction Compared with Negations by Failure.* Proc. of 6th ICLP, 1989.

Frisch A., Scherl, R.B.: *A Constraint Logic Approach to Modal Deduction.* Proc. of JELIA, Springer LNAI 478, 1990.

Frost, R.A.: *Introduction to Knowledge Base Systems*. Collins, London, 1986.

Gallier, J.: *Logic for Computer Science: Foundations of Automated Theorem Proving*. Harper and Row, 1986.

Gallier, J., Raatz, S.: *Logic Programming and Graph Rewriting*. Symp. of Logic Programming, 208-219, 1985.

Genesereth, M.R., Nilsson, N.J.: *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, 1987.

Green, C.: *Theorem-Proving by Resolution as a Basis for Question-Answering Systems.* Machine Intelligence, **4**:183-205, 1969.

van Hentenryck, P.: *Constraint Satisfaction in Logic Programming*. MIT Press, 1989.

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

Imielsky, T.: *Intelligent Query Answering in Rule Based Systems*. J. Logic Programming, **4**:229-257, 1987.

Jaffar, J., Lassez, J.-L.: *Constrained Logic Programming*. Proc. ACM Symp. on Principles of Programming Languages, 1987.

Jouannaud, J.-P., Kirchner, C.: *Solving Equations in Abstract Algebras: A Rule-based Survey of Unification*. In Lassez, J.-L., Plotkin, G. (eds.): *Essays in Honour of Alan Robinson*: MIT Press, 1991.

Kakas, A.C., Kowalski, R.A., Toni, F.: *Abductive Logic Programming*. Research report, forthcoming.

Konolige, K.: *A General Theory of Abduction*. Working Notes of Spring Symposium on Automated Deduction (Ed.: O'Rourke, P.), Techn. Report 90-32, University of California, Irvine, 1990.

Kowalski, R.: *Logic for Problem Solving*. North-Holland, 1979.

Lloyd, J.W.: *Foundations of Logic Programming*. Springer, 1984.

Luckham, D.C., Nilsson, N.J.: *Extracting Information from Resolution Proof Trees*. Artificial Intelligence, **2**(1):27–54, 1971.

Maher, M.: *Logic Semantics for a Class of Committed-Choice Programs*. Proc. 4th ICLP, 1987.

McCarthy, J.: *Circumscription – A Form of Non-monotonic Logic*. Artificial Intelligence, **13**:27-39, 1980.

Merziger, G.: *Approaches to Abductive Reasoning – An Overview*. DFKI Research Report RR-92-08, 1992.

Motro, A., Yuan, Q.: *Querying Database Knowledge*. Proc. of ACM SIGMOD Intern. Conf. on Management of Data, 173-183, 1990.

Pople, H.E.Jr.: *On the mechanization of abductive logic*. Proc. of 3rd IJCAI, 147-152, 1973

Poole, D.: *A logical framework for default reasoning*. Artificial Intelligence **36**:27.47, 1988.

Poole, D.: *Explanation and Prediction: An Architecture for Default and Abductive Reasoning*. Computational Intelligence, **5**(2):97-110, 1989.

Reiter, R.: *A Logic for Default Reasoning*. Artificial Intelligence, **13**:81-132, 1980.

O'Rourke, P.: *Working Notes of Spring Symposium on Automated Deduction*. Techn. Report 90-32, University of California, Irvine, 1990.

Shoenfield, J.R.: *Mathematical Logic*. Addison-Wesley, 1967.

Siekmann, J.H.: *Unification Theory. A Survey*. In Kirchner, C. (ed.): *Unification*. Academic Press, 1-68, 1990.

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

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
67608 Kaiserslautern
FRG

DFKI Publikationen

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

DFKI Research Reports

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

DFKI Publications

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

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

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

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

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

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

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

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

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

RR-93-03
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 Tech-niques and Decision Problems for Disunification
29 pages

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

RR-93-44
Martin Buchheit, Manfred A. Jeusfeld, Werner Nutt, Martin Staudt: Subsumption between Queries to Object-Oriented Databases
36 pages

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

RR-93-46
Philipp Hanschke: A Declarative Integration of Terminological, Constraint-based, Data-driven, and Goal-directed Reasoning
81 pages

DFKI Technical Memos

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

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

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

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

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

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

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

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

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

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

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

TM-93-04
Hans-Günther Hein: Propagation Techniques in WAM-based Architectures — The FIDO-III Approach
105 pages

DFKI Documents

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

D-93-15
Robert Laux: Untersuchung maschineller Lernverfahren und heuristischer Methoden im Hinblick auf deren Kombination zur Unterstützung eines Chart-Parsers
86 Seiten

D-93-20
Bernhard Herbig:
Eine homogene Implementierungsebene für einen hybriden Wissensrepräsentationsformalismus
97 Seiten

D-93-21
Dennis Drollinger:
Intelligentes Backtracking in Inferenzsystemen am Beispiel Terminologischer Logiken
53 Seiten