Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Technical
Memo
TM-91-13

Forward Logic Evaluation: Developing a Compiler from a Partially Evaluated Meta Interpreter

Knut Hinkelmann

October 1991

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

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

Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11
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 und Saarbrücken is a non-profit organization which was founded in 1988 by the shareholder companies ADV/Orga, AEG, IBM, Insiders, Fraunhofer Gesellschaft, GMD, Krupp-Atlas, Mannesmann-Kienzle, Philips, 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

Forward Logic Evaluation: Developing a Compiler from a Partially Evaluated Meta Interpreter

Knut Hinkelmann

DFKI-TM-91-13

Thies report is the revised version of the contribution presented at the workshop "Alternative Konzepte für Sprachen und Rechner", Bad Honnef 1991.

This work has been supported by a grant from The Federal Ministry for Research and Technology (FKZ ITW-8902 C4).

Forward Logic Evaluation: Developing a Compiler from a Partially Evaluated Meta Interpreter

Knut Hinkelmann

Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI)

Postfach 2080

6750 Kaiserslautern

email: hinkelma@dfki.uni-kl.de

Abstract

Pure horn logic does not prescribe any inference strategy. Clauses could be applied in forward and backward direction. This paper presents a translation of rules into forward clauses which simulate a forward chaining deduction if executed by Prolog's resolution procedure. Premises of forward rules are verified by Prolog's backward proof procedure using the original clauses. Thus, without any changes to the Prolog interpreter integrated bidirectional reasoning of horn rules is possible. The translation is obtained from a meta interpreter for forward reasoning written in horn logic. Data-driven partial evaluation of this meta interpreter wrt the original horn clauses results in a forward program. The approach is applied to the problem of recognizing production-specific features from a product model. A product model contains geometrical, topological, and technological information collected during the design phase. From these data features giving valuable hints about manufacturing are derived.

Keywords: logic programming, Prolog, bottom-up reasoning, forward chaining, partial evaluation, meta reasoning, production planning

Table of Contents

1 Introduction

Horn logic is a declarative knowledge representation formalism for logic programming systems. Horn logic is based on horn clauses - clauses with at most one positive literal. In principle there are two reasoning directions for horn clauses. Top-down reasoning starts with a query applying the clauses in backward direction until a fact is reached for a goal. The result of backward reasoning is a substitution for the variables of the query. Top-down reasoning can be implemented rather efficient: for Prolog's SLD-resolution [Llo87] the Warren Abstract Machine (WAM, [War83]) is an often implemented architecture. Bottom-up strategies start with the facts and apply the rules[1] in forward direction. Naive and semi-naive evaluation strategies [Ban86a], [Ban88] reason forward until a fixpoint is reached making all the knowledge explicit as facts. Bidirectional reasoning can be achieved by explicit choice of the reasoning direction or by dynamic choice (cp. $\mathcal{LDL}$ [Naq89], [Tre87]).

Although horn logic itself does not prescribe any inference strategy, a kind of top-down reasoning is mostly used in logic programming, e.g. in Prolog. Several attempts have been made to integrate forward chaining into Prolog. Common to most of these approaches is that they use disjoint sets of rules for both reasoning directions ([Mor81], [Cha87], [Fin89]). In this paper an approach is presented to explicitly perform, besides the usual backward chaining, forward reasoning over the *same* set of horn clauses. The original horn rules are translated into special forward clauses. The translation is obtained from a meta interpreter for forward reasoning by partial evaluation. Applying Prolog's backward resolution to these clauses simulates forward reasoning of the original rules. A rule is applied in forward direction if one of its premises is unifiable with one of the initial facts. The remaining premises are verified by Prolog's SLD-resolution. The conclusion is asserted and can trigger further forward rules.

The next section will give a short introduction into the intended application of the system: production planning. The representation of the product model and the feature specification in horn logic will be presented. In Section 3 a forward reasoning meta interpreter is described and applied to feature recognition (Section 3.2). Partial evaluation of this meta interpreter is demonstrated in Section 4. In Section 5 a vertical compilation developed from this partially evaluated interpreter is explained.

2 Production-specific Features

The aim of our application is to generate a working plan for a given workpiece on a lathe turning machine. Production planning starts with a *product model* containing a geometrical model, topological information, tolerances, technological data etc. These data are originating from the concept and design phase missing the information how the workpiece

---

[1] We will speak of horn rules if we mean horn clauses which could to be applied also in forward direction.

should be manufactured. On the other hand significant features, which can be derived from the product model, give valuable hints on how the workpiece should be produced. For instance, narrow grooves like the ones in the example workpiece (Fig. 2) are manufactured with a piercing tool. A first step in production planning is the recognition of those production-specific features giving indication on manufacturing.

2.1 Surface Representation of Workpieces

In the ARC-TEC project (Acquisition, Representation and Compilation of TEChnical knowledge) a symbolic product model representation has been developed [Ber90]. Since the emphasize will be on the forward evaluation strategy, only a simplified geometrical model will be considered in the rest of this paper. The geometrical representation is based on a surface boundary representation. There are a number of primitive surfaces by which a whole workpiece can be described. In Figure 1 only those surfaces are presented that are relevant to model our example workpiece shown in Figure 2. A coordinate system is given with z-axis being the rotation axis of the workpiece.

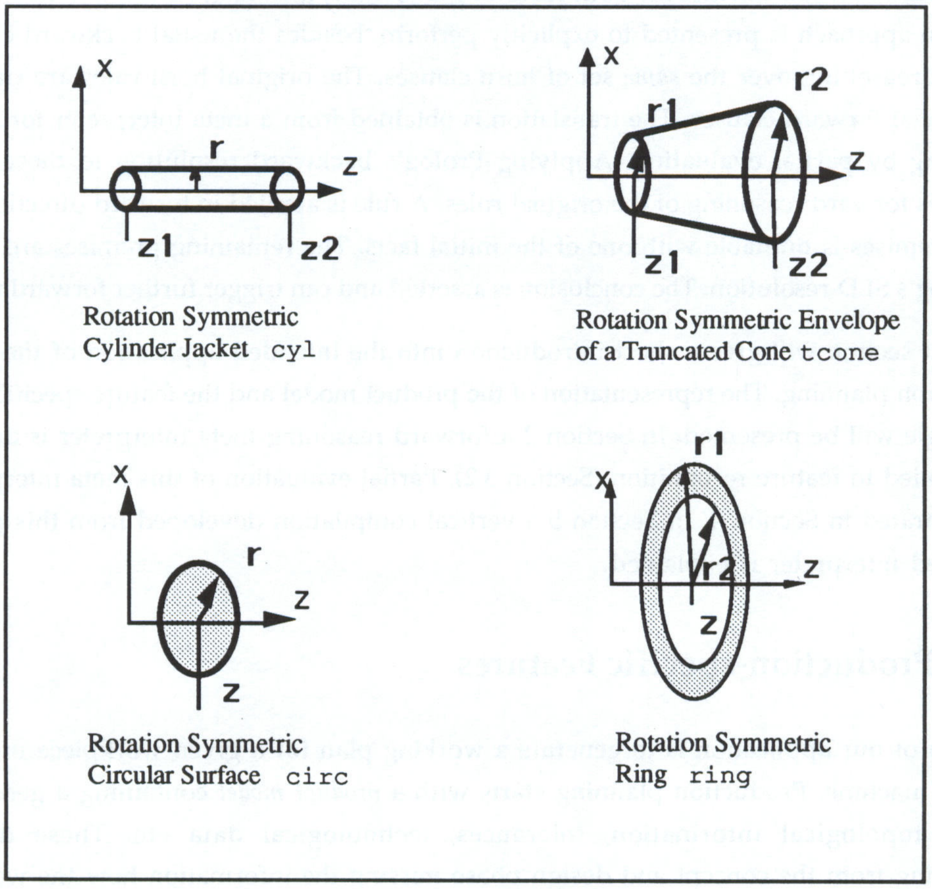

Figure 1: Surfaces

Since we use horn logic as representation formalism surfaces are represented as facts. The predicate specifies the type of the surface. Each primitive surface is characterized by a definite identifier and a number of parameters. For a cylinder jacket the parameters are: two co-ordinates denoting the bounds of its axis (`Leftcoord` and `Rightcoord`), its radius `Rad` and a parameter `Mdir` indicating whether the cylinder is filled (a "normal" cylinder, indicated by -) or whether it is hollow (e.g. a drill-hole, indicated by a +):

```
cyl(Id,Leftcoord,Rightcoord,Rad,Mdir)
```

A cylinder is a special truncated cone with equal radius at both sides. Therefore for a truncated cone a second radius parameter is needed:

```
tcone(Id,Leftcoord,Rightcoord,RadL,RadR,Mdir)
```

A circular surface has length zero; it has only one co-ordinate:

```
circ(Id,Refpoint,Rad,Mdir)
```

A ring is a circular disk with another smaller disk subtracted from it and both disks are colinear. It has two radii, one for the inner (`Radi`) and one for the outer circle (`Rado`):

```
ring(Id,Coord,Rado,Radi,Mdir)
```

Now we can present the representation of our example workpiece (Fig. 3). The identifiers are numbered starting from `s1` for the leftmost circle to `s17` for the rightmost circle.

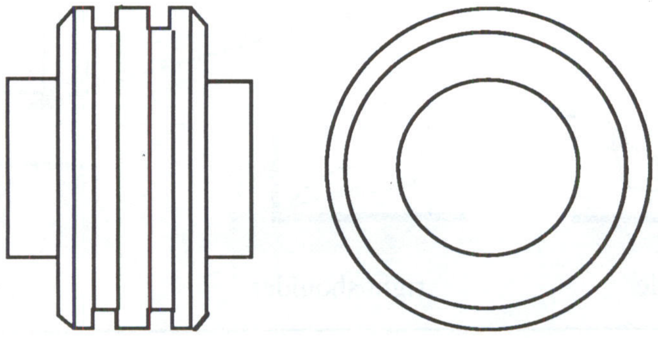

Figure 2: Example workpiece

```
circ(s1,0,100,+).              ring(s10,137,180,162,-).
cyl(s2,0,54,100,-).            cyl(s11,137,160,162,-).
ring(s3,54,179,100,+).         ring(s12,160,180,162,+).
tcone(s4,54,55,179,180,-).     cyl(s13,160,189,180,-).
cyl(s5,55,84,180,-).           tcone(s14,189,190,180,179,-).
ring(s6,84,180,162,-).         ring(s15,190,179,100,-).
cyl(s7,84,107,162,-).          cyl(s16,190,240,100,-).
ring(s8,107,180,162,+).        circ(s17,240,100,-).
cyl(s9,107,137,180,-).
```

Figure 3: Facts representing the example workpiece

2.2 Specification of Production-specific Features

The number of possible features can be very large. Each feature covers one or more surfaces. In Figure 4 three significant features for lathe-tooling are shown together with a tool by which they could be manufactured. Their horn clause definitions are presented in Figure 5.

A shoulder consists of two surfaces: a cylinder neighboring a ring. Neighborhood is specified by equal co-ordinates. The radius of the cylinder and the inner radius of the ring must be equal. We distinguish a left shoulder (the ascending ring is to the left of the cylinder) and a right shoulder (the ring is to the right). The depth of a shoulder is equal to the difference between the ring's outer and inner radii. Another significant feature is a groove consisting of three components: a cylinder with ascending rings at both sides (Figure 4). Its width is equal to the length of the cylinder.

For feature recognition a data-driven bottom-up strategy is preferable to a goal-directed one. Instead of enumerating all the possible features and testing whether they can be found in a product model, one starts with (a subset of) the facts describing the product model to identify occuring features. This problem-solving method can be specified explicitly as a forward reasoning meta interpreter which is written in the same language as the domain model: horn logic.

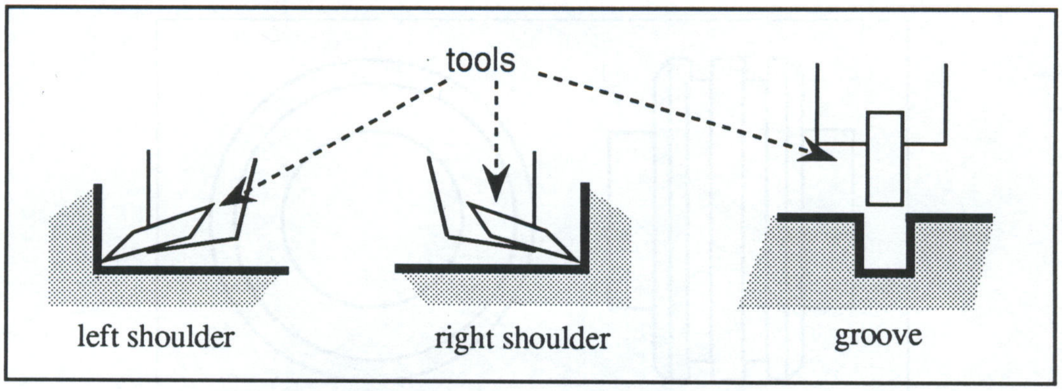

Figure 4: Production specific features

```
lshoulder(s(Rng,C)) :- ring(Rng,Z,_,Radi,-),
                       cyl(C,Z,_,Radi,-).
rshoulder(s(Rng,C)) :- ring(Rng,Z,_,Radi,+),
                       cyl(C,_,Z,Radi,-).
groove(g(Rng1,C,Rng2)) :- lshoulder(s(Rng1,C)),
                          rshoulder(s(Rng2,C)).

depth(s(Rng,C),D) :- lshoulder(s(Rng,C)),
                     ring(Rng,_,Rado,Radi,_),
                     D is Rado - Radi.
```

Figure 5: Feature Definitions

3 Forward Reasoning Horn Rules

The specification of production-specific features in horn logic as presented in Figure 5 does not give any demands on their evaluation. Reasoning can be performed using two principal directions. While forward inference proceeds from the facts in the knowledge base reasoning bottom-up to derive new facts, backward inference applies the rules in a top-down fashion starting with a query. Prolog - like most logic programming systems - evaluates clauses top-down (SLD-resolution, [Llo87]). To extract features using this strategy an iteration over all possible features is necessary. There must be a literal enumerating all the features and a second literal testing whether this feature occurs in the examined workpiece.

Interpreting horn clauses in the natural forward implication direction leads to the view of a logic program as a declarative rule system. A conclusion is a fact which is true if all the premises are satisfied. If bottom-up evaluation would be applied to the whole database many needless facts would be derived. That is why in the approach presented here forward reasoning is restricted deriving just the implications of a specified set of knowledge items. In particular forward chaining starts with a set of initial facts $p_1(x_1,...,x_{n1}),...,p_m(x_1,...,x_{nm})$. Only consequences of these facts are computed by the following procedure:

1. Start with a set of initial facts $\mathfrak{F}= \{p_1(x_1,...,x_{n1}), ...,p_m(x_1,...,x_{nm})\}$
2. Select one fact $p(x_1,...,x_n) \in \mathfrak{F}$ to be the actual fact F. Stop if there are no (more) facts.
3. Find the next potentially applicable rule: a rule C:- P_1,...P_k is triggered, if any P_i, $1 \leq i \leq k$, is unifiable with the actual fact F with substitution σ.

   If no rule is applicable go to 2.
4. Test the rule's premises. The conjunction of the remaining premises $P_1,...,P_{i-1}$, $P_{i+1},...,P_k$ is verified by backward reasoning giving a substitution $\tau \geq \sigma$.

   If the premises are not satisfiable go to 3.
5. Apply the rule: Record the instantiated conclusion $C\tau$ as a derived fact in the set of facts $\mathfrak{F}$.

   Proceed with 2. ❑

The forward reasoning strategy depends on the order in which the actual fact is selected from the list of initial and derived facts in step 2. At least two strategies are possible:

- *breadth first*: The actual fact F is kept until there is no further rule for it. Then F is set to the oldest not already expanded fact.
- *depth first*: F is set to the most recently derived fact $C\tau$ for which there are any rules to be applied.

3.1 A Forward Reasoning Meta Interpreter

To achieve forward reasoning of logic progams there are in general two approaches:

- a special forward reasoning interpreter
- a meta-interpreter written in the logic programming system itself

For deductive databases many effort has been involved in developing bottom-up reasoning strategies like naive or semi-naive evaluation [Ban86a], [Ban88]. Improvements of these approaches like Magic Sets [Ban86b] or Alexander Method [Roh86] are just applicable for goal-directed reasoning. A seperate forward-interpreter besides a conventional logic programming system for top-down evaluation, however, would make the system more complex. It requires an interface to the normal logic programming system.

A meta-interpreter on the other hand is rather inefficient, because it interprets the original program as data. But it can be made more efficient by program transformation techniques like partial evaluation (cp. [Har87]). This approach will be presented here. The meta interpreter (see Figure 6) can be divided into two parts:

(1) Selecting a fact which should serve as a trigger (step 6 of the above procedure)

(2) Finding and executing a rule (steps 3 to 5 above)

For selecting a fact a <u>d</u>epth-<u>f</u>irst <u>enum</u>eration and <u>b</u>readth-<u>f</u>irst <u>enum</u>eration strategies are presented. Asking the query `?- df-enum(p(X,Y),Result)` will successively bind `Result` to all the consequences of any instantiation of `p(X,Y)`.

Calling the predicate `forward` finds and executes one horn rule in forward direction. Its arguments are the actual fact (bound when called) and the conclusion of the applied rule (a free variable): a goal `?-forward(Fact,Conclusion)` succeeds, if `Conclusion` is a one-step derivation of `Fact`. The built-in predicate `clause` selects one rule at a time. The goal `trigger(Body,Fact,ToProve)` succeeds, if one premise in `Body` is unifiable with the actual fact. Then the variable `ToProve` is instantiated with the list of the remaining premises, which have to be proved. These premises are verified by `provelist` just making a call on them, i.e. they are satisfied by Prolog's SLD-resolution. If all the premises are satisfied, the conclusion of the rule is asserted.

To avoid loops, the conclusion is accepted only if it is not *subsumed* by any previously derived fact. This test is performed by the predicate `not_reached`, which is not listed here. The principle of the subsumption test, however, is very simple: a term $p(x_1,...,x_n)$ subsumes a term $p(y_1,...,y_n)$ if the ground term $p(x_1,...,x_n)\sigma$ - instantiated with *new* constants - is unifiable with $p(y_1,...,y_n)$. (see [Hin91]). If the conclusion is not subsumed by any previously derived fact, it is asserted as `reached` and `open_node`. A reached node is an open node, if it is not already selected as a trigger for breadth-first reasoning.

```
% Applying one rule in forward direction:
forward(Fact,Head) :- clause(Head,Body),
                      trigger(Body,Fact,ToProve),
                      provelist(ToProve),
                      retain(Head).

trigger([Fact|Rest],Fact,Rest) :- not_builtin_p(Fact).
trigger([First|Rest],Fact,[First|ProveRest]) :-
                trigger(Rest,Fact,ProveRest).

provelist([]).
provelist([First|Rest]) :- First,
                           provelist(Rest).

retain(Conclusion) :- not_reached(Conclusion),
                      asserta(reached(Conclusion)),
                      assertz(open_node(Conclusion)).

% Depth-first reasoning strategy:
df_enum(Fact,Inference) :- fc_initialize,
                           Fact,
                           df_one(Fact,Inference).

df_one(Fact,Inference) :- forward(Fact,Conclusion),
                          df_one_more(Conclusion,Inference).

df_one_more(Conclusion,Conclusion).
df_one_more(Conclusion,Next) :- df_one(Conclusion,Next).

% Breadth-first reasoning strategy:
bf_enum(Fact,Inference) :- fc_initialize,
                           Fact,
                           forward(Fact,Inference).
bf_enum(Fact,Inference) :- forward_one(Inference).

forward_one(Inference) :- open_node(Fact),
                          retract(open_node(Fact)),
                          forward(Fact,Inference).

fc_initialize :- abolish(open_node,1),
                 abolish(reached,1).
```

Figure 6: A forward reasoning meta interpreter

3.2 Forward Reasoning Feature Recognition

Now we come back to our application of feature recognition. The set of initial facts for feature recognition is the representation of a workpiece's product model. Then using the above described strategy forward reasoning seems to be an adequat strategy to recognize features. We will exemplify it with the rules of Figure 5. Starting with the facts describing the workpiece (Fig. 3) only the features occuring in the workpiece should be derived. Here is a procedure which will serve this purpose. The predicate `feature` has two arguments:

`Surfaces` is a list of surface representations our workpiece consists of (the initial facts). During execution the second argument `F` will be bound to a feature occurring in this workpiece:

```
features([Surface|_],F)  :- df-enum(Surface,F).
features([_|Surfacelist],F) :- feature(Surfacelist, F).
```

The query `?- features([ring(s6,84,180,162,-)],F)` will successively bind `F` to

```
F = lshoulder(s(s6,s7));
F = groove(g(s6,s7,s8));
F = depth(s(s6,s7),18);
no
```

4 Partially Evaluating the Meta Interpreter

Partial evaluation is a program transformation technique which, given a normal program P and a goal G, produces from P and G a specialized program P' which will evaluate G more efficiently. To increase efficiency of our meta interpreter we partially evaluate the interpreter clause `forward` wrt to a logic program. The result of this partially evaluated meta interpreter is a set of forward rules which, when evaluated by a top-down reasoning system, simulate the forward application of the original horn clauses.

The predicate `forward` has the actual fact as parameter, which is bound when `forward` is called. But it is not known in advance and thus does not help for partial evaluation. A second input to the `forward` clause is the hole logic progam itself, since the predicate `clause` is called with two unbound variables. So our partial evaluation procedure specializes `forward` wrt the whole program. This approach is called data-driven in [Cos91]. There are various rules for partial evaluation like unfolding, folding, goal replacement, and new definition [Tam84], [Gal90], but here only unfolding is needed:

Definition: Let P be a normal program. Let C be a clause in P of the form A :- $B_1,\ldots,B_i,\ldots,B_n$. Let H_1 :- $Q_{11},\ldots,Q_{1k}$

$\ldots$

H_m :- $Q_{m1},\ldots,Q_{mk}$

be clauses in P with heads unifiable with B_i yielding unifiers $\sigma_1,\ldots,\sigma_m$. The result of *unfolding* C on B_i is the set of clauses

$\{(A$:- $B_1,\ldots,B_{i-1},Q_{11},\ldots,Q_{1k},B_{i+1},\ldots,B_n)\sigma_1$,

...,

$(A$:- $B_1,\ldots,B_{i-1},Q_{m1},\ldots,Q_{mk},B_{i+1},\ldots,B_n)\sigma_m\}$

P is transformed to P' by replacing C by these clauses. ❑

Now we will explain the partial evaluation of the meta interpreter (Figure 6) wrt a program of only two clauses:

```
lshoulder(s(Rng,C))    :- ring(Rng,Left,_,Rad,-),
                          cyl(C,Left,_,Rad,-).
depth(s(Rng,C),D)  :- lshoulder(s(Rng,C)),
                      ring(Rng,_,Ro,Ri,_),
                      D is Ro - Ri.
```

First we unfold the `forward` clause on its first premise `clause(Head,Body)`. Since `clause` is a built-in predicate its definition is not given as a set of horn clauses. To be consistent with the definition of unfolding imagine that the object program is specified by a set of facts with predicate `clause`:

```
clause(lshoulder(s(Rng,C)), [ring(Rng,Left,_,Rad,-),
                             cyl(C,Left,_,Rad,-)]).
clause(depth(s(Rng,C),D), [lshoulder(s(Rng,C)),
                           ring(Rng,_,Ro,Ri,_),
                           D is Ro - Ri]).
```

The result of the first transformation step are two new clauses. Since `clause` is defined by facts, the premise `clause(Head,Body)` is eliminated and the variables `Head` and `Body` are instantiated:

```
forward(Fact,lshoulder(s(Rng,C))) :-
    trigger([ring(Rng,Left,_,Rad,-),cyl(C,Left,_,Rad,-)],
            Fact,
            ToProve),
    provelist(ToProve),
    retain(lshoulder(s(Rng,C))).

forward(Fact,depth(s(Rng,C),D)) :-
    trigger([lshoulder(s(Rng,C)),ring(Rng,_,Ro,Ri,_),D is Ro - Ri],
            Fact,
            ToProve),
    provelist(ToProve),
    retain(depth(s(Rng,C),D)).
```

By unfolding each of these rules on `trigger`, for each premise of the original object rule a new `forward` clause is generated. The variable `Fact` is bound to this premise. The variable `ToProve` is bound to the list of the remaining premises, respectively. Since `trigger` is a recursive rule unfolding will also be applied recursively. It should be noticed, however, that Prolog built-ins and negated premises cannot serve as a trigger during forward reasoning, which is tested by the premise `not_builtin_p(Fact)`. Therefore no extra forward clause is generated for them:

```
forward(ring(Rng,Left,_,Rad,-),lshoulder(s(Rng,C))) :-
    provelist([cyl(C,Left,_,Rad,-)]),
    retain(lshoulder(s(Rng,C))).
forward(cyl(C,Left,_,Rad,-),lshoulder(s(Rng,C))) :-
    provelist([ring(Rng,Left,_,Rad,-)]),
    retain(lshoulder(s(Rng,C))).

forward(lshoulder(s(Rng,C)),depth(s(Rng,C),D)) :-
    provelist(ring(Rng,_,Ro,Ri,_), D is Ro - Ri]),
    retain(depth(s(Rng,C),D)).
forward(ring(Rng,_,Ro,Ri,_),depth(s(Rng,C),D)) :-
    provelist(lshoulder(s(Rng,C)), D is Ro - Ri]),
    retain(depth(s(Rng,C),D)).
```

As a last transformation step each clause is unfolded on the call of `provelist`. The result of our partial evaluation wrt to the above rules are four clauses:

```
forward(ring(Rng,Left,_,Rad,-),lshoulder(s(Rng,C))) :-
    cyl(C,Left,_,Rad,-),
    retain(lshoulder(s(Rng,C))).
forward(cyl(C,Left,_,Rad,-),lshoulder(s(Rng,C))) :-
    ring(Rng,Left,_,Rad,-),
    retain(lshoulder(s(Rng,C))).
forward(lshoulder(s(Rng,C)),depth(s(Rng,C),D)) :-
    ring(Rng,_,Ro,Ri,_),
    D is Ro - Ri,
    retain(depth(s(Rng,C),D)).
forward(ring(Rng,_,Ro,Ri,_),depth(s(Rng,C),D)) :-
    lshoulder(s(Rng,C)),
    D is Ro - Ri,
    retain(depth(s(Rng,C),D)).
```

Thus, from the original horn rules a set of forward clauses have been computed. The meta program of Figure 6 has been turned to an object program, because the meta predicates `clause`, `trigger` and `provelist` are eliminated and the forward clauses are called directly.

By this partial evaluation procedure every rule $q(...)$:- $p_1(...),...,p_n(...)$. is translated into a sequence of forward rules following this pattern:

forward$(p_1(...),q(...))$:- $p_2(...),...,p_n(...)$, *retain*$(q(...))$.

forward$(p_2(...),q(...))$:- $p_1(...),p_3(...),...,p_n(...)$, *retain*$(q(...))$.

....

forward$(p_n(...),q(...))$:- $p_1(...),p_2(...),...,p_{n-1}(...)$, *retain*$(q(...))$.

These clauses together with the original clauses form the new bidirectional program. Because forward evaluation of a horn rule can be triggered by a fact unifying any premise of the rule, for every premise $p_1(...),...,p_n(...)$ of the original rule a forward clause is generated. This is an important distinction to Yamamoto and Tanaka's translation for production rules [Yam86], where only goal-directed forward reasoning is supported, and a rule's premises are verified only by comparing them to the fact base instead of calling SLD resolution.

5 Vertical Compilation into WAM Code

The Warren Abstract Machine [War83] is an often implemented architecture for backward reasoning of horn clauses. After horizontal transformation of a horn clause program P into a forward clause program P' (either by direct transformation or by partially evaluating the meta interpreter, cf. Section 4) the clauses of P and P' - together with the reasoning strategies M' of the meta interpreter - are compiled vertically into WAM code (see Fig. 7). But since the WAM was developed especially for backward reasoning, several improvements for forward rules are possible. They extend the WAM by a special stack area for derived facts, called *retain stack*, and a one-way unification for subsumption tests. In the following Subsections names of operations, stacks, and registers are taken from [Gab85]. The tags REF(erence), STR(ucture), and LIS(t) are borrowed from [Ait90].

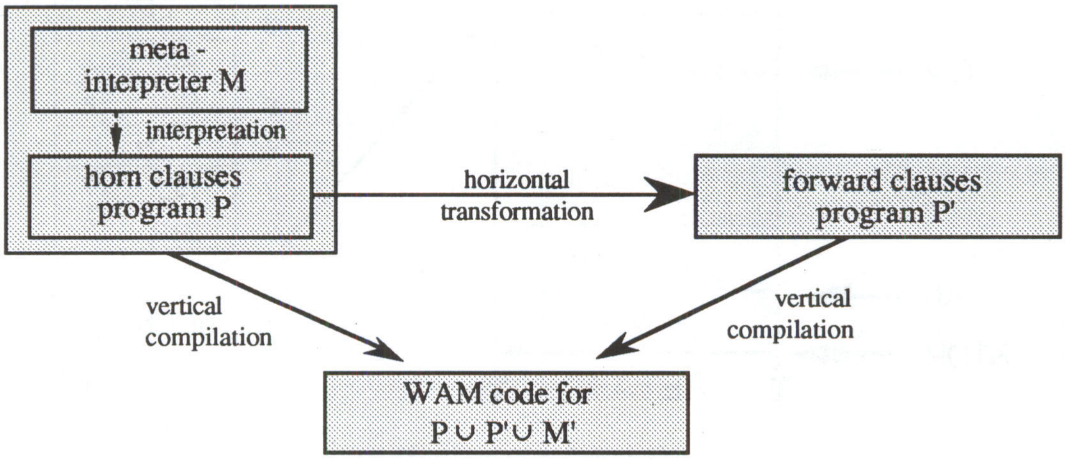

Figure 7: Two phase compilation for forward evaluation of horn clauses

5.1 The Retain Stack

Derived facts in horizontally compiled forward rules are retained by assertion with predicate `reached/1` (see Fig. 6). Such assertions are rather inefficient because program code itself is altered dynamically. Information about derived facts can be held more compactly at machine level in a special data area which will be called the *retain stack RETAIN* (see Fig. 8). The stack is organized as a list: The first REF cell points to the current entry and the following LIS cell points to the beginning of the next item. Every entry on the stack is an internal representation of a proposition derived by forward rule application. It consists of variable, constant, list and structure cells distinguished by tags. An example is given in Figure 8.

The pointer RTOP indicates the top of the retain stack. All entries of the retain stack are reached nodes. For the breadth-first strategy of forward reasoning the nodes are selected as actual fact for rule triggering in the order in which they are generated. This order is identical to the order of the nodes on the retain stack.

Therefore additional information for breadth-first forward evaluation has to be held to manage the retain stack: open nodes are derived facts, which have not already been selected as actual facts for forward chaining. Open nodes are accessed by the register ON. Every node with an address higher than ON is an open node. As soon as the node at address ON is selected, ON is increased.

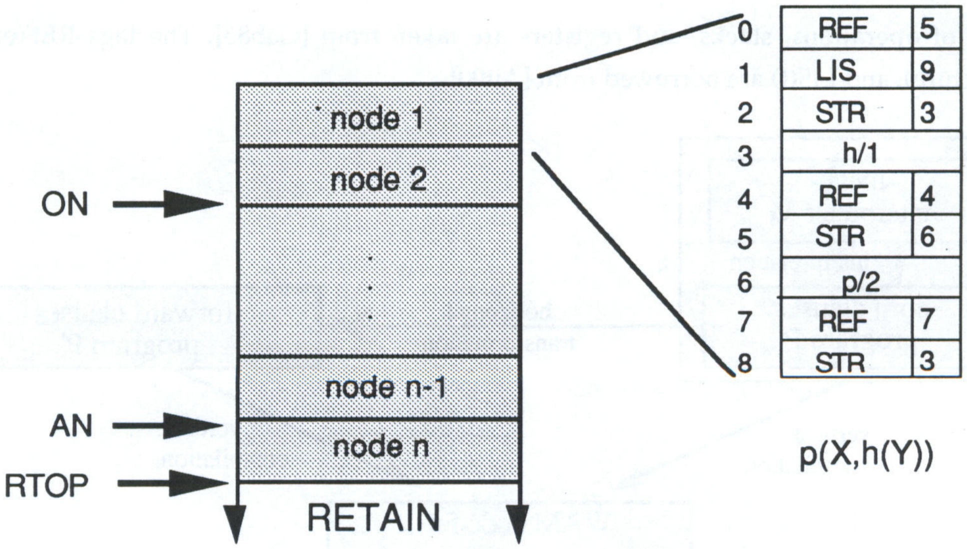

Figure 8: Retain Stack

5.2 Compiling Retain

The clause for the predicate `retain/1` (Fig. 6) in a forward clause is compiled into a sequence of WAM operations pushing its argument -- the derived fact -- onto the retain stack.

```
retain/1:  not_r_subsumed X1          % Test for subsumption
           push_fact_retain X1        % Copying the fact to RETAIN
```

To accept the new fact it must be secured that it is not subsumed by any structure already existing on the stack. A new operation `not_r_subsumed Xi` is introduced performing this test. The new fact referenced by `Xi` is matched against *every* entry on the retain stack. It calls the function `subsumes(x,y)` to test subsumption. The functions `unify(x,y)` and `subsumes(x,y)` differ only in two cases: If x and y are unbound REF cells then a new constant c_i is created and y is bound to c_i. If y is an unbound REF cell and x is a non-REF cell, the test fails, because a REF cell is not subsumed by a value. The rest of the procedure remains unchanged. Backtracking occurs, if subsumption of the derived fact with any previously derived fact succeeds.

If subsumption fails no backtracking occurs and the new fact is pushed onto the retain stack by the operation `push_fact_retain Xi`. The values on the retain stack are "more persistent" than values on the global stack or the local stack. While values on the local and global stack may be destroyed by backtracking, derived facts must survive for the whole forward inference chain. Because of this no reference from the retain stack to any other memory cell is permitted. This is why a derived fact is *copied* onto RETAIN. Before pushing variables are dereferenced. If the dereferenced value is not an unbound variable cell it is *copied* onto the retain stack and dereferencing is performed recursively for every subvariable in the functor structure. Otherwise, for an unbound variable, a new REF cell is pushed onto the retain stack referring to itself. Finally, RTOP is increased, completing the retain operation.

5.3 Compiled Strategies

The clauses representing different reasoning strategies refer to structures residing on the retain stack. So their compiled version needs some modifications compared to a straightforward compilation. These modifications are rather obvious, but since the retain stack is an extension to the conventional WAM, novel operations are introduced.

- Performing forward chaining initialization resets the pointers ON and RTOP to the bottom address of the retain stack: `fc_initialize`

  Accessing an open node is implemented by getting the structure at stack position ON, which is performed by a call to `open_node Xi`. Then `Xi` refers to the actual open node.

- After accessing an open node ON thas to be increased to point to the successive stack item by executing `next_open_node`

- Breadth-first reasoning stops, if there exists no further open node. This is equivalent to the state when ON = RTOP

6 Future Work and Conclusions

An approach for combined forward and backward reasoning of horn rules has been presented. The whole system is embedded in a logic programming environment. A common horn rule set is used for both reasoning directions. By partially evaluating the meta interpreter the original horn clause program is transformed into clauses corresponding to one step of forward reasoning. For these forward clauses a compilation into an extended Warren Abstract Machine (WAM [War83]) has been developed. Instead of simply asserting derived facts, a special stack area - called retain stack - extends the WAM. Also the subsumption test of a new fact with previously derived ones is made more efficient by variations of the WAM's unification operations. Accomodation of the WAM to forward reasoning is one of our current research activities.

The presented plain control strategy is induced by the SLD-resolution procedure of logic programming. It is very similar to the Prolog implementation of the KORE/IE production system [Shi88]. Forward rules are selected for execution in a strictly sequential manner. Also a rule's premises are tested sequentially. But implementation methods for production systems like TREAT [Mir87] or Rete [For82] algorithm are not appropriate, since premises are proved by backward reasoning in our approach. Nevertheless, besides breadth-first and depth-first strategies, more sophisticated control strategies are conceivable, especially in larger applications, where rules reflect an expert's heuristics. In the presented approach the strategies are themselves are represented as horn clauses, so they can be adapted for the specific application. This flexibility does not cost too much overhead, since by partial evalution of the meta interpreter the border between object and meta level has been blurred. The access to the object level by a call of the `clause` predicate is abandoned from the meta interpreter. To represent meta information [Pet88] promotes to control rule firing at instance level taking into account variable instantiations. Looking for the appropriate level of rule firing control and its integration into the meta interpreter is one matter of future research. It will be influenced strongly by our application of production planning.

References

[Ait90] Ait-Kaci, H. The WAM: A (Real) Tutorial. Report 5, Digital, Paris Research Laboratory, January, 1990.

[Ban86a] Bancilhon, F. and Ramakrishnan, R. An Amateur's Introduction to Recursive Query Processing Strategies. In *Proceedings of the ACM SIGMOD Conference*, ACM, 1986, pp. 16-52.

[Ban86b] Bancilhon, F., Maier, D., Sagiv, Y., and Ullman, J.D. Magic Sets and Other Strange Ways to Implement Logic Programs. In *Proceedings 5th ACM SIGMOD-SIGACT Symposium on Principles of Database Systems*, ACM, 1986, pp. 1-15.

[Ban88] Bancilhon, F. and Ramakrishnan, R. Performance Evaluation of Data Intensive Logic Programs. In *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann Publishers, Inc., Minker, J., pp. 441-517, Los Altos, CA, 1988.

[Ber90] Bernardi, A., Klauck, C., and Legleitner, R. Formalismus zur Repräsentation von Geometrie- und Technologieinformationen als Teil eines Wissensbasierten Produktmodells. Dokument D-90-05, DFKI GmbH, 1990.

[Cha87] Chan, D., Dufresne, P., and Enders, R. Report on PHOCUS. Technical Report TR-LP-21-02, ECRC, Arabellastraße 17 D8 München 81, April, 1987.

[Cos91] Cosmadopoulos, Y., Sergot, M., and Southwick,R.W. . In *PDK'91 - International Workshop on Processing Declarative Knowledge*, 1991.

[Fin89] Finin, T., Fritzson, R., and Matuszek, D. Adding Forward Chaining and Truth Maintenance to Prolog. In *Artificial Intelligence Applications Conference*, IEEE, Miami, March 1989, pp. 123-130.

[For82] Forgy, C.L. Rete: A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem. Artificial Intelligence *19*(1982), pp. 17-37.

[Gab85] Gabriel, J., Lindholm, T., Lusk, E.L., and Overbeek, R.A. A Tutorial on the Warren Abstract Machine for Computational Logic. Report ANL-84-84, Argonne National Laboratory, Argonne, Illinois 60439, June, 1985.

[Gal90] Gallagher, J., *Program Analysis and Transformation*, Logic Programming Summer School LPSS, 1990.

[Har87] van Harmelen, F. Improving the efficiency of meta-level reasoning. DAI Discussion Paper No. 40, University of Edinburgh, Proposal for a Ph.D. thesis, 1987.

[Hin91] Hinkelmann, K. Bidirectional Reasoning of Horn Clause Programs: Transformation and Compilation. Technical Memo TM-91-02, DFKI GmbH, January, 1991.

[Llo87] Lloyd, J.W. *Foundations of Logic Programming*, Springer-Verlag, Berlin, Heidelberg, New York (1987).

[Mir87] Miranker, D.P. TREAT: A Better Match Algorithm for AI Production Systems. In *Proc. of AAAI-87*, Philadelphia, PA, 1987, pp. 42-47.

[Mor81] Morris, P. A Forward Chaining Problem Solver. Logic Programming Newsletter 2(Autumn 1981), pp. 6-7.

[Naq89] Naqvi, S.A. and Tsur, S. *A Logical Language for Data and Knowledge Bases*, W.H. Freeman (1989).

[Pet88] Petrie, C.J. and Huhns, M.N. Controlling Forward Rule Instances. MCC Technical Report ACA-AI-012-88, Microelectronics and Computer Technology Corporation, 3500 West Balcones Center Drive, Austin, TX, January, 1988.

[Roh86] Rohmer, J., Lescoeur, R., and Kerisit, J.M. The Alexander Method - A Technique for The Processing of Recursive Axioms in Deductive Databases. New Generation Computing (1986), pp. 273-285.

[Shi88] Shintani, T. A Fast Prolog-Based Production System KORE/IE. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, 1988, pp. 26-41.

[Tam84] Tamaki, H. and Sato, T. Unfold/Fold Transformations of Logic Programs. In *Proceedings of the Second International Conference on Logic Programming*, Uppsala, 1984, pp. 127-138.

[Tre87] Treitel, R. and Genesereth, M.R. Choosing Directions for Rules. Journal of Automated Reasoning *3*(1987), pp. 395-431.

[War83] Warren, D.H.D. An Abstract Prolog Instruction Set. Technical Note 309, SRI International, Menlo Park, CA, October, 1983.

[Yam86] Yamamoto, A. and Tanaka, H. Translating Production Rules into a Forward Reasoning Prolog Program. New Generation Computing *4*(1986), pp. 97-105.

Diplomarbeit

Christian Falter: Eine abstrakte Machine für die Vorwärtsverarbeitung von Hornklauseln

Stuatus: Konzeptionsphase

Projektarbeiten

Thomas Labisch: Implementierung einer semi-naiven Strategie für bottom-up Evaluierung von RELFUN Hronklauseln

Status: abgeschlossen seit Juni 1991

Thomas Oltzen: Term Subsumption in der WAM

Status: Beim Aufschreiben

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
6750 Kaiserslautern
FRG

DFKI Publikationen

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

DFKI Publications

The following DFKI publications or the list of currently available publications can be ordered from the above address.
The reports are distributed free of charge except if otherwise indicated.

DFKI Research Reports

RR-90-03
Andreas Dengel, Nelson M. Mattos: Integration of Document Representation, Processing and Management
18 pages

RR-90-04
Bernhard Hollunder, Werner Nutt: Subsumption Algorithms for Concept Languages
34 pages

RR-90-05
Franz Baader: A Formal Definition for the Expressive Power of Knowledge Representation Languages
22 pages

RR-90-06
Bernhard Hollunder: Hybrid Inferences in KL-ONE-based Knowledge Representation Systems
21 pages

RR-90-07
Elisabeth André, Thomas Rist: Wissensbasierte Informationspräsentation:
Zwei Beiträge zum Fachgespräch Graphik und KI:
1. Ein planbasierter Ansatz zur Synthese illustrierter Dokumente
2. Wissensbasierte Perspektivenwahl für die automatische Erzeugung von 3D-Objektdarstellungen

24 Seiten

RR-90-08
Andreas Dengel: A Step Towards Understanding Paper Documents
25 pages

RR-90-09
Susanne Biundo: Plan Generation Using a Method of Deductive Program Synthesis
17 pages

RR-90-10
Franz Baader, Hans-Jürgen Bürckert, Bernhard Hollunder, Werner Nutt, Jörg H. Siekmann: Concept Logics
26 pages

RR-90-11
Elisabeth André, Thomas Rist: Towards a Plan-Based Synthesis of Illustrated Documents
14 pages

RR-90-12
Harold Boley: Declarative Operations on Nets
43 pages

RR-90-13
Franz Baader: Augmenting Concept Languages by Transitive Closure of Roles: An Alternative to Terminological Cycles
40 pages

RR-90-14
Franz Schmalhofer, Otto Kühn, Gabriele Schmidt: Integrated Knowledge Acquisition from Text, Previously Solved Cases, and Expert Memories
20 pages

RR-90-15
Harald Trost: The Application of Two-level Morphology to Non-concatenative German Morphology
13 pages

RR-90-16
Franz Baader, Werner Nutt: Adding Homomorphisms to Commutative/Monoidal Theories, or: How Algebra Can Help in Equational Unification
25 pages

RR-90-17
Stephan Busemann: Generalisierte Phasenstrukturgrammatiken und ihre Verwendung zur maschinellen Sprachverarbeitung
114 Seiten

RR-91-01
Franz Baader, Hans-Jürgen Bürckert, Bernhard Nebel, Werner Nutt, Gert Smolka: On the Expressivity of Feature Logics with Negation, Functional Uncertainty, and Sort Equations
20 pages

RR-91-02
Francesco Donini, Bernhard Hollunder, Maurizio Lenzerini, Alberto Marchetti Spaccamela, Daniele Nardi, Werner Nutt: The Complexity of Existential Quantification in Concept Languages
22 pages

RR-91-03
B.Hollunder, Franz Baader: Qualifying Number Restrictions in Concept Languages
34 pages

RR-91-04
Harald Trost: X2MORF: A Morphological Component Based on Augmented Two-Level Morphology
19 pages

RR-91-05
Wolfgang Wahlster, Elisabeth André, Winfried Graf, Thomas Rist: Designing Illustrated Texts: How Language Production is Influenced by Graphics Generation.
17 pages

RR-91-06
Elisabeth André, Thomas Rist: Synthesizing Illustrated Documents A Plan-Based Approach
11 pages

RR-91-07
Günter Neumann, Wolfgang Finkler: A Head-Driven Approach to Incremental and Parallel Generation of Syntactic Structures
13 pages

RR-91-08
Wolfgang Wahlster, Elisabeth André, Som Bandyopadhyay, Winfried Graf, Thomas Rist: WIP: The Coordinated Generation of Multimodal Presentations from a Common Representation
23 pages

RR-91-09
Hans-Jürgen Bürckert, Jürgen Müller, Achim Schupeta: RATMAN and its Relation to Other Multi-Agent Testbeds
31 pages

RR-91-10
Franz Baader, Philipp Hanschke: A Scheme for Integrating Concrete Domains into Concept Languages
31 pages

RR-91-11
Bernhard Nebel: Belief Revision and Default Reasoning: Syntax-Based Approaches
37 pages

RR-91-12
J.Mark Gawron, John Nerbonne, Stanley Peters: The Absorption Principle and E-Type Anaphora
33 pages

RR-91-13
Gert Smolka: Residuation and Guarded Rules for Constraint Logic Programming
17 pages

RR-91-14
Peter Breuer, Jürgen Müller: A Two Level Representation for Spatial Relations, Part I
27 pages

RR-91-15
Bernhard Nebel, Gert Smolka: Attributive Description Formalisms ... and the Rest of the World
20 pages

RR-91-16
Stephan Busemann: Using Pattern-Action Rules for the Generation of GPSG Structures from Separate Semantic Representations
18 pages

RR-91-17
Andreas Dengel, Nelson M. Mattos: The Use of Abstraction Concepts for Representing and Structuring Documents
17 pages

RR-91-18
John Nerbonne, Klaus Netter, Abdel Kader Diagne, Ludwig Dickmann, Judith Klein: A Diagnostic Tool for German Syntax
20 pages

RR-91-19
Munindar P. Singh: On the Commitments and Precommitments of Limited Agents
15 pages

RR-91-20
Christoph Klauck, Ansgar Bernardi, Ralf Legleitner FEAT-Rep: Representing Features in CAD/CAM
48 pages

RR-91-21
Klaus Netter: Clause Union and Verb Raising Phenomena in German
38 pages

RR-91-22
Andreas Dengel: Self-Adapting Structuring and Representation of Space
27 pages

RR-91-23
Michael Richter, Ansgar Bernardi, Christoph Klauck, Ralf Legleitner: Akquisition und Repräsentation von technischem Wissen für Planungsaufgaben im Bereich der Fertigungstechnik
24 Seiten

RR-91-24
Jochen Heinsohn: A Hybrid Approach for Modeling Uncertainty in Terminological Logics
22 pages

RR-91-25
Karin Harbusch, Wolfgang Finkler, Anne Schauder: Incremental Syntax Generation with Tree Adjoining Grammars
16 pages

RR-91-26
M. Bauer, S. Biundo, D. Dengler, M. Hecking, J. Koehler, G. Merziger:
Integrated Plan Generation and Recognition
- A Logic-Based Approach -
17 pages

RR-91-27
A. Bernardi, H. Boley, Ph. Hanschke, K. Hinkelmann, Ch. Klauck, O. Kühn, R. Legleitner, M. Meyer, M. M. Richter, F. Schmalhofer, G. Schmidt, W. Sommer:
ARC-TEC: Acquisition, Representation and Compilation of Technical Knowledge
18 pages

RR-91-28
Rolf Backofen, Harald Trost, Hans Uszkoreit:
Linking Typed Feature Formalisms and Terminological Knowledge Representation Languages in Natural Language Front-Ends
11 pages

RR-91-29
Hans Uszkoreit: Strategies for Adding Control Information to Declarative Grammars
17 pages

RR-91-30
Dan Flickinger, John Nerbonne:
Inheritance and Complementation: A Case Study of *Easy* Adjectives and Related Nouns
39pages

RR-91-31
H.-U. Krieger , J. Nerbonne:
Feature-Based Inheritance Networks for Computational Lexicons
11 pages

RR-91-32
Rolf Backofen, Lutz Euler, Günther Görz:
Towards the Integration of Functions, Relations and Types in an AI Programming Language
14 pages

DFKI Technical Memos

TM-90-03
Franz Baader, Bernhard Hollunder: KRIS: Knowledge Representation and Inference System -System Description-
15 pages

TM-90-04
Franz Baader, Hans-Jürgen Bürckert, Jochen Heinsohn, Bernhard Hollunder, Jürgen Müller, Bernhard Nebel, Werner Nutt, Hans-Jürgen Profitlich: Terminological Knowledge Representation: A Proposal for a Terminological Logic
7 pages

TM-91-01
Jana Köhler: Approaches to the Reuse of Plan Schemata in Planning Formalisms
52 pages

TM-91-02
Knut Hinkelmann: Bidirectional Reasoning of Horn Clause Programs: Transformation and Compilation
20 pages

TM-91-03
Otto Kühn, Marc Linster, Gabriele Schmidt:
Clamping, COKAM, KADS, and OMOS:
The Construction and Operationalization of a KADS Conceptual Model
20 pages

TM-91-04
Harold Boley:
A sampler of Relational/Functional Definitions
12 pages

TM-91-05
Jay C. Weber, Andreas Dengel , Rainer Bleisinger:
Theoretical Consideration of Goal Recognition Aspects for Understanding Information in Business Letters
10 pages

TM-91-06
Johannes Stein: Aspects of Cooperating Agents
22 pages

TM-91-08
Munindar P. Singh: Social and Psychological Commitments in Multiagent Systems
11 pages

TM-91-09
Munindar P. Singh: On the Semantics of Protocols Among Distributed Intelligent Agents
18 pages

TM-91-10
Béla Buschauer, Peter Poller, Anne Schauder, Karin Harbusch: Tree Adjoining Grammars mit Unifikation
149 pages

TM-91-11
Peter Wazinski: Generating Spatial Descriptions for Cross-modal References
21 pages

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

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

DFKI Documents

D-90-05
Ansgar Bernardi, Christoph Klauck, Ralf Legleitner: Formalismus zur Repräsentation von Geo-metrie- und Technologieinformationen als Teil eines Wissensbasierten Produktmodells
66 Seiten

D-90-06
Andreas Becker: The Window Tool Kit
66 Seiten

D-91-01
Werner Stein , Michael Sintek: Relfun/X - An Experimental Prolog Implementation of Relfun
48 pages

D-91-03
Harold Boley, Klaus Elsbernd, Hans-Günther Hein, Thomas Krause: RFM Manual: Compiling RELFUN into the Relational/Functional Machine
43 pages

D-91-04
DFKI Wissenschaftlich-Technischer Jahresbericht 1990
93 Seiten

D-91-06
Gerd Kamp: Entwurf, vergleichende Beschreibung und Integration eines Arbeitsplanerstellungssystems für Drehteile
130 Seiten

D-91-07
Ansgar Bernardi, Christoph Klauck, Ralf Legleitner
TEC-REP: Repräsentation von Geometrie- und Technologieinformationen
70 Seiten

D-91-08
Thomas Krause: Globale Datenflußanalyse und horizontale Compilation der relational-funktionalen Sprache RELFUN
137 pages

D-91-09
David Powers and Lary Reeker (Eds):
Proceedings MLNLO'91 - Machine Learning of Natural Language and Ontology
211 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-91-10
Donald R. Steiner, Jürgen Müller (Eds.):
MAAMAW'91: Pre-Proceedings of the 3rd European Workshop on „Modeling Autonomous Agents and Multi-Agent Worlds"
246 pages
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

D-91-11
*Thilo C. Horstmann:*Distributed Truth Maintenance
61 pages

D-91-12
Bernd Bachmann:
$\mathrm{H}^{\mathrm{iera}}\mathrm{C}_{\mathrm{on}}$ - a Knowledge Representation System with Typed Hierarchies and Constraints
75 pages

D-91-13
International Workshop on Terminological Logics
Organizers: Bernhard Nebel, Christof Peltason, Kai von Luck
131 pages

D-91-14
Erich Achilles, Bernhard Hollunder, Armin Laux, Jörg-Peter Mohren: KRIS : Knowledge Representation and Inference System
- Benutzerhandbuch -
28 Seiten

Forward Logic Evaluation: Developing a Compiler from a Partially Evaluated Meta Interpreter

Knut Hinkelmann

TM-91-13
Technical Memo