Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research
Report
RR-91-26

Integrated Plan Generation and Recognition

- A Logic-Based Approach -

M. Bauer, S. Biundo, D. Dengler, M. Hecking,
J. Koehler, G. Merziger

August 1991

Deutsches Forschungszentrum für Künstliche Intelligenz
GmbH

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

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

Deutsches Forschungszentrum für Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, SEMA Group, and Siemens. Research projects conducted at the DFKI are funded by the German Ministry for Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct *systems with technical knowledge and common sense* which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- ❑ Intelligent Engineering Systems
- ❑ Intelligent User Interfaces
- ❑ Computer Linguistics
- ❑ Programming Systems
- ❑ Deduction and Multiagent Systems
- ❑ Document Analysis and Office Automation.

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

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

Dr. Dr. D. Ruland
Director

Integrated Plan Generation and Recognition
- A Logic-Based Approach -

M. Bauer, S. Biundo, D. Dengler, M. Hecking, J. Koehler, G. Merziger

DFKI-RR-91-26

The paper will be published in the Proceedings of the '4. Internationaler GI-Kongreß Wissensbasierte Systeme', Springer-Verlag, 1991.

This work has been supported by a grant from The Federal Ministry for Research and Technology (FKZ ITW-9000 8).

Integrated Plan Generation and Recognition - A Logic-Based Approach -

M. Bauer, S. Biundo, D. Dengler, M. Hecking, J. Koehler, G. Merziger
Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Stuhlsatzenhausweg 3
W-6600 Saarbrücken 11
e-mail: <last name>@dfki.uni-sb.de

Abstract

The work we present in this paper is settled within the field of intelligent help systems. Intelligent help systems aim at supporting users of application systems by the achievements of qualified experts. In order to provide such qualified support our approach is based on the integration of plan generation and plan recognition components. Plan recognition in this context serves to identify the users goals and so forms the basis for an active user support. The planning component dynamically generates plans which are proposed for the user to reach her goal. We introduce a logic-based approach where plan generation and plan recognition is done on a common logical basis and both components work in some kind of cross-talk.

Contents

1 Introduction

Intelligent help systems aim at supporting users of application systems by the achievements of qualified experts, e.g., cf. [NWWng], [HKN+88]. This support can be considerably improved if help systems are provided with plan recognition and plan generation components. In this context *Plan recognition* serves to identify the users goals and thus forms the basis for providing active help (cf. [Fin83], [DGH87]). *Plan generation* is an essential prerequisite for supporting the user with plans to reach his goals (cf. [Lur88], [Bre90], [Heg91]).

Whereas previous approaches were working with separated plan recognition and plan generation components it is our aim to realize some kind of *cross-talk* between both: Plan recognition and plan generation components work in integrated mutual cooperation. We distinguish between three different kinds of cross-talk which will be introduced in section 2.

Plan recognition as well as planning will be done in a deductive way and will be based on a common logical formalism. A brief sketch of the underlying logic will be given in section 3. Finally, sections 4 and 5 show by means of short examples how plan recognition, plan generation and *plan reuse* can be realized in an appropriate deductive framework based on this logic.

2 Architecture and Cross-Talk Modes

We intend to implement a system called PHI[1] (see figure 1) that constitutes the kernel of an active intelligent help system. An *Application Interface* provides as input observed actions and goals. On the other hand, it receives recognized, generated, and optimal plans.

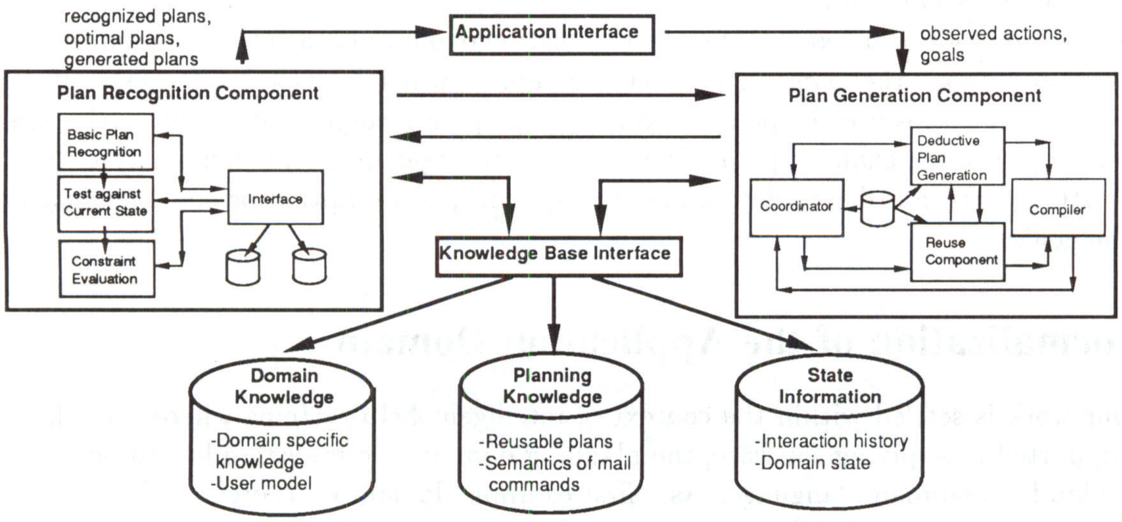

Figure 1: The PHI Architecture

[1]The PHI project is supported by the BMFT (Bonn) under Grant ITW 9000 8.

Plan recognition and plan generation use a common knowledge base containing planning knowledge and state information as well as user and domain specific knowledge.
One main point of interest in our research concentrates on realizing the *cross-talk* between plan recognition and plan generation. This cross-talk in particular presupposes a common logical representation formalism for all kinds of knowledge. We distinguish between three different cross-talk modes:

- **First cross-talk mode**
  The plan recognizer works with plans produced by the plan generator. The basis for generation consists of already *observed actions* and *standard assumptions* about goals, that typically occur in the domain considered. The user-specific characteristics as designated in the *user model* also play a role in the generation process. If a set of hypothetical plans has been produced by the plan generator, it is made available to the plan recognizer. If the new observed actions cannot be mapped on the hypothetical plans or standard assumptions change, the plan generator is activated again and the plan recognizer is supplied with a set of hypotheses that covers the increased number of observed actions. The plan recognition process is successfully completed when a plan has been found which connects the observed actions in such a way that they lead to one of the assumed goals.

- **Second cross-talk mode**
  The main topics in this mode are the *identification of suboptimal plans* and the *generation of optimal plans.* The plan recognizer employs given domain-specific suboptimal plans which typically show up. If after a few observed actions a suboptimal plan is assumed by the plan recognizer, these actions and the goal corresponding to the plan are given to the plan generator. The plan generator then produces one or more optimal plans for this goal and provides them to the application system.

- **Third cross-talk mode**
  The third mode is an example for the application of our approach to plan monitoring. For a given goal, the plan generator creates a plan, which is passed to the user. If the user does not execute the plan as expected, the plan recognizer determines the goal pursued with the changed plan. The plan monitoring component then analyses both existing goals in order to determine inconsistency, subsumption or compatibility of the goals.

3 Formalization of the Application Domain

Since our work is settled within the context of intelligent help systems where users have to be supported in applying software, the planning domain of our system has to be based on some kind of command language. As a first example domain we therefore use a subset of the operating system UNIX, namely the UNIX mail system, which is of manageable size and additionally provides a great variety in building and recognizing plans.
The deductive plan recognition and plan generation formalisms are based on a many-sorted modal temporal logic (cf. [RP86], [Krö87], [Hal89]). Besides the temporal operators

$\bigcirc$(*next*), $\Diamond$ (*sometimes*), $\Box$ (*always*), and ; (chop), *assignments* and *control structures* are provided following imperative programming languages . The idea behind using those programming language constructs is that plans and in particular *abstract* plans which subsume a variety of concrete ones can in general be viewed as programs (cf. [MW86], [Bib86]).
The mail commands (e.g. *read*, *delete*, *quit*) are axiomatized as basic actions in the following form:

$$precondition \wedge EX(command) \rightarrow \bigcirc effect.$$

The *read*-command, for example, is defined according to its effect of changing the flag of a current mail object:

$$\forall x : mail\_object$$
$$[\neg flag(x) = \text{"}d\text{"} \wedge EX(read(x)) \rightarrow \bigcirc flag(x) = \text{"}r\text{"}],$$

where $EX(c)$ means: "Execution of command c".
The control structures are the following:

- ; (chop operator)
  The formula $\phi;\psi$ means that ϕ holds *before* ψ thus denoting the sequential composition of both subformulas.

- **if** ... **then** ... **else** (conditional)
  The formula **if** ϕ **then** ψ_1 **else** ψ_2 stands for $[\phi \rightarrow \psi_1] \wedge [\neg\phi \rightarrow \psi_2]$.

- **while** ... **do** ... **od** (while - loop)
  The **while**-operator is axiomatized according to
  $$[\textbf{while } \phi \textbf{ do } \psi \textbf{ od}; \alpha] \leftrightarrow [\textbf{if } \phi \textbf{ then } \psi; [\textbf{while } \phi \textbf{ do } \psi \textbf{ od}; \alpha] \textbf{ else } \alpha].$$

Certain formulas of our temporal logic are viewed as *plans*. Those *plan formulas* are

- all formulas $EX(c)$, where c is a term of type *command_name*;
- assignments of form $a := t$, where a is a *local variable* and t is a term;
- all formulas $\phi;\psi$ where ϕ and ψ are plan formulas;
- all formulas **if** ϕ **then** ψ_1 **else** ψ_2, where ψ_1 and ψ_2 are plan formulas and ϕ is a formula not containing any temporal operator or basic plan formula;
- all formulas **while** ϕ **do** ψ **od** ; α, where ψ and α are plan formulas and ϕ is a formula not containing any temporal operator or basic plan formula;
- all formulas $\Diamond\,\phi$, where ϕ is a plan formula;
- all formulas $\phi \vee \psi$ where ϕ and ψ are plan formulas.

4 Plan Recognition

The plan recognition component differs in two aspects from the systems mentioned in e.g.[FLS85], [SC85], [Hec87], [HKN+88]: It works deductively and communicates with a plan generation component in different cross-talk modes (cf. figure 1). It will work incrementally and non-monotonically (first approaches are described in [Hec91] and [Mer91]). During the recognition process, which is described below in more detail, the following functionality must be realized:

Basic Plan Recognition: Identify those plans which contain the observed action.

Test against Current State: Test whether the observed action fits into the time structure.

Constraint Evaluation: Test whether all constraints are fulfilled.

Recognized plans, plan hypotheses, and the recognition history are stored in the knowledge base to be used later.

The plan recognition process is an

> *iterative process for selecting plan hypotheses which account for the observed actions.*

Before describing the plan recognition procedure, we first consider some properties of its input: the plan hypotheses and the observed actions.
In general, the plan hypotheses are no concrete action sequences, but contain several degrees of abstraction:

1. The commands may not be completely instantiated, i.e., they contain formal parameters instead of an actual argument.

2. The temporal structure of the plan hypotheses may be ambiguous. They may contain subformulas like $\Diamond EX(a)$ which means "execute command *a sometimes* within the duration of the plan hypothesis currently considered."

3. Nondeterministic choices like $EX(a) \vee EX(b)$ can appear which mean "execute command *a or* command *b*".

4. Besides actual domain commands, a plan hypothesis may also contain *abstract commands* like *readmails* (cf. example below).

Observed single actions are described by formulas like $EX(a)$, whereas action sequences are expressed by $EX(a_1); EX(a_2); ...; EX(a_n)$.
At the beginning of the process a set of possible plan hypotheses $\triangle_0$ is provided by the plan generation component. Together with the observed action $EX(Command_1)$ the plan recognizer determines in the next state the set of hypotheses $\triangle_1$ so that every member of $\triangle_1$ contains the observed action, or more formally:

$$\triangle_0 \cup \{EX(Command_1)\} \vdash_{PR} \triangle_1$$

($\vdash_{PR}$ means that plan recognition specific deductions are used). If a sequence of observations $EX(Command_1)$,..., $EX(Command_n)$ must be processed, the recognition process can be abstractly described as follows ($\circ^i$ means that the command is executed in the i-th state):

$$\triangle_0 \cup \{EX(Command_1)\} \vdash_{PR} \triangle_1$$
$$\vdots$$
$$\triangle_i \cup \{\circ^i EX(Command_{i+1})\} \vdash_{PR} \triangle_{i+1}$$
$$\vdots$$

During this iterative process:

- completely recognized plans can be deleted from $\triangle_i$, and
- if no hypothesis can explain the observed actions, an adapted set $\triangle_0$ of generated possible hypotheses must be delivered by the generation component.

Assume that until now an action sequence $\Phi = EX(a_1);...;EX(a_{n-1})$ was observed and that each of the plan hypotheses in $\triangle_{n-1} = \{P_1, P_2, ..., P_m\}$ could explain those observations. Let a_n be the next observed action. Then the plan recognition procedure works as follows: It selects those hypotheses for which the formula $\Phi; EX(a_n)$ constitutes a valid starting sequence of actions. For each such P_i this means:

(a) The formula $\Phi' = EX(a_1);...;EX(a_n)$ contains the same actions as P_i wherever the hypothesis demands these actions to be executed at a certain time and a parameter binding compatible to the one demanded in P_i.

(b) There is a suitable concrete domain command in Φ' wherever the plan hypothesis contains an abstract command.

(c) For every nondeterministic choice in P_i, Φ' contains exactly one of the alternative actions.

(d) Φ' induces a temporal structure compatible with the initial part of P_i.

The process taking place at each step of the recognition process can be described for each plan hypothesis $P \in \triangle_i$ as follows: Let $EX(Command_i)$ be the formula describing the last observed action. Then the plan recognizer tries to derive a new hypothesis P' which will become a member of $\triangle_{i+1}$:

$$P \wedge \circ^i EX(Command_i) \ \vdash_{PR} \ P^1$$

where P and P^1 are related in the following way: There is a way to split P into an initial segment $Init_P$, a terminating segment $Rest_P$ and a segment Mid_P of commands describing just that part of P currently considered. Informally, $Init_P$ is that part of the hypothesis already recognized. It exactly corresponds to the sequence of observed actions of former recognition steps, whereas $Rest_P$ is that part which will be considered in the next step if the current recognition step is successful, i.e., if Mid_P and $EX(Command_i)$ fulfill the requirements (a) – (d) listed above. Thus we have

$$P = Init_P; Mid_P; Rest_P$$

$$P^1 = Init_P; EX(command_i); Rest_{P^1}$$

where $Rest_{P^1}$ results from $Rest_P$ by substituting formal parameters bound in the last step. If $Rest_{P^1}$ becomes empty, the plan corresponding to this hypothesis was successfully recognized.

The potential of the plan recognition capabilities with a temporal logic described abstractly above is explained through an example. The following *plans* are used as hypotheses:

$$\begin{aligned}\forall arg1 : mbox, arg2 : integer\ [EX(Plan1(arg1, arg2)) \leftrightarrow\ & \\ & EX(folder(arg1)); \\ & \Diamond EX(showmails()); \\ & \Diamond EX(readmails(arg2)); \\ & EX(d(arg2)); \\ & EX(folder(['\#'])) \vee EX(quit())]\end{aligned}$$

$$\begin{aligned}\forall arg1 : mbox, arg2 : integer\ [EX(Plan2(arg1, arg2)) \leftrightarrow\ & \\ & EX(folder(arg1)); \\ & \Diamond EX(showmails()); \\ & \Diamond EX(readmails(arg2)); \\ & EX(quit())]\end{aligned}$$

The definition of the *abstractions between commands* is expressed by:

$$\begin{aligned} EX(f(['*'])) &\rightarrow EX(showmails()) \\ EX(h([])) &\rightarrow EX(showmails()) \end{aligned}$$

$$\begin{aligned} \forall x : integer.EX(read(x)) &\rightarrow EX(readmails(x)) \\ \forall x : integer.EX(next(x)) &\rightarrow EX(readmails(x)) \\ &\vdots \end{aligned}$$

The following sequence of commands is observed:

$$\begin{aligned} & EX(folder([UnansweredMails])) \\ & EX(h([])) \\ & EX(read([7])) \\ & EX(d([7])) \\ & EX(folder(['\#'])) \end{aligned}$$

Assume that the initial set $\triangle_0$ of plan hypotheses contains *Plan1* and *Plan2*. The first observation $EX(folder([UnansweredMails]))$ fulfills the constraint (a) for both hypotheses, (b) – (d) need not be considered. Thus, after the first step we have:

$$\begin{aligned}
\triangle_1 &= \{Plan1^1, Plan2^1\} \\
Init_{Plan1^1} &= Init_{Plan2^1} = EX(folder([UnansweredMails])) \\
Mid_{Plan1^1} &= Mid_{Plan2^1} = \Diamond\, EX(showmails()) \\
Rest_{Plan1^1} &= \Diamond\, EX(readmails(arg2))\,; ...; EX(folder(['\#'])) \vee EX(quit()) \\
Rest_{Plan2^1} &= \Diamond\, EX(readmails(arg2))\,; EX(quit())
\end{aligned}$$

The description of the second observed command is $\circ EX(h([\,]))$. None of the hypotheses in $\triangle_1$ contains a concrete action in its *Mid* part, but the abstract command *showmails*. The command abstraction axioms tell us that *h([ ])* is a suitable instance for this command, so that (b) holds for $Plan1^1$ and $Plan2^1$. While (c) plays no role, we see that the temporal structure of $\circ EX(h([\,]))$ is compatible with those of Mid_{Plan1^1} and Mid_{Plan2^1}, and (d) holds.[2] Thus

$$\begin{aligned}
\triangle_2 &= \{Plan1^2, Plan2^2\} \\
Init_{Plan1^2} &= Init_{Plan2^2} = EX(folder([UnansweredMails])); EX(h([\,])) \\
Mid_{Plan1^2} &= Mid_{Plan2^2} = \Diamond\, EX(readmails(arg2)) \\
Rest_{Plan1^2} &= EX(d([arg2])); EX(folder(['\#'])) \vee EX(quit()) \\
Rest_{Plan2^2} &= EX(quit())
\end{aligned}$$

Having skipped one step where $\circ^2 EX(read([7]))$ was observed, we get $\circ^3 EX(d([7]))$. $Plan2^3$ is no longer a valid hypothesis because (a) is not fulfilled. So we get

$$\begin{aligned}
\triangle_4 &= \{Plan1^4\} \\
Init_{Plan1^4} &= EX(folder([UnansweredMails])); EX(h([\,])); EX(read([7])); EX(d([7])) \\
Mid_{Plan1^4} &= EX(folder(['\#'])) \vee EX(quit()) \\
Rest_{Plan1^4} &= \emptyset
\end{aligned}$$

In the final step, the observation of $\circ^4 EX(folder(['\#']))$ leads to a successful recognition of the first hypothesis because $Rest_{Plan1^5}$ contains no more actions. Thus,

$$\begin{aligned}
\triangle_5 = {} & EX(folder([UnansweredMails])); EX(h([\,])); EX(read([7])); \\
& EX(d([7])); EX(folder(['\#']))
\end{aligned}$$

is a concrete instance of our initial hypothesis *Plan1* and the recognition process succeeds.

[2] If it is allowed to do some action *sometimes*, it is feasible to execute it in the *next* state.

5 Plan Generation

The plan generation facility consists of four different modules and a local knowledge base. The *deductive planner* takes formal logic plan specifications as its input and automatically generates abstract plans from them. These plans are represented by plan formulas as described in section 3. The generation of plans is guided by strategies and heuristics which have succesfully been developed for a deductive program synthesis system [Biu88]. To produce concrete and executable plans, the abstract ones are forwarded to a *compiler* module which incrementally generates sequences of basic operations. These sequences constitute the output of the plan generation facility in the second cross-talk mode. The *coordinator* module (see figure 1) analyzes user inputs, actions, and goals and activates the planner to completely generate a new plan or it activates the *reuse component.* This module enables the system to reuse previously generated plans and implements *planning from second principles.*
Subsequently, we focus on the deductive planner and its integrated reuse facility as the main parts of the plan generation system and explain how the generation and reuse of plans proceeds.

Deductive Planning

The deductive plan generator starts from a formal plan specification given as a formula of modal temporal logic. This *specification formula* contains as a subformula an atom of the form $EX(z)$, where z is an existentially quantified variable of type command-name. Generating a plan from such a specification means to first replace the variable z by an appropriate *skolem term*, e.g., $plan(x)$ and then produce an axiom $\forall x(EX(plan(x)) \leftrightarrow \phi)$, where ϕ is a modal *plan formula* as described in section 3. It additionally must have the property that replacing $EX(z)$ by ϕ in the specification formula makes this formula true, i.e., the plan ϕ to be generated has to satisfy its specification. To achieve this, the plan formula ϕ is derived from the specification formula using special plan generation rules. These rules are partly borrowed from a set of transformation rules initially developed for the deductive synthesis of programs in [Biu91] and adapted to the solution of planning problems in [Biu90].
To give an idea of how deductive planning works in this context we give a short example. Suppose we want to generate a plan for reaching the goal: *"Read and delete all mails from sender otto"*. This plan specification is represented by the following specification formula:

$$\begin{array}{l}\forall m : mbox\ \exists z : command\_name \\ \quad [EX(z) \rightarrow \forall x : mail\_object \quad [member(x,m) \wedge sender(x) = \text{“}otto\text{”} \wedge \neg flag(x) = \text{“}d\text{”} \\ \qquad\qquad\qquad\qquad\qquad\qquad \rightarrow \Diamond\, [flag(x) = \text{“}r\text{”} \wedge \Diamond\, flag(x) = \text{“}d\text{”}\ ]\,]\,]\end{array}$$

Skolemization of this formula replaces z by the term $plan(m)$, where *plan* is supposed to be a new function symbol, and yields the formula

$$\forall m : mbox\ [EX(plan(m)) \rightarrow \forall x : mail\_object \quad [member(x,m) \wedge sender(x) = \text{“otto”} \wedge \neg flag(x) = \text{“d”} \rightarrow \Diamond\,[flag(x) = \text{“r”} \wedge \Diamond\, flag(x) = \text{“d”}\,]]]$$

In order to obtain an axiom $\forall m : mbox\ (EX(plan(m)) \leftrightarrow \phi)$ defining the specified plan two tasks have to be performed. The first one is deriving a subplan $plan'(x)$ which for *any* of the specified mail objects reaches the subgoals of reading and deleting it. The second task is to find an appropriate control structure (in our case a *while* loop) which guarantees that $plan'(x)$ will be carried out for *each* of the described mail objects.
We will start with the first task and show how this part of the final plan can be derived using a widely extended version of the so-called *implication rule* (cf. [Biu91]) together with the following axioms which are supposed to be available in our knowledge base:

Ax1: $\circ\phi \rightarrow \Diamond\,\phi$

Ax2: $\circ(\phi \wedge \psi) \leftrightarrow (\circ\phi \wedge \circ\psi)$

Ax3: $\circ\Diamond\,\phi \leftrightarrow \Diamond\,\circ\phi$

Ax4: $\forall x : mail\_object$
$$[\neg flag(x) = \text{“d”} \wedge EX(read(x)) \rightarrow \circ flag(x) = \text{“r”}\,]$$

Ax5: $\forall x : mail\_object$
$$[\neg flag(x) = \text{“d”} \wedge EX(delete(x)) \rightarrow \circ flag(x) = \text{“d”}\,]$$

Ax4 and Ax5 describe the *read* and *delete* actions, respectively.

Let C, L, M, and $K_i (1 \leq i \leq n)$ be formulas. The implication rule then reads:

IMPL:
$$\frac{C \rightarrow (\sigma L \wedge M)}{C \rightarrow (\sigma K_1 \wedge M), \ldots, C \rightarrow (\sigma K_n \wedge M)}$$

provided there exists an axiom $(K_1 \wedge ... \wedge K_n) \rightarrow L$ in the knowledge base. According to the underlying modal logic the following rule derived from IMPL will also be used:

NEXT_IMPL:
$$\frac{C \rightarrow (\circ\sigma L\ \wedge M)}{C \rightarrow (\circ\sigma K_1\ \wedge M), \ldots, C \rightarrow (\circ\sigma K_n\ \wedge M)}$$

The implication rule is used to replace a (sub)goal in the plan specification by new subgoals which are sufficient for it.

In order to derive a plan formula for our subplan $plan'(x)$ from its specification

$$\forall x : mail\_object \quad [EX(plan'(x)) \rightarrow [\neg flag(x) = \text{“d”} \rightarrow \Diamond\,[(flag(x) = \text{“r”} \wedge \Diamond\, flag(x) = \text{“d”}\,)]\,]\,]$$

we start with

$$[\neg flag(x) = \text{“d”} \rightarrow \Diamond\,[flag(x) = \text{“r”} \wedge \Diamond\, flag(x) = \text{“d”}\,]]$$

and apply the implication rule together with axiom Ax1, i.e., we replace the conclusion by

$\circ[flag(x) = \text{“}r\text{”} \wedge \Diamond flag(x) = \text{“}d\text{”}]$ obtaining

$$[\neg flag(x) = \text{“}d\text{”} \rightarrow \circ[flag(x) = \text{“}r\text{”} \wedge \Diamond flag(x) = \text{“}d\text{”}]]$$

as a new formula.

According to Ax2 this formula can be equivalently transformed into

$$[\neg flag(x) = \text{“}d\text{”} \rightarrow \circ flag(x) = \text{“}r\text{”} \wedge \circ\Diamond flag(x) = \text{“}d\text{”}].$$

Now the implication rule together with axiom Ax4 is applied in order to replace the subgoal

$\circ flag(x) = \text{“}r\text{”}$ by the plan formula $EX(read(x))$.

We obtain two new formulas:

$$\phi_1 : \neg flag(x) = \text{“}d\text{”} \rightarrow EX(read(x)) \wedge \circ\Diamond flag(x) = \text{“}d\text{”}$$

and

$$\phi_2 : \neg flag(x) = \text{“}d\text{”} \rightarrow \neg flag(x) = \text{“}d\text{”} \wedge \circ\Diamond flag(x) = \text{“}d\text{”}\ .$$

The formula ϕ_1 is now transformed in order to even obtain a plan formula for the second subgoal $\circ\Diamond flag(x) = \text{“}d\text{”}$.

First of all ϕ_1 can, according to Ax3, be replaced by:

$$\neg flag(x) = \text{“}d\text{”} \rightarrow EX(read(x)) \wedge \Diamond \circ flag(x) = \text{“}d\text{”}\ .$$

Now the implication rule is applied with Ax1 to get

$$\neg flag(x) = \text{“}d\text{”} \rightarrow EX(read(x)) \wedge \circ\circ flag(x) = \text{“}d\text{”}$$

and finally applying that rule with Ax5 yields:

$$\neg flag(x) = \text{“}d\text{”} \rightarrow EX(read(x)) \wedge \circ EX(delete(x))\,.$$

Applying rule NEXT_IMPL in a final step we again obtain two new formulas:

$$\phi_3 : \neg flag(x) = \text{“}d\text{”} \rightarrow EX(read(x)) \wedge \circ EX(delete(x))$$

and

$$\phi_4 : \neg flag(x) = \text{“}d\text{”} \rightarrow EX(read(x)) \wedge \circ\neg flag(x) = \text{“}d\text{”}\ .$$

From ϕ_3 the following plan formula can be derived:

$$\phi_3 : \neg flag(x) = \text{“}d\text{”} \rightarrow EX(read(x));\ EX(delete(x)).$$

Hence, we obtain

$$\forall x : mail\_object \quad [EX(plan'(x)) \leftrightarrow [\neg flag(x) = \text{“}d\text{”} \rightarrow EX(read(x)); EX(delete(x))]]$$

as a defining axiom for the specified plan $plan'(x)$.

The formulas ϕ_2 and ϕ_4 which also have been derived during the generation process describe two properties of the new plan:

$$\forall x : mail\_object[EX(plan'(x)) \rightarrow \ [\neg flag(x) = \text{“}d\text{”} \rightarrow [\neg flag(x) = \text{“}d\text{”} \wedge \circ\Diamond flag(x) = \text{“}d\text{”}]$$

and

$$\forall x : mail\_object[EX(plan'(x)) \rightarrow \quad [\neg flag(x) = \text{“}d\text{”} \\ \rightarrow [EX(read(x)) \wedge \circ \neg flag(x) = \text{“}d\text{”}\,]]$$

They represent so-called *verification formulas* that have to be proved in order to guarantee that the generated plan indeed satisfies its specification. This proof can be easily done using the definition of $plan'(x)$ above and an axiom asserting the *read*- and *delete*-flags to be different.
Selecting the appropriate axioms and rules is essential for the plan generation process to succeed. Additionally, this selection in particular influences the degree of abstraction the generated plan has. If, for example, we had decided to use instead of axioms Ax4 and Ax5 the *weaker* versions Ax4' and Ax5' with

Ax4': $\forall x : mail\_object$
$$[\neg flag(x) = \text{“}d\text{”} \wedge EX(read(x)) \rightarrow \Diamond\, flag(x) = \text{“}r\text{”}\,]$$

Ax5': $\forall x : mail\_object$
$$[\neg flag(x) = \text{“}d\text{”} \wedge EX(delete(x)) \rightarrow \Diamond\, flag(x) = \text{“}d\text{”}\,],$$

then the generated plan definition would have read:

$$\forall x : mail\_object[EX(plan'(x)) \leftrightarrow \quad [\neg flag(x) = \text{“}d\text{”} \rightarrow \\ \Diamond\, EX(read(x))\,;\Diamond\, EX(delete(x))\,]\,]$$

To finally end up with the plan generation process starting from our initial specification of *plan*:

$$\forall m : mbox \quad [EX(plan(m)) \rightarrow \\ \forall x : mail\_object \quad [member(x,m) \wedge sender(x) = \text{“}otto\text{”} \wedge \neg flag(x) = \text{“}d\text{”} \\ \rightarrow \Diamond\,[flag(x) = \text{“}r\text{”} \wedge \Diamond\, flag(x) = \text{“}d\text{”}\,]\,]\,]$$

we have to introduce a while-loop in order to work through the list of all mail objects from sender "otto" and carry out the generated subplan $plan'(x)$ for each of its elements. Finally we obtain the following plan definition:

$$\forall m : mbox\ [EX(plan(m)) \leftrightarrow \quad [a := from(sender, \text{“}otto\text{”}, m); \\ while\ \neg Empty(a)\ do \\ b := first(a);\ EX(plan'(b));\ a := tail(a)\ od]\,]$$

Plan Reuse

A plan as generated in section 5 represents problem solving knowledge that was used by the planning system to achieve a given goal state from a particular initial state. Therefore, we develop a reuse mechanism that enables the planner to save generated plans for a later reuse and thus extend the problem solving knowledge. The planning knowledge can now be applied to find out whether a problem can be solved by adapting an already existing plan. The architecture of the *reuse component* is based on a 4-phase model (cf. [Köh91]) describing the reuse process:

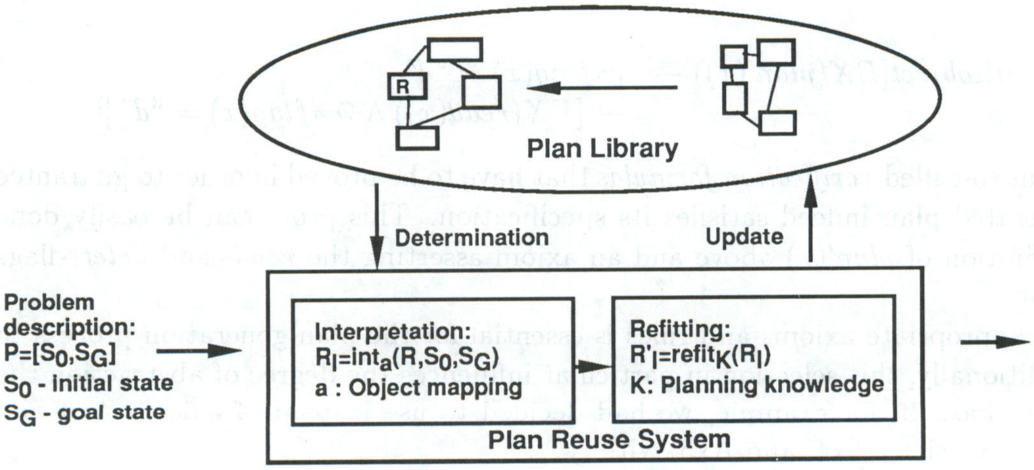

Figure 2: A 4-Phase Model of Plan Reuse

To explain how the reuse process works we reuse the plan that was generated in the preceding example to solve the new planning task: *"Read all mails from otto, save them in the folder with the sender's name, and then delete the mails"*. It is represented by the following specification formula:

$$
\begin{array}{lll}
\langle P :\rangle & \forall m, n : & mbox\ \exists z : command\_name \\
& [EX(z) & \rightarrow \forall x : mail\_object \\
& & [member(x,m) \wedge sender(x) = \text{"}otto\text{"} \wedge \neg flag(x) = \text{"}d\text{"} \\
& & \wedge\ folder(n) = \text{"}otto\text{"} \\
& & \rightarrow \Diamond\, [flag(x) = \text{"}r\text{"} \wedge \Diamond\, [flag(x) = \text{"}\star\text{"} \wedge member(x,n) \\
& & \wedge \Diamond\, flag(x) = \text{"}d\text{"}\ ]\,]\,]\,]
\end{array}
$$

Determination of a Reuseable Plan Entry

To solve the planning problem, a stored plan entry from the plan library is determined. We presuppose that the plan library does not contain (user-)predefined plan entries, but is built up using information provided by the deductive plan generation component, e.g., the generalized specification formula, the generalized plan schema, the verification formulas for the plan. The determination process mainly concentrates on a syntactical comparison of the current specification formula P with the generalized specification formulas R occurring in the various plan entries. In our example the determination process chooses the following generalized plan specification R from the plan library as a hypothesis on which a solution for P can be based upon:

$$
\begin{array}{llll}
\langle R :\rangle & \forall u : mbox & \forall s : sender & \exists v : command\_name \\
& & [EX(v) \rightarrow & \forall w : mail\_object \\
& & & [member(w,u) \wedge sender(w) = s \wedge \neg flag(w) = \text{"}d\text{"} \\
& & & \rightarrow \Diamond\, [flag(w) = \text{"}r\text{"} \wedge \Diamond\, flag(w) = \text{"}d\text{"}\ ]\,]\,]
\end{array}
$$

Interpretation of the Plan Entry in the Current Planning Situation

Now R has to be interpreted in the current planning situation by matching the two formulas. The main problem here is to find the correct mapping a of objects in P to the variables in R to generate a correct instantiation of R. Obviously, an optimal solution can be obtained by applying the substitution $\{v \leftarrow z, u \leftarrow m, w \leftarrow x, s \leftarrow otto\}$ to R leading to its instantiation:

$$
\begin{array}{ll}
\langle R_I :\rangle & \forall m : mbox \;\; \exists z : command\_name \\
& \quad [EX(z) \rightarrow \forall x : mail\_object \\
& \qquad [member(x,m) \wedge sender(x) = \text{“}otto\text{”} \;\wedge \neg flag(x) = \text{“}d\text{”} \\
& \qquad \rightarrow \Diamond\, [flag(x) = \text{“}r\text{”} \wedge \Diamond\, flag(x) = \text{“}d\text{”}\;]\,]\,]
\end{array}
$$

Refitting of the Interpreted Plan Entry

By completing the instantiation phase in our example we obtain a fully instantiated plan specification R_I which we can now compare with the current plan specification P to evaluate whether we already obtained a solution. In general, we will be confronted with the problem that the plan specifications differ in the description of the initial or the goal state, thus requiring a refitting of the plan corresponding to R_I. In our example a number of formulas in P have no corresponding formula in R_I, meaning that the plan we want to choose for reuse will only partially solve the current goal. Thus, we obtain a formula R_I' which contains the generated plan $plan'(x)$, but also an open subgoal for which the planner has to be activated again:

$$
\begin{array}{ll}
\langle R_I' :\rangle & \forall m, n : mbox \; \exists z : command\_name \\
& \quad [EX(z) \rightarrow \forall x : mail\_object \\
& \qquad [member(x,m) \wedge sender(x) = \text{“}otto\text{”} \wedge \neg flag(x) = \text{“}d\text{”} \\
& \qquad \wedge\; folder(n) = \text{“}otto\text{”} \\
& \qquad \rightarrow EX(read(x));\; \Diamond\, flag(x) = \text{“}\star\text{”} \wedge member(x,n)\,;\; EX(delete(x))]\,]
\end{array}
$$

This specification describes that the plan to be reused has to be modified in such a way, that an additional condition has to hold in the initial state and that an additional action has to be included.

Updating the Plan Library

The reuse process finishes with the update of the plan library. The decision whether a plan is "worth" storing in the plan library depends on its similarity to already stored plans. A new plan entry is built up from the specification formula for the plan, the plan itself, the verification conditions for the plan, and the transformation rules used in the generation process. Furthermore, an abstraction process (cf. section 4) will be applied leading to the storage of abstract plan entries.

References

[Bib86] W. Bibel. A deductive solution for plan generation. *New Generation Computing*, 4:115–132, 1986.

[Biu88] S. Biundo. Automated synthesis of recursive algorithms as a theorem proving tool. In *Proceedings of the 8th European Conference on Artificial Intelligence, München*, pages 553–558, 1988.

[Biu90] S. Biundo. Plan generation using a method of deductive program synthesis. Research Report RR-90-09, German Research Center for Artificial Intelligence Inc., 1990.

[Biu91] S. Biundo. *Automatische Synthese rekursiver Algorithmen als Beweisverfahren.* Informatik Fachberichte. Springer, Berlin, 1991. forthcoming.

[Bre90] J. Breuker. *EUROHELP Developing Intelligent Help Systems.* EC, Kopenhagen, 1990.

[DGH87] D. Dengler, M. Gutmann, and G. Hector. Der Planerkenner REPLIX. Memo No. 16, Dept. of Computer Science, University of Saarbrücken, W.Germany, 1987.

[Fin83] T. W. Finin. Providing help and advice in task oriented systems. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 176–178, 1983.

[FLS85] G. Fischer, A. Lemke, and T. Schwab. Knowledge-based help systems. In *Proceedings of Human Factors in Computing Systems (CHI'85)*, pages 161–167, 1985.

[Hal89] R. W. S. Hale. Programming in temporal logic. Technical Report 173, Computer Laboratory, University of Cambridge, England, 1989.

[Hec87] M. Hecking. How to Use Plan Recognition to Improve the Abilities of the Intelligent Help System SINIX Consultant. In *Proceedings of the Second IFIP Conference on Human-Computer Interaction, held at the University of Stuttgart, Federal Republic of Germany, 1-4 September, 1987*, pages 657–662, 1987.

[Hec91] M. Hecking. *Eine logische Behandlung der verteilten und mehrstufigen Planerkennung.* PhD thesis, University of Saarbrücken, 1991. forthcoming.

[Heg91] S.J. Hegner. Plan realization for complex command interactions in the unix help domain. In P. Norwig, W. Wahlster, and R. Wilensky, editors, *Intelligent Help Systems for UNIX - Case Studies in Artificial Intelligence.* Springer, 1991.

[HKN+88] M. Hecking, C. Kemke, E. Nessen, D. Dengler, M. Gutmann, and G. Hector. The SINIX Consultant - A Progress Report. Memo No. 28, Dept. of Computer Science, University of Saarbrücken, W.Germany, 1988.

[Köh91] J. Köhler. Approaches to the reuse of plan schemata in planning formalisms. Technical Memo TM-91-01, German Research Center for Artificial Intelligence Inc., 1991.

[Krö87] F. Kröger. *Temporal Logic of Programs.* Springer, Heidelberg, 1987.

[Lur88] M. Luria. Knowledge intensive planning. Technical report ucb/csd 88/433, Computer Science Division, University of California, 1988.

[Mer91] G. Merziger. Approaches to abduction - an overview. Technical memo, German Research Center for Artificial Intelligence Inc., 1991. forthcoming.

[MW86] Z. Manna and R. Waldinger. How to clear a block: Plan formation in situational logic. In *Proceedings CADE 86*, pages 622–640, 1986.

[NWWng] P. Norwig, W. Wahlster, and R. Wilensky. *Intelligent Help Systems for UNIX - Case Studies in Artificial Intelligence.* Springer, Heidelberg, 1991 (forthcoming).

[RP86] R. Rosner and A. Pnueli. A choppy logic. In *Symposium on Logic in Computer Science*, Cambridge, Massachusetts, 1986.

[SC85] M. Sullivan and P. R. Cohen. An endorsement-based plan recognition program. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 475–479, 1985.

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-90-01
Franz Baader: Terminological Cycles in KL-ONE-based Knowledge Representation Languages
33 pages

RR-90-02
Hans-Jürgen Bürckert: A Resolution Principle for Clauses with Constraints
25 pages

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 pages

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, and 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, and 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-91-15
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-91-20
Christoph Klauck, Ansgar Bernardi, Ralf Legleitner
FEAT-Rep: Representing Features in CAD/CAM
48 pages

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

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 -
14 pages

DFKI Technical Memos

TM-89-01
Susan Holbach-Weber: Connectionist Models and Figurative Speech
27 pages

TM-90-01
Som Bandyopadhyay: Towards an Understanding of Coherence in Multimodal Discourse
18 pages

TM-90-02
Jay C. Weber: The Myth of Domain-Independent Persistence
18 pages

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 and Rainer Bleisinger
Theoretical Consideration of Goal Recognition Aspects for Understanding Information in Business Letters
10 pages

DFKI Documents

D-89-01
Michael H. Malburg, Rainer Bleisinger: HYPERBIS: ein betriebliches Hypermedia-Informationssystem
43 Seiten

D-90-01
DFKI Wissenschaftlich-Technischer Jahresbericht 1989
45 pages

D-90-02
Georg Seul: Logisches Programmieren mit Feature -Typen
107 Seiten

D-90-03
Ansgar Bernardi, Christoph Klauck, Ralf Legleitner: Abschlußbericht des Arbeitspaketes PROD
36 Seiten

D-90-04
Ansgar Bernardi, Christoph Klauck, Ralf Legleitner: STEP: Überblick über eine zukünftige Schnittstelle zum Produktdatenaustausch
69 Seiten

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
$H^{iera}C_{on}$ - a Knowledge Representation System with Typed Hierarchies and Constraints
75 Seiten

Integrated Plan Generation and Recognition
- A Logic-Based Approach -

M. Bauer, S. Biundo, D. Dengler, M. Hecking, J. Koehler, G. Merziger

RR-91-26
Research Report