Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research
Report
RR-94-11

# A Consequence-Finding Approach for Feature Recognition in CAPP

Knut Hinkelmann

March 1994

Deutsches Forschungszentrum für Künstliche Intelligenz
GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
66123 Saarbrücken 11, FRG
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

# Deutsches Forschungszentrum für Künstliche Intelligenz

# A Consequence-Finding Approach for Feature Recognition in CAPP

**Knut Hinkelmann**

DFKI-RR-94-11

This research report will be published in the Proceedings of the Seventh International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA/AIE'94), Austin Texas, May 31-June 3, Gordon and Breach Science Publishers, 1994.

ISSN 0946-008X

# A Consequence-Finding Approach for Feature Recognition in CAPP

Knut Hinkelmann
DFKI (German Research Center for Artificial Intelligence)
Postfach 2080, 67608 Kaiserslautern, Germany
Email: hinkelma@dfki.uni-kl.de

**Abstract**

We present a rewriting approach for a consequence-finding inference of logic programs. Consequence finding restricts the derivations of a logic program to exactly those facts that depend on an explicitly given set of initial facts. The rewriting approach extends the Generalized Magic Sets rewriting, well-known from deductive databases, by an *up* propagation in addition to the usual *down* propagation. The initial motivation for this inference was to realize the abstraction phase of a knowledge-based CAPP system for lathe turning. The input to the CAPP system is a detailed description of a workpiece. During the abstraction phase characteristic parts, called features, are recognized for which predefined skeletal plans exist. Consequence finding is a method to restrict the computation such that exactly the features of the actual workpiece are derived. The same inference can also be used for checking integrity constraints: given an update of a deductive database or a logic program, consequence finding applies only those rules that are effected by the update operation.

## Contents

## 1 Introduction

In order to create a knowledge-based CAPP (Computer-Aided Process Planning) system for lathe turning [Bernardi *et al.*, 1991] we studied the actions of a human process planner [Klauck *et al.*, 1993]: regarding process planning as a problem-solving task, this procedure can be seen as an instance of the "Heuristic Classification" inference scheme [Clancey, 1985] (see Fig. 1). The expert is given the description of a workpiece (see Fig. 2). It consists of all geometrical and technological data which are necessary to generate the process plan. These data describe the surfaces of the workpiece on a very detailed level. To generate a work plan the expert abstracts from these details and looks for characteristic parts, so-called *features*, of the workpiece for which he has in mind (or in a library) a number of prefabricated skeletal plans. The connection of features and associated skeletal plans reflects the experience of the expert. The skeletal plans play the role of cases in case-based reasoning [Kolodner, 1993]. The final plan is created by merging the abstract skeletal plans associated to the recognized features and by adapting them for the particular workpiece.

It is important to realize that this observation implies that the features and skeletal plans depend on the concrete expert as well as on the concrete working environment and may vary for different companies. Therefore, in order to have a general solution for an automated CAPP system which is transferable to a changing environment, we have developed a domain-specific shell on top of a hybrid knowledge representation formalism by extending and specializing well-known knowledge representation and reasoning techniques. The shell consists of domain-specific representation languages which offer all the necessary constructs to describe the workpiece, the features, and the skeletal plans. Thus, in order to build a particular application it is easy to represent the domain-specific features and associated skeletal plans. The shell offers all the inferences for the abstraction, match, and refinement phase of the problem-solving process (Fig. 1). Inference engines of the hybrid knowledge representation and compilation laboratory CoLab [Boley *et al.*, 1993] have been tailored for the production planning application (e.g. [Meyer and Müller, 1993; Meyer and Müller, 1993; Baader and Hanschke, 1992]).

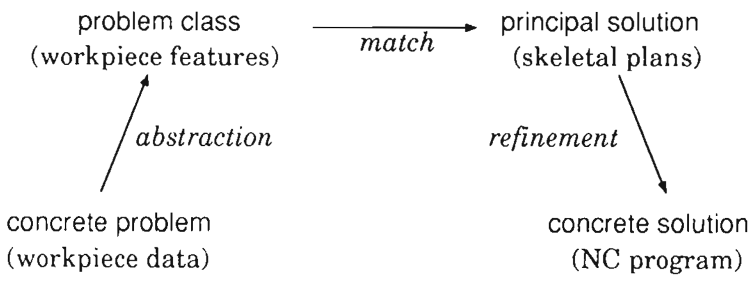

FIGURE 1: Heuristic Classification Inference Scheme

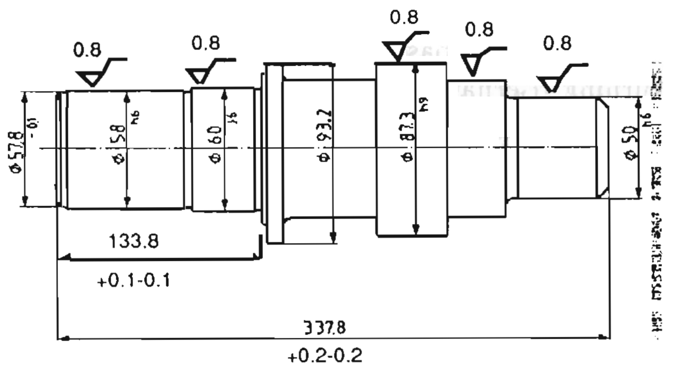

FIGURE 2: Lathe-turning Workpiece

In this paper we will present a consequence-finding inference for the abstraction phase of CAPP. Consequence finding restricts the derivations of a logic program to exactly those facts that depend on an explicitly given set of initial facts, e.g. the description of the actual workpiece. The inference procedure is not strictly bottom-up but also relies on the proof of conditions. To integrate these bottom-up and top-down phases, we extend well-known query-answering techniques for deductive databases, propagate query information *down* at compile-time, and reason strictly bottom-up at run-time. This set-oriented evaluation is efficient, in particular, if the facts reside on a database. This is especially useful since in general software systems are not used stand-alone but must have interfaces to already existing software and data. Thus, the presented realization of the consequence-finding inference not only is an elegant way to integrate bottom-up and top-down evaluation as it is necessary for this application but it also serves as a link to allow access to data stored in conventional databases.

## 2 Feature Recognition

The application problem we are dealing with for the rest of the paper is the abstraction phase of CAPP: the generation of an abstract feature description of the workpiece is the first step of the process planning process. The importance of feature recognition stems from the fact that each feature can be associated with knowledge about how the feature should be manufactured. From this point of view, feature recognition forms a major component of the CAD/CAM interface for CAPP [Chang, 1990].

The *workpiece* is composed of adjacent rotational-symmetric surfaces that are fixed to the symmetry axis of the lathe work. Attributes of each surface carry detailed geometrical and technological information. Since the surfaces are fixed to an axis,

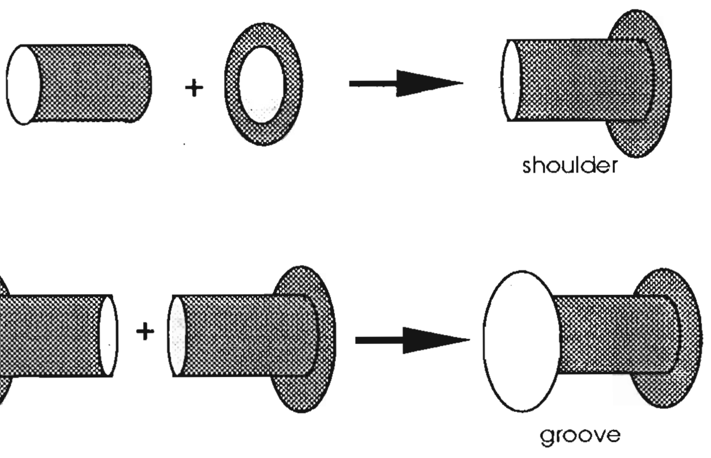

FIGURE 4: Feature Aggregation

they can be characterized by four rational numbers $r_1$, $r_2$, $c_1$, and $c_2$, two radii and two coordinates (Fig. 3). An important geometric element is the truncated cone. This surface can be specialized to a cylinder by restricting the radii as being equal. Similarly, the definitions of ascending and descending truncated cones, rings, etc. can be obtained by specialization. In addition to this geometric data the surfaces have attributes for technological information. There are also topological relationships specifying which surfaces are adjacent.

Most *features* cover a number of surfaces. This means that it is natural to define a feature as consisting of simpler features (and having some additional requirements, e.g. neighborhood). The basic components of a feature are surfaces of the workpiece. In Figure 4 it is shown schematically that the feature *shoulder* can be recognized, if we can find two neighboring surfaces, a cylinder and a ring. With two shoulders that share their ground surface we can build another feature, called a groove. Thus, finding a feature means to find instances representing the components and to generate a *new* instance aggregating the simpler features using e.g. part-of attributes. Therefore, feature recognition is equivalent to the aggregation of its components.

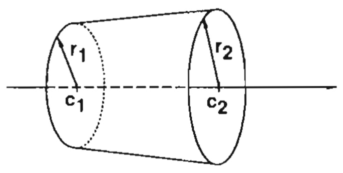

FIGURE 3: A truncated cone

In [Hanschke and Hinkelmann, 1992] we have suggested a hybrid declarative formalism for the abstraction phase of heuristic classification. Following the distinction between concepts and instances, it is rather natural to define all the possible features and surfaces as concepts in a terminological language and to represent a single case, i.e. a workpiece, by assertions. Since terminological reasoning systems as also used in CoLab directly support the abstraction mechanisms generalization

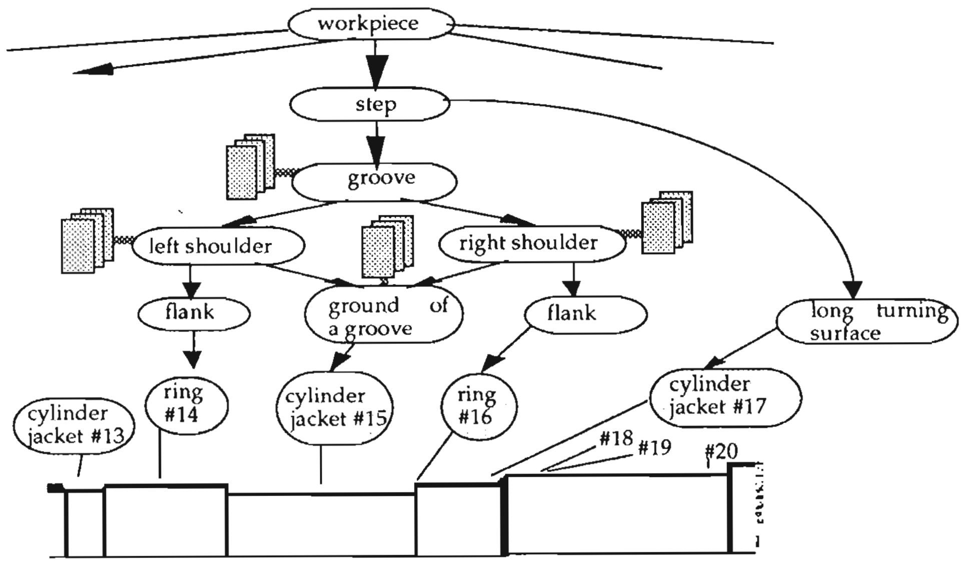

FIGURE 5: A Feature Tree Resulting from the Abstraction Phase

and classification but do not bother about aggregation, we have integrated terminological reasoning with a logical rule component: Horn clauses are a declarative knowledge-representation formalism for logic programming which serve also as query language for deductive databases [Minker, 1988; Kowalski, 1991]. Rules for feature aggregation can be expressed as Horn clauses in a natural way. The (simplified) rule[1]

$$\begin{aligned} shoulder(f(X,Y)) \quad \leftarrow \quad & cylinder(X), \\ & ring(Y), \\ & neighboring(X,Y) \end{aligned}$$

specifies that a cylinder and a neighboring ring can be regarded as a shoulder (see Fig. 4).

The task of the feature aggregation phase is to derive all the features appearing on the workpiece. Similar to parsing strategies, a bottom-up reasoning approach is most appropriate, since it is not known in advance, which kinds of features are present in the workpiece [Klauck and Mauss, 1992]. In bottom-up reasoning systems a rule is applied, if all its premises are satisfied. Then, starting from the workpiece's surfaces, a features tree is successively build up (Fig. 5). Let the workpiece description contain the following assertions:

$cylinder(cyl1)$
$ring(rng2)$
$neighboring(cyl1, rng2)$

[1]For reasons of simplicity we omitted, for instance, details of checking coordinates of feature components and technological requirements.

Then applying the above rule, bottom-up evaluation will derive that $f(cyl1, rng2)$ is a *shoulder*, since all the premises of the rule are satisfied by the facts.

Bottom-up reasoning starts with all the facts in the working memory. In each phase, however, only a subset of the rules are needed. For the abstraction phase, for instance, we want to apply only those feature rules that can be satisfied with facts representing the surfaces of the actual workpiece. This can be achieved by modularization of the rules, such that in each phase (abstraction, match, refinement, Fig. 1) only the relevant subset of the rules is accessible.

A further problem is that we do not want to use all the available facts in each phase. For the abstraction phase we must have access to facts representing the *actual* workpiece, while others are not needed. Depending on the environment in a real company, data often do not reside in a working memory but must be retrieved from a database. But in a database there might be data representing many workpieces. Additionally, there are data about many other things that are not needed in the abstraction phase and thus should not be used for testing rule applicability in this phase. An example are the facts about the tools that can be used to manufacture the workpiece which are selected in the refinement phase [Meyer, 1992]).

For feature aggregation we have developed a consequence-finding inference which derives the consequences of an explicitly given set of facts wrt to a theory of logical formulas. The theory in our case is the set of rules defining workpiece features and the initial facts are the facts describing one particular workpiece. The consequence-finding inference makes sure that exactly those facts are derived which are consequences of these initial facts.

## 3 Integrity Constraints

The same consequence-finding inference that is applied for feature aggregation can also be used to detect whether database updates (e.g. because of changes in the design of the workpiece) would lead to inconsistencies. Consider a logic program with integrity constraints denoting negative or disjunctive knowledge. These integrity constraints are represented as denials, i.e. clauses with empty head. Eshghi and Kowalski use this kind of integrity constraints for their abduction procedure [Eshghi and Kowalski, 1989]. We can also represent them as clauses with the special atom *inconsistent* as conclusion [Manthey and Bry, 1987].

**Example 1** Let $S1$ and $S2$ be two surfaces and let the relations *coordinate*1 and *coordinate*2 denote the attributes $c_1$, and $c_2$ of the surface (see Fig. 3) Then the following rule demands that two connected solids must coincide at their contact point:

$$\begin{array}{lcl} \mathit{inconsistent} & \leftarrow & \mathit{neighboring}(S1, S2), \\ & & \mathit{coordinate2}(S1, C1), \\ & & \mathit{coordinate1}(S2, C2), \\ & & C1 \neq C2. \end{array}$$

A real database will have many of these integrity constraints. Let's assume that the facts

$\mathit{coordinate2}(\mathit{cyl1}, 5)$
$\mathit{coordinate1}(\mathit{truncone2}, 6)$

are in the database. Now we want to connect these elements. Using a proof-finding approach one has to assert the new fact $\mathit{neighboring}(\mathit{cyl1}, \mathit{truncone2})$ and then ask the query

?- *inconsistent*

This procedure would invoke all integrity constraints in backward direction even if they are independent from the new fact. Instead, it would be more efficient to derive only those facts, that are consequences of this new assertion. In [Eshghi and Kowalski, 1989] it is argued to do this kind of constraint checking by forward reasoning starting with the new fact. Forward reasoning alone, however, is not sufficient.

**Example 2** Consider the following program:

$$\begin{array}{lcl} \mathit{endpoint}(X) & \leftarrow & \mathit{cone}(X), \mathit{radius2}(X, 0) \\ \mathit{startpoint}(X) & \leftarrow & \mathit{cone}(X), \mathit{radius1}(X, 0) \\ \mathit{cone}(c1) & & \\ \mathit{radius1}(c1, 0) & & \\ \mathit{radius2}(c1, 20) & & \\ \mathit{cylinder}(\mathit{cyl1}) & & \end{array}$$

The following integrity constraints say that you cannot connect two elements if there is no surface but only a point at the end of one element (see Fig. 6):

$$\begin{array}{lcl} \mathit{inconsistent} & \leftarrow & \mathit{neighboring}(I1, I2), \\ & & \mathit{endpoint}(I1) \\ \mathit{inconsistent} & \leftarrow & \mathit{neighboring}(I1, I2), \\ & & \mathit{startpoint}(I2) \end{array}$$

Adding the new fact $\mathit{neighboring}(\mathit{cyl1}, c1)$ would lead to an inconsistency which will not be detected by forward chaining alone. Additionally we need to prove whether the premise $\mathit{startpoint}(c1)$ can be satisfied.

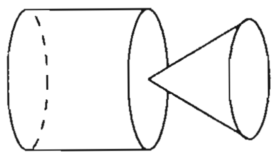

FIGURE 6: Forbidden Connection

The extended SLDNF resolution of [Sadri and Kowalski, 1988] combines forward and backward chaining depending on whether a positive or negative literal is resolved upon. In [Manthey and Bry, 1987] a model-generation approach has been applied for this problem. In the following we will regard checking of integrity constraints as a consequence-finding problem. We will restrict the generation of a complete minimal model by specializing the relevant rules. Instead of using a backward-chaining component to prove premises, we apply the Generalized Magic Sets rewriting algorithm well-known from deductive databases.

## 4 Consequence Finding

Consequence-finding had been investigated intensively after the resolution principle has been invented but then hasn't been studied very much for a number of years. Recently, [Inoue, 1991] presented a consequence-finding implementation based on ordered linear resolution. In this section we will give an overview on consequence finding. Then in the next section we will present an approach that is efficient for applications that use data on external databases.

The consequence-finding problem [Lee, 1967] may be formulated as:

> Given a set of statements $A_1, \ldots, A_n$, find a statement $B$ such that $B$ follows from $A_1, \ldots, A_n$ [Slagle *et al.*, 1969].

There are a number of specializations of this general formulation. They are characterized by restrictions of the statement $B$ which we are interested in. In [Lee, 1967] the consequence-finding problem is expressed in a more restricted form:

> Given a set of formulas $T$ and a resolution procedure P, for any logical consequence $D$ of $T$, can P derive a logical consequence $C$ of $T$ such that $C$ subsumes $D$?

In [Slagle *et al.*, 1969] consequence finding is examined for prime (non-trivial) consequences. Also proof finding can be regarded as a special case of consequence finding if $B$ is the empty clause.

Inoue presents an extended ordered linear resolution strategy which is complete for consequence finding in the sense that only clauses having a certain property (called characteristic clauses) should be found [Inoue, 1991]. A useful specialization is to compute exactly the newly derivable consequences caused by new information added to the theory. These new consequences are called new characteristic clauses.

Here we will adapt this latter version of the consequence-finding problem for logic programs. In logic programming, deductive databases, and rule-based reasoning we are not interested in general consequences but only in ground unit clauses (i.e. ground facts). Thus, our consequence-finding problem can be informally described as:

> Given a set of Horn clauses $T$ and a set of ground unit clauses $F \subseteq T$ ($F$ is called the set of initial facts), derive all the consequences $B$ of $T$ such that $B$ is a ground unit clause and at least one clause $C \in F$ has been used to derive $B$.

## 5 How to Implement Consequence Finding

From Example 2 about integrity constraints it became clear that forward chaining and backward chaining may interchange to solve the consequence-finding problem. An obvious approach would be to use an integrated system with forward and backward chaining components.

The forward chaining process starts with the set $F$ of initial facts. A rule

$$C \leftarrow P_1, \ldots, P_n$$

is triggered if at least one $P_i, i \in \{1, \ldots, n\}$ is unifiable with a fact $f \in F$. Then the remaining premises have to be proved. Besides changing the implementation, this can be achieved by a program transformation declaring the premises as backward-provable and evaluating them by the backward chaining component. Since each of the premises can serve as a trigger premise, this approach is equivalent to the following transformation of the above rule:

$$\begin{array}{lcl} C & \leftarrow & P_1, prove(P_2), \ldots, prove(P_n) \\ C & \leftarrow & prove(P_1), P_2, \ldots, prove(P_n) \\ C & \leftarrow & prove(P_1), prove(P_2), \ldots, P_n \end{array}$$

The rules are evaluated by forward chaining. Each premise can act as a trigger while the remaining ones have to be tested by backward chaining. The predicate *prove* acts as a built-in operator activating the backward chaining component to prove its argument.

In the following we will present a rewriting approach for consequence finding which avoids a call to the backward chaining system. It is an extension of well-known rewriting techniques for query-answering in deductive databases. These techniques allow query answering by bottom-up evaluation. Information about variable bindings given by the query is propagated *down* into the bodies of the rules at compile-time. When the rewritten program is evaluated by a bottom-up fixpoint procedure like semi-naive evaluation [Bancilhon and Ramakrishnan, 1986], only those facts are derived that are necessary to answer a query. Since these techniques in some sense integrate bottom-up and top-down reasoning, it seems natural to extend them for consequence finding.

## 5.1 Generalized Magic Sets Rewriting

Before we will present the consequence-finding transformation, let us first give an impression of the rewriting techniques for query answering. Readers already familiar with this approach may skip this subsection. Rewriting strategies like Generalized Magic Sets [Bancilhon *et al.*, 1986; Beeri and Ramakrishnan, 1991] or more recently Magic Templates [Ramakrishnan, 1988] have been developed for efficient query answering in deductive databases. Magic-Sets rewriting propagates the values of bound arguments of a query *down* to the premises of applicable rules at compile time. Consider a logic program containing two rules:

$$\begin{aligned} r(X,Y) &\leftarrow t(X,Z), s(Z,Y) \\ s(Z,Y) &\leftarrow u(W,Z), v(Y) \end{aligned}$$

and the query

?- $r(a,Y)$ .

By using a top-down query answering approach, the query would be unified with the head of the first clause binding variable $X$ to $a$. This means that $t$ is also called with first argument bound to $a$. Bottom-up reasoning, however, does not have this kind of information passing. The only kind of information passing for bottom-up reasoning is called sideway information passing: by solving a premise predicate variable bindings are obtained which can be passed to another premise in the same rule to restrict the computation for that predicate. Magic-Sets rewriting introduces an additional premise $magic\_r^{bf}(X)$ to imitate the information passing strategy of top-down reasoning by sideway information passing. The arguments of the new premise correspond to the bound arguments of the query. To pass the actual value of $X$, a new fact – called Magic Seed – is asserted. A superscript *bf* denotes that (according to a sideway information passing strategy) the first argument of a literal is bound and the second argument is free:

$$\begin{aligned} r^{bf}(X,Y) \leftarrow\ & magic\_r^{bf}(X), \\ & t^{bf}(X,Z), \\ & s^{bf}(Z,Y) \end{aligned}$$
$$magic\_r^{bf}(a)$$

Now, evaluating this rule by a bottom-up strategy will satisfy the first premise $magic\_r^{bf}(X)$ with the new fact binding $X$ to $a$ as it would be the case by top-down evaluation. By satisfying the second premise - with $X$ bound to $a$ - the variable $Z$ will also be bound such that the computation of $s$ will be restricted, too. The Generalized Magic Sets method further propagates this binding down to the rules defining $t$ and $s$. For further details see [Beeri and Ramakrishnan, 1991].

## 5.2 Consequence-Finding Transformation

Now we will extend Generalized Magic Sets (GMS) to support the consequence-finding inference. Instead of calling a backward chaining system to prove the remaining premises of triggered rules, we can apply the GMS rewriting approach. It specializes a logic program by introducing additional rules and predicates. When the rewritten program is evaluated by a bottom-up fixpoint procedure, exactly the consequences of the initial facts are derived. In our system we use the semi-naive evaluation strategy for logic programs [Bancilhon and Ramakrishnan, 1986] which avoids multiple derivations of equivalent facts. Because it is a set-oriented strategy it is very efficient if facts have to be retrieved from a database.

GMS rewriting needs a query to start the transformation. For consequence finding, however, we do not have a query but a number of facts from which to reason forward. For the consequence-finding inference, the *down* propagation of the Generalized Magic Sets rewriting is extended by an *up* propagation phase. For the initial facts we can find the applicable rules by testing if a premise is unifiable with an initial fact. The remaining premises of this rule have to be proved. This means that a GMS rewriting has to be made for them. Their adornments can be derived by the sideway information passing strategy. For each of the rules found in this first step we now look for successor rules, i.e. rules which have a premise that is unifiable with the conclusion of an already found rule. For the premises of these rules again a GMS rewriting is made. Finding successor rules corresponds to the up-propagation phase. The propagation stops if no *new* successor rule is found. In the following we will demonstrate bottom-up consequence finding with an example.

**Example 3** Consider a logic program containing the following rules where the predicates $b_1, b_2, b_3, b_4$, and $b_5$ are base predicates:

$$\begin{array}{lll}
r_1: & p(X,Y) & \leftarrow & p_1(X,Z), b(Z,V), p_3(X,V,Y) \\
r_2: & q(U,V,W) & \leftarrow & q_1(U,Z,W), p(Z,V) \\
r_3: & p_1(X,Z) & \leftarrow & b_1(Z,Y), b_2(Y,X) \\
r_4: & p_3(X,V,Y) & \leftarrow & b_3(X,V,Z), b_4(Z,Y) \\
r_5: & q_1(U,Z,W) & \leftarrow & b_1(X,Z), b_5(U,X,W)
\end{array}$$

Let's assume we want to derive the consequences of the fact $b(a,b)$. That is, the

*Upmagic Seed* specifying the bound arguments of the initial facts - for consequence finding is

$upmagic\_b(a, b)$

To derive the consequences of a given fact, a first step is to select all the rules which can be triggered by this fact. In our example it is rule $r_1$. The arguments of the initial fact $upmagic\_b(a, b)$ bind variables $Z$ and $V$ by bottom-up propagation. If we assume that the sideway information passing strategy is determined by a left-to-right evaluation of the remaining premises, we see that we have to prove $p_1^{fb}, p_3^{bbf}$. Thus, the rewritten rule $r_1'$ is

$$\begin{array}{lll} p(X,Y) & \leftarrow & upmagic\_b(Z,V), \\ & & p_1^{fb}(X,Z), \\ & & p_3^{bbf}(X,V,Y) \end{array}$$

The derived facts of rule $r_1$ can themselves trigger further rules. In our example rule $r_2$ is such a successor rule. Analogously as before, the values for $p(X, Y)$ are propagated **up** to rule $r_2$, resulting in the following rule:

$$q(U,V,W) \leftarrow p(Z,V), q_1^{fbf}(U,Z,W)$$

For the adorned predicates $p_1^{fb}, p_3^{bbf}$ (from rule $r_1$) and $q_1^{fbf}$ (from rule $r_2$) we apply the usual Generalized Magic Sets rewriting, propagating the initial values **down** to the rules defining $p_1$, $p_3$ and $q_1$:

$$\begin{array}{lll} p_1^{fb}(X,Z) & \leftarrow & magic\_p_1^{fb}(Z), \\ & & b_1(Z,Y), \\ & & b_2(Y,X) \\ p_3^{bbf}(X,V,Y) & \leftarrow & magic\_p_3^{bbf}(X,V), \\ & & b_3(X,V,Z), \\ & & b_4(Z,Y) \\ q_1^{fbf}(U,Z,W) & \leftarrow & magic\_q_1^{fbf}(Z), \\ & & b_1(X,Z), \\ & & b_5(U,X,W) \end{array}$$

The argument values of the trigger fact, i.e. the initial fact $upmagic\_b(a, b)$ and the derived consequences $p(Z, V)$, are propagated **up** as initial values (seeds) for these rules:

$$\begin{array}{lll} magic\_p_1^{fb}(Z) & \leftarrow & upmagic\_b(Z,V) \\ magic\_p_3^{bbf}(X,V) & \leftarrow & upmagic\_b(Z,V), p_1^{fb}(X,Z) \\ magic\_q_1^{fbf}(Z) & \leftarrow & p(Z,V) \end{array}$$

This kind of transformation has to be performed for each of the initial facts. Then the evaluation of the rewritten program by a bottom-up reasoning system, e.g. the semi-naive strategy, solves the consequence-finding problem.

## 5.3 Specifying initial Facts

Instead of explicitly giving a single initial fact as in Example 3, there are a number of possibilities for specifying mulitple initial facts. The transformation itself does not change, only the representation of the Upmagic Seed has to be adapted.

- For a set of initial facts an Upmagic Seed is generated for each element of the set. For a workpiece the initial facts are the descriptions of the surfaces. The set of initial facts may be $F = \{cylinder(cyl1), ring(rng2), cylinder(cyl3)\}$. The corresponding seeds are:

  $upmagic\_cylinder(cyl1)$
  $upmagic\_ring(rng2)$
  $upmagic\_cylinder(cyl2)$

  The first and third fact differ only for their constants. This means that exactly the same transformation will be made for them. This is recognized by the rewriting algorithm: each rule will be generated exactly ones.

- It is not necessary to explicitly present every initial fact. To express that every surface neighboring cylinder *cyl1* should serve as initial fact we can use a variable: *neighboring(cyl1,X)*. In this case the Upmagic Seed cannot be expressed by a single fact since it is not range-restricted. The following rule, however, accomplishes the needs:

  $$upmagic\_neighboring(cyl1, X) \leftarrow neighboring(cyl1, X)$$

  Variables can also be used to express conditions about initial facts, for instance, if two arguments should coincide in their values: *b(X, Y, X)*.

- For feature aggregation we can go one step further: since for the facts describing the workpiece surfaces all arguments are bound, we need to tell the rewriting algorithm only the predicates representing the surfaces, e.g. {*truncone, cylinder, ring*}. The corresponding Upmagic Seeds are:

  $upmagic\_truncone(X) \leftarrow truncone(X)$
  $upmagic\_cylinder(X) \leftarrow cylinder(X)$
  $upmagic\_ring(X) \leftarrow ring(X)$

  The facts describing the workpiece trigger these rules, which themselves trigger the rules deriving their consequences. Thus, the rewritten rule system then is a specialized rule system, which exactly derives the feature tree for every workpiece. This means, that rewriting has to be done only once after the feature rules have been defined and the specialized rule system can be reused for various workpieces.

- Similar to $\mathcal{LDL}$ [Naqvi and Tsur, 1989], we can also give fact *forms* as initial facts: it is sufficient to tell the rewriting algorithm which of the arguments are bound (designing them with a special symbol $) instead of explicitly giving the value of bound arguments, e.g. *neighboring($,X)*.

**Example 4** Consider again the rules and facts of Example 1

$$\begin{array}{lcl}
endpoint(X) & \leftarrow & cone(X), radius2(X,0) \\
startpoint(X) & \leftarrow & cone(X), radius1(X,0) \\
cone(c1) & & \\
radius1(c1,0) & & \\
radius2(c1,20) & & \\
cylinder(cyl1) & &
\end{array}$$

with integrity constraints

$$\begin{array}{lcl}
inconsistent & \leftarrow & neighboring(I1,I2), \\
 & & endpoint(I1) \\
inconsistent & \leftarrow & neighboring(I1,I2), \\
 & & startpoint(I2)
\end{array}$$

If the initial facts are specified as *neighboring(X, Y)*, the rewritten knowledge base is:

$$\begin{array}{lcl}
upmagic\_neighboring(X,Y) \leftarrow & & \\
 & & neighboring(X,Y) \\
inconsistent & \leftarrow & upmagic\_neighboring(I1,I2), \\
 & & endpoint^b(I1) \\
inconsistent & \leftarrow & upmagic\_neighboring(I1,I2), \\
 & & startpoint^b(I2) \\
endpoint^b(X) & \leftarrow & magic\_endpoint^b(X), \\
 & & cone(X), \\
 & & radius2(X,0) \\
startpoint^b(X) & \leftarrow & magic\_startpoint^b(X), \\
 & & cone(X), \\
 & & radius1(X,0) \\
magic\_endpoint^b(I1) \leftarrow & & \\
 & & upmagic\_neighboring(I1,I2) \\
magic\_startpoint^b(I2) \leftarrow & & \\
 & & upmagic\_neighboring(I1,I2) \\
cone(c1) & & \\
radius1(c1,0) & & \\
radius2(c1,20) & & \\
cylinder(cyl1) & &
\end{array}$$

Adding the new fact $neighboring(cyl1, c1)$ will derive $upmagic\_neighboring(cyl1, c1)$, which then initiates the derivation of $magic\_startpoint^b(c1)$ and $startpoint^b(c1)$ which finally leads to the derivation of *inconsistent* as a consequence of this update operation.

## 6 Conclusion

We have presented a consequence-finding approach that has been developed for the abstraction phase of a process planning system. The theory is the set of rules defining workpiece features and the initial facts are the facts describing the workpiece [Boley *et al.*, 1993]. The approach extends the Generalized Magic Sets rewriting approach of deductive databases by an *up* propagation in addition to the usual *down* propagation. Thus, it has been investigated how the inference for query answering can be embedded into a complex problem-solving process. Because of the set-oriented reasoning strategy it is efficient, in particular if the facts reside on a database. This shows that declarative knowledge representation can be applied to real-world problems.

The consequence-finding transformation is an example of our knowledge compilation paradigm. Instead of having its own interpreter for each problem solving inference, we prefer a transformation approach, where a knowledge base – described in a declarative representation formalism – is specialized for a particular task. The rewritten knowledge base can be interpreted by a "general-purpose" reasoning system. Thus, we have the advantage that we can reduce the number of special-purpose inference engines. This is especially useful in a hybrid representation system. The larger the number of interpreters the greater would be the effort for integrating them into the system, because the interpreters have to cooperate to solve the overall problem. Investigating the opportunities of knowledge compilation for an application in a technical environment has been the objective of the development of the hybrid knowledge representation and compilation laboratory CoLab.

## References

[Baader and Hanschke, 1992] Franz Baader and Philipp Hanschke. Extensions of Concept Languages for a Mechanical Engineering Application. In *Proceedings of the 16th German Workshop on Artificial Intelligence (GWAI-92)*, number 671 in Lecture Notes on Artificial Intelligence. Springer-Verlag, September 1992. Also available as DFKI Research Report RR-92-36.

[Bancilhon and Ramakrishnan, 1986] Francois Bancilhon and Raghu Ramakrishnan. An amateur's introduction to recursive query processing strategies. In *Proceedings of the ACM SIGMOD Conference*, pages 16–52. ACM, 1986.

[Bancilhon *et al.*, 1986] F. Bancilhon, D. Maier, Y. Sagiv, and J. D. Ullman. Magic sets and other strange ways to implement logic programs. In *Proceedings 5th ACM SIGMOD-SIGACT Symposium on Principles of Database Systems*, pages 1–15. ACM, 1986.

[Beeri and Ramakrishnan, 1991] Catriel Beeri and Raghu Ramakrishnan. On the power of magic. *Journal of Logic Programming*, 10:255–299, October 1991.

[Bernardi *et al.*, 1991] Ansgar Bernardi, Harold Boley, Knut Hinkelmann, Philipp Hanschke, Christoph Klauck, Otto Kühn, Ralf Legleitner, Manfred Meyer, Michael M. Richter, Gabi Schmidt, Franz Schmalhofer, and Walter Sommer. ARC-TEC: Acquisition, Representation and Compilation of Technical Knowledge. In *Expert Systems and their Applications: Tools, Techniques and Methods*, Avignon, France, 1991. Also available as Research Report RR-91-27, DFKI GmbH.

[Boley *et al.*, 1993] Harold Boley, Philipp Hanschke, Knut Hinkelmann, and Manfred Meyer. COLAB: A hybrid knowledge compilation laboratory. Research Report RR-93-08, DFKI, Kaiserslautern, Germany, January 1993. Also to appear in *Annals of Operations Research.*

[Chang, 1990] T. C. Chang. *Expert Process Planning for Manufacturing.* Addison-Wesley, 1990.

[Clancey, 1985] J. Clancey, W. Heuristic classification. *Artificial Intelligence*, 27:289–350, 1985.

[Eshghi and Kowalski, 1989] Kave Eshghi and Robert Kowalski. Abduction compared with negation by failure. In *6th International Conference on Logic Programming (ICLP'89)*, 1989.

[Hanschke and Hinkelmann, 1992] Philipp Hanschke and Knut Hinkelmann. Combining terminological and rule-based reasoning for abstraction processes. In *Proceedings of the 16th German Workshop on Artificial Intelligence (GWAI-92)*, number 671 in Lecture Notes on Artificial Intelligence. Springer-Verlag, September 1992. Also available as DFKI Research Report RR-92-40.

[Inoue, 1991] Katsumi Inoue. Consequence-finding based on ordered linear resolution. In *Proc. of the $12^{th}$ IJCAI*, Sidney, Australia, 1991.

[Klauck and Mauss, 1992] Christoph Klauck and Jakob Mauss. A heuristic drive chart-parser for attributed node labeled graph grammars and its application to feature recognition in cim. In *International Workshop on Structural & Syntactic Pattern Recognition (SSPR'92)*, 1992.

[Klauck *et al.*, 1993] Christoph Klauck, Ansgar Bernardi, and Ralf Legleitner. Heuristic classification for automated CAPP. In *Proceddings of the Eleventh Conference on Applications of Artificial Intelligence (AAI-XI) - Knowledge-Based Systems in Aerospace and Industry, SPIE*, 1993.

[Kolodner, 1993] Janet Kolodner. *Case-Based Reasoning.* Morgan Kaufman Publisher, 1993.

[Kowalski, 1991] Robert A. Kowalski. Logic Programming in Artificial Intelligence. In *IJCAI-91*, pages 596–603, 1991.

[Lee, 1967] R. C. T. Lee. *A Completeness Theorem and a Computer Program for Finding Theorems Derivable from Given Axioms.* PhD thesis, University of California, Berkeley, 1967.

[Manthey and Bry, 1987] Rainer Manthey and Francois Bry. SATCHMO: a theorem prover implemented in prolog. In *Conference on Automated Deduction, CADE*, 1987.

[Meyer and Müller, 1993] Manfred Meyer and Jörg Müller. Finite Domain Consistency Techniques: Their Combination and Application in Computer-Aided Process Planning. In *Seventh International Symposium on Methodologies for Intelligent Systems (ISMIS'93), Trondheim, Norway*, number 689 in Lecture Notes in Artificial Intelligence (LNAI), pages 385–394. Springer-Verlag, Berlin, Heidelberg, June 1993.

[Meyer, 1992] Manfred Meyer. Using Hierarchical Constraint Satisfaction for Lathe-Tool Selection in a CIM Environment. In *Fifth International Symposium on Artificial Intelligence (ISAI), Cancun, Mexico*, pages 167–177. AAAI Press, December 1992.

[Minker, 1988] Jack Minker. *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann Publishers, Inc., 1988.

[Naqvi and Tsur, 1989] Shamim Naqvi and Shalom Tsur. *A Logical Language for Data and Knowledge Bases.* Computer Science Press, Rockville, Maryland USA, 1989.

[Ramakrishnan, 1988] Raghu Ramakrishnan. Magic templates: A spellbinding approach to logic programms. In R.A. Kowalski and K.B. Bowen, editors, *Proceedings of the 5th International Conference and Symposium on Logic Programming*, 1988.

[Sadri and Kowalski, 1988] Fariba Sadri and Robert Kowalski. A theorem-proving approach to database integrity. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 313–362. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1988.

[Slagle *et al.*, 1969] J. R. Slagle, C. L. Chang, and R. C. T. Lee. Completeness theorems for semantic resolution in consequence-finding. In *IJCAI-69*, pages 281–285, 1969.

**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 $\mu$-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-93-30**
*Stephen P. Spackman, Elizabeth A. Hinkelman:*
Corporate Agents
14 pages

**RR-93-31**
*Elizabeth A. Hinkelman, Stephen P. Spackman:*
Abductive Speech Act Recognition, Corporate Agents and the COSMA System
34 pages

**RR-93-32**
*David R. Traum, Elizabeth A. Hinkelman:*
Conversation Acts in Task-Oriented Spoken Dialogue
28 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-93-43**
*M. Bauer, G. Paul:* Logic-based Plan Recognition for Intelligent Help Systems
15 pages

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

**A Consequence-Finding Approach for Feature Recognition in CAPP**

**Knut Hinkelmann**

**RR-94-11**
Research Report