Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research Report
RR-92-11

Deductive Planning and Plan Reuse in a Command Language Environment

Susanne Biundo, Dietmar Dengler, Jana Koehler

March 1992

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

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

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

Deutsches Forschungszentrum für Künstliche Intelligenz

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

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

- Intelligent Engineering Systems
- Intelligent User Interfaces
- Intelligent Communication Networks
- Intelligent Cooperative Systems.

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

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

Prof. Dr. Gerhard Barth
Director

Deductive Planning and Plan Reuse in a Command Language Environment

Susanne Biundo, Dietmar Dengler, Jana Koehler

DFKI-RR-92-11

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

Deductive Planning and Plan Reuse in a Command Language Environment

S. Biundo D. Dengler J. Koehler
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3
D-W-6600 Saarbrücken 11
Germany
e-mail: <last name>@dfki.uni-sb.de

Abstract

In this paper we introduce a deductive planning system currently being developed as the kernel of an intelligent help system. It consists of a deductive planner and a plan reuse component and with that provides planning from first as well as planning from second principles. Both components rely upon an interval-based temporal logic. The deductive formalisms realizing plan formation from formal specifications and the reuse of already existing plans respectively are presented and demonstrated by examples taken from an operating system's domain.

Contents

1 Introduction

Intelligent help systems aim at supporting users of complex software systems. Advanced active help can thereby be provided if the help system on the one hand is able to observe and interpret the users actions in order to recognize the goals she pursues. On the other hand, based on this information plans have to be generated and supplied to the user that enable her to reach these goals properly.
Consequently, the PHI-System [BBD+91] currently being developed as the kernel of an intelligent help system provides both, a *plan recognizer* to anticipate the users goals and a *plan generation component* that supports the user with plans to reach these goals.
One of PHI's main characteristics are the close mutual cooperation between the plan recognition and plan generation components. One feature of this cooperation distinguishes itself by the use of (abstract) *plans* as the basis for plan recognition. Starting from a formal plan specification the generation component produces a set of hypothetical plans. These hypotheses are used by the recognizer to identify the users plan by trying to map the observed actions on an instance of any of the plan hypotheses. By *abstract* plans we mean plans that contain variables, abstract commands, control structures, and indeterministic branching.
Besides these abstract ones also concrete plans, i.e., sequences of fully instantiated basic actions, play a central role in our scenario since the user has to be supported by executable and even by optimal plans [BBD+91].
The planning system we introduce in this paper meets the claims described above by relying on methods borrowed from the formal (logic-based) treatment of programs. The reason is that we follow the *"plans are programs"* paradigm proposed by other authors as well (cf. [Bib86] and [MW87]), because this seems to be highly adequate in our case: The planning system works in a help system's context. Hence, the planning domain is a command language environment where the basic actions are elementary statements of the application system's language. The state changes performed by these basic actions correspond to changes provided by assignment statements in programming languages. As a consequence, the logical framework we have developed to realize *deductive planning* and *plan reuse* in this context differs in several aspects from the deductive planning approaches known from the literature (cf. [Gre69], [Kow79], [Bib86], and [MW87]). It relies upon an interval-based temporal logic that combines features of traditional programming logics, like, for example, *dynamic logic* [Har79] or *temporal logic of programs* [Krö87].
In the examples presented in this paper our planning domain is chosen to be a subset of an operating system, namely a *mail system*, where commands like *type*, *delete*, or *save* manipulate objects, like *messages* or *mailboxes*.

The paper is organized in the following way: In Section 2 we briefly sketch the architecture of our planning system. Section 3 introduces the logical framework underlying both, the deductive planner as well as the deductive reuse component, and describes our deductive planning method by means of a short example. In Section 5 a four-phase plan reuse model is proposed. We present a method to realize *plan modification* deductively and demonstrate this method by a detailed example.

2 The Planning System

The planning system, shown in Figure 1, consists of a *deductive plan generator*, the *reuse component*, and a *plan interpreter*.

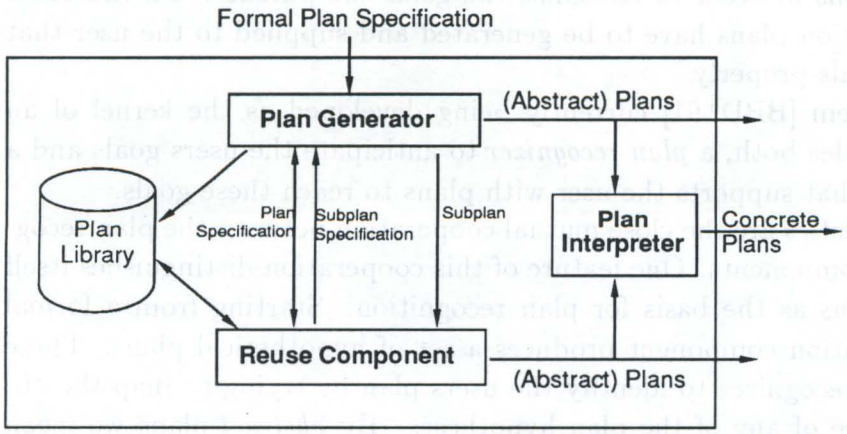

Figure 1: The Planning System

To solve the tasks of producing abstract plan hypotheses and executable plans, respectively, the system provides means for both, planning from first as well as planning from second principles. It works in the following way:
A formal plan specification Φ (i.e. a special LLP-formula, cf. Section 3) given to the deductive planner is forwarded to the reuse component. If the reuse component succeeds in hunting up a plan from the library that (perhaps after minor modifications) can be used to solve Φ the *plan modification* process starts (cf. Section 5). This process implements planning from second principles: It takes an existing plan together with its generation process (which in our case is represented by a *proof tree*, cf. Section 3) out of the library. If the plan has to be modified, for example, by inserting additional actions, a formal subplan specification is generated and passed to the planner. The planner generates a subplan, which then is used to extend the already existing plan in such a way that it satisfies even the current specification Φ.
If no reuse "candidate" can be found the deductive planner has to generate a completely new plan out of the given specification by carrying out a special kind of *constructive proof* of the specification formula. As a result a so-called *plan formula* occurs that represents the specified plan (cf. Section 3).
Besides linear plans, being seqences of basic actions, even conditional and *while*-plans can be derived [BD91] as well as plans containing indeterministic branching. If executable plans are required to be produced in a certain situation the plan interpreter finally is activated to eliminate these control structures if necessary.

3 The Logical Framework

The *logical language for planning* (LLP) we have developed to do deductive planning in our help system context is an interval-based modal temporal logic that combines features of *choppy logic* [RP86] with a *temporal logic for programs* [Krö87]. The basis of LLP

relies on a many-sorted first-order language and, besides the normal logical variables, LLP provides a set of so-called *local variables* for each sort. The local variables are borrowed from programming logics where they correspond to program variables whose values can change from one state to another. We use local variables in the same way and describe the effects of basic actions by a change of values of certain local variables.
The modal operators provided by LLP are $\bigcirc$(next), $\Diamond$ (sometimes), $\Box$ (always), and a sequential composition of formulas by the two-place modal operator ; (chop). Besides these operators, like in programming logics, also *control structures* are available. The conditional $\underline{if}\ \epsilon\ \underline{then}\ \alpha\ \underline{else}\ \beta$ for example, stands for the formula $[\epsilon \rightarrow \alpha] \wedge [\neg\epsilon \rightarrow \beta]$. The *while*-operator is defined by the following axiom:
$\underline{while}\ \epsilon\ \underline{do}\ \alpha\ \underline{od}\ ; \beta \leftrightarrow [\underline{if}\ \epsilon\ \underline{then}\ [\alpha\ ; \underline{while}\ \epsilon\ \underline{do}\ \alpha\ \underline{od}\ ; \beta]\ \underline{else}\ \beta]$.
Basic actions are represented by atomic formulas using the predicate EX ("execute"). $EX(type(1, mbox))$, for example, represents the basic action of reading the first message in a mailbox $mbox$.
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*,
- all formulas $\alpha; \beta$ where α and β are plan formulas,
- all formulas $\underline{if}\ \epsilon\ \underline{then}\ \alpha\ \underline{else}\ \beta$, where α and β are plan formulas and ϵ is a formula not containing any temporal operator or basic plan formula,
- all formulas $\underline{while}\ \epsilon\ \underline{do}\ \alpha\ \underline{od}\ ; \beta$, where α and β are plan formulas and ϵ is a formula not containing any temporal operator or basic plan formula.

3.1 Syntax

LLP provides a many-sorted language with equality where we have a nonempty set of *sort symbols* S, a S-sorted signature of function symbols $\Sigma^F = (\Sigma^F_{ws})_{w \in S^*,\, s \in S}$ and a S-sorted signature of predicate symbols $\Sigma^P = (\Sigma^P_v)_{v \in S^*}$, where $\{EX\} \subseteq \Sigma^P_{command}$ and $\{\mathrm{T}\} \subseteq \Sigma^P_\epsilon$.
For $f \in \Sigma^F_{ws}$ we call ws the rank, w the arity, and s the sort of f. For $p \in \Sigma^P_v$ is v the arity of p. The signature Σ is defined according to $\Sigma = \Sigma^F \cup \Sigma^P$.
Having VG_s and VL_s as the sets of all global and local variables of sort $s \in S$, respectively, and defining $V_s = VG_s \cup VL_s$ as all variables of sort $s \in S$ the set of *well-sorted* Σ-terms of sort $s \in S$ is obtained as usual. Finally, we have $\mathcal{T}_\Sigma = (\mathcal{T}(\Sigma)_s)_{s \in S}$ as the set of all Σ-terms.
The set $\mathcal{F}_\Sigma$ of Σ-formulas is built using the following operators $\{\neg, \wedge, \forall, \bigcirc, \Box, ;\}$.
We use the abbreviations $\Diamond\phi \leftrightarrow \neg\Box\neg\phi$, $\phi \rightarrow \psi \leftrightarrow \neg(\phi \wedge \neg\psi)$, and $\neg\ \mathrm{T} \leftrightarrow \mathrm{F}$.

3.2 Semantics

Given a signature Σ, a Σ-structure is a pair $(D,\ I)$, where $D = (D_s)_{s \in S}$ is called the *domain* and $I = (I(f))_{f \in \Sigma}$ is a family of mappings assigning functions and predicates over $(D,\ I)$ to the symbols in Σ.
Global variables are mapped to elements of the domain using the *sort-preserving* valuation function $\beta : VG \rightarrow_s D$.
To define the notion of an *interval* we start from a nonempty infinite set of states $\mathcal{S} = \{\sigma_0, \ldots, \sigma_n, \ldots\}$. Each state σ_i is a pair $\sigma_i = (\sigma_i^1, \sigma_i^2)$. $\sigma_i^1 : VL \rightarrow_s D$ is a valuation that assigns an element of D to each local variable. $\sigma_i^2 \in D_{command}$ is the so-called *control component* that indicates the command to be executed in state σ_i.

We define an interval σ to be a nonempty sequence of states: $< \sigma_0\sigma_1 \ldots >$ and W denotes a nonempty infinite set of intervals.
The *immediate accessibility* on intervals is defined as the subinterval relationship R with $\sigma R\sigma'$ iff $\sigma =< \sigma_0\sigma_1 \ldots >$ and $\sigma' =< \sigma_1 \ldots >$
R' and R^* denote the transitive and the reflexive and transitive closure of R, respectively.
The *composition* is defined as a partial function over the set of intervals:

$$\sigma \circ \sigma' = \begin{cases} \sigma, \text{ if } \sigma \text{ is infinite} \\ < \sigma_0 \ldots \sigma_n \ldots >, \text{ if } \sigma =< \sigma_0 \ldots \sigma_n > \text{ and } \sigma' =< \sigma_n \ldots > \end{cases}$$

We call the triple (W, R, $\circ$) a *frame*.
Given a Σ-*interpretation* $\mathcal{I}$ the value of a term $t \in \mathcal{T}_\Sigma$ in an interval $\sigma \in W$ is defined according to:

- $\mathcal{I}_\Sigma(x) = \beta(x)$ for every $x \in VG$;
- $\mathcal{I}_\Sigma(a) = \sigma_0(a)$ for every $a \in VL$;
- Function expressions ft^* are interpreted as usual.

A formula $\phi \in \mathcal{F}_\Sigma$ holds under a Σ-interpretation $\mathcal{I}$ in an interval $\sigma \in W$ ($\sigma \models_\mathcal{I} \phi$) according to:

- $\sigma \models_\mathcal{I} \mathrm{T}$
- $\sigma \models_\mathcal{I} EX(t)$ iff $\mathcal{I}_\Sigma(t) = \sigma_0^2$
- $\sigma \models_\mathcal{I} \bigcirc\phi$ iff $\sigma' \models_\mathcal{I} \phi$ for all $\sigma' \in W$ where $\sigma R\sigma'$
- $\sigma \models_\mathcal{I} \Box\phi$ iff $\sigma' \models_\mathcal{I} \phi$ for all $\sigma' \in W$ where $\sigma R^*\sigma'$
- $\sigma \models_\mathcal{I} \phi\,;\psi$ iff there are $\sigma', \sigma'' \in W$, where $\sigma = \sigma' \circ \sigma''$, σ' finite and $\sigma' \models_\mathcal{I} \phi$ and $\sigma'' \models_\mathcal{I} \psi$
- $\sigma \models_\mathcal{I} \psi$ is defined as usual for $\psi = pt_1 \ldots t_n$, $\psi = t_1 \equiv t_2$, $\psi = \neg\phi$, $\psi = [\phi_1 \wedge \phi_2]$, $\psi = \forall x\ \phi$.

Finally, a Σ-interpretation $\mathcal{I}$ is a *model* of a formula $\phi \in \mathcal{F}_\Sigma$ ($\models_\mathcal{I} \phi$) iff $\sigma \models_\mathcal{I} \phi$ for every $\sigma \in W$. $\phi \in \mathcal{F}_\Sigma$ is *valid* iff $\models_\mathcal{I} \phi$ for every Σ-interpretation $\mathcal{I}$. A formula $\phi \in \mathcal{F}_\Sigma$ *follows* from a set of formulas $\Phi \subset \mathcal{F}_\Sigma$ iff $\models_\mathcal{I} \phi$ for every Σ-interpretation $\mathcal{I}$ with $\models_\mathcal{I} \psi$ for every $\psi \in \Phi$.

3.3 Calculus

The calculus we use for LLP is based on a complete sequent calculus for S4 modal logic as it is defined in [Wal89]. We have extended this calculus by giving additional rules for handling the modalities $\bigcirc$, ; and *while*.
For lack of space we describe here only the *next*- and the *chop composition*-rule and introduce other basic as well as derived rules when we use them in the sequel.
Remember that a *sequence* is denoted by $\Gamma \Rightarrow \Delta$, where Γ and Δ are sequences of (LLP-) formulas and the *conjunction* of the formulas in the *antecedent* Γ *implies* the *disjunction* of the formulas in the *consequent* Δ.

- *next*-rule: $\dfrac{\Gamma^* \Rightarrow A, \Delta^*}{\Gamma \Rightarrow \bigcirc A, \Delta}$ with $\begin{array}{l} \Gamma^* = \{B \mid \bigcirc B \in \Gamma\} \cup \{\Box B \mid \Box B \in \Gamma\}, \text{ and} \\ \Delta^* = \{B \mid \bigcirc B \in \Delta\} \cup \{\Diamond B \mid \Diamond B \in \Delta\} \end{array}$

- *chop composition*-rule: $\dfrac{P_1 \Rightarrow S_1 \qquad P_2 \Rightarrow S_2}{P_1;P_2 \Rightarrow S_1;S_2}$

3.4 Representation of the Planning Domain

As described above the application domain we choose for our examples is a mail system. The main objects in this domain are "mailboxes" and "messages"; a mailbox is viewed as a list of one or more messages. During the activation of the mail system different aspects of messages can be changed by the commands the user executes: so every executed command causes a state transition of the current mailbox. In our logical formalism we deal with this behaviour by the use of local variables for identifying objects of type *mailbox* or *message*, respectively.
The axioms describing the different mail commands as basic actions are given like axioms for assignment statements in programming logics.
As an example we sketch the axiomatization of the "type" command for reading a message:

$$\begin{array}{l}\forall i: \ integer \\ \quad [[\neg flag(i, Current\_mbox) \equiv \text{“d”} \wedge \\ \quad \mathsf{P}^{\,flag(i,Current\_mbox)\ \ Current}_{\text{“r”}\qquad Current+1} \wedge EX(type(i, Current\_mbox))] \rightarrow \circ \mathsf{P}]\end{array}$$

The symbol P is a *metavariable* for formulas; the substitution instructions correspond to the *effect* of the "type" command: "type" does nothing else than changing the $flag$ of the i-th message in $Current\_mbox$ to "r" and increases the $Current$-counter by 1. Applying the "type" axiom during the deductive plan generation process is done by building an appropriate instance of the above axiom schema and applying it to the actual sequent.
One instance that is often used, for example, is:

$$\begin{array}{l}\forall i: \ integer \\ \quad [[\neg flag(i, Current\_mbox) \equiv \text{“d”} \wedge EX(type(i, Current\_mbox))] \\ \quad \rightarrow \circ flag(i, Current\_mbox) \equiv \text{“r”}]\end{array}$$

A corresponding instance of the axiom schema describing the "delete" command reads:

$$\begin{array}{l}\forall i: \ integer \\ \quad [[\neg flag(i, Current\_mbox) \equiv \text{“d”} \wedge EX(delete(i, Current\_mbox))] \\ \quad \rightarrow \circ flag(i, Current\_mbox) \equiv \text{“d”}]\end{array}$$

Note, that the axiom schemata describing the mail actions can also be instantiated with arbitrary *frame conditions*. That means only *one* axiom schema is needed for each action to describe its effects as well as its invariants and with that we also have obtained a representational solution of the frame problem [BD91].
Basic actions are required to terminate. This fact is expressed by special axioms. We have:

$$\forall c : command \ [EX(c) \rightarrow \circ\circ F]$$

4 Deductive Planning

The planning process starts from a plan specification formula. Specifications are formulas containing metavariables for plans. Deriving a plan from such a specification is done by constructing a sequence proof that provides appropriate instantiations for these variables. That means, based on the specification we develop a proof tree applying several sequence rules in turn until all leaves of the tree are closed, i.e. are instances of the initial sequent $\Gamma,\ \phi \ \Rightarrow\ \phi,\ \Delta$. The instantiations to be made for the plan metavariable are restricted to plan formulas. This means if we starting from the specification formula end up with a proof tree the instantiation generated for the plan variable represents a correct (i.e. *executable*) plan, i.e., a plan that satisfies the given specification.
We distinguish between different types of plan specifications. Among them we have assertions about *intermediate states* (also called *liveness properties* [Krö87]). They read

$$\mathsf{Plan} \ \rightarrow\ [\phi_i \rightarrow \Diamond\phi_g]$$

stating that ϕ_g holds some time during the execution of Plan. The examples we will present deal with these kind of specifications.
Suppose, the plan specification is "Read any message of the mailbox C_mb and delete it". The input for the plan generation process is then a formula of the form:

(1) $\mathsf{Plan} \rightarrow [flag(x, C\_mb) \not\equiv \text{“d”} \rightarrow \Diamond[flag(x, C\_mb) \equiv \text{“r”} \wedge \Diamond flag(x, C\_mb) \equiv \text{“d”}]]$

Now we give a sequent proof for formula (1) during which Plan will be replaced by a plan formula satisfying the above specification.
We start with formula (1) which corresponds to the following sequent:

(2) $\mathsf{Plan},\ flag(x, C\_mb) \not\equiv \text{“d”} \Rightarrow \Diamond[flag(x, C\_mb) \equiv \text{“r”} \wedge \Diamond flag(x, C\_mb) \equiv \text{“d”}]$

Applying the rule $rule1$:

$$rule1 \quad \frac{\Gamma \Rightarrow \bigcirc\phi \wedge \neg\bigcirc F, \Delta}{\Gamma \Rightarrow \Diamond\phi, \Delta}$$

we obtain sequent (3):

(3) $\mathsf{Plan},\ flag(x, C\_mb) \not\equiv \text{“d”} \Rightarrow \bigcirc[flag(x, C\_mb) \equiv \text{“r”} \wedge \Diamond flag(x, C\_mb) \equiv \text{“d”}] \wedge \neg\bigcirc F$

To prove (3) it is splitted into two sequents (4) and (5):

(4) $\mathsf{Plan}, flag(x, C\_mb) \not\equiv \text{“d”} \Rightarrow \bigcirc[flag(x, C\_mb) \equiv \text{“r”} \wedge \Diamond flag(x, C\_mb) \equiv \text{“d”}]$

(5) $\mathsf{Plan}, flag(x, C\_mb) \not\equiv \text{“d”} \Rightarrow \neg\bigcirc F$

Sequent (5) can easily be proved if the instantiation for Plan has been found, because it only says that the plan is not the empty plan. We are further going on with sequent (4), make an equivalence transformation reaching (6):

(6) $\mathsf{Plan}, flag(x, C\_mb) \not\equiv \text{“d”} \Rightarrow \bigcirc flag(x, C\_mb) \equiv \text{“r”} \wedge \bigcirc\Diamond flag(x, C\_mb) \equiv \text{“d”}$

and then apply rule $rule2$:

$$rule2 \quad \frac{\Gamma \Rightarrow \psi \wedge \Diamond[\bigcirc\phi \wedge \neg\bigcirc F], \Delta}{\Gamma \Rightarrow \psi \wedge \bigcirc\Diamond\phi, \Delta}$$

reaching sequent (7):

(7) $\mathsf{Plan}, flag(x, C\_mb) \not\equiv \text{“d”} \Rightarrow \bigcirc flag(x, C\_mb) \equiv \text{“r”} \wedge \Diamond[\bigcirc flag(x, C\_mb) \equiv \text{“d”} \wedge \neg\bigcirc F]$

Applying rule *rule*1 and some equivalence transformations sequent (8) and two assertions are reached. Simultaneously, we introduced a structure into the metavariable Plan by the assumption $\mathsf{Plan} \leftrightarrow \mathsf{P}_1;\mathsf{P}_2$.

(8) $flag(x, C\_mb) \not\equiv$ "d", $\mathsf{P}_1;\mathsf{P}_2 \Rightarrow \quad \bigcirc flag(x, C\_mb) \equiv$ "r" $\wedge \bigcirc\bigcirc flag(x, C\_mb) \equiv$ "d"

and the two assertions

(8') $\mathsf{P}_1;\mathsf{P}_2, flag(x, C\_mb) \not\equiv$ "d" $\Rightarrow \quad \neg\bigcirc F$

(8") $\mathsf{P}_1;\mathsf{P}_2, flag(x, C\_mb) \not\equiv$ "d" $\Rightarrow \quad \bigcirc\neg\bigcirc F$

On sequent (8) we apply rule *rule*3:

$$rule3 \quad \frac{\Gamma \Rightarrow \bigcirc\phi \wedge \bigcirc\bigcirc F;\bigcirc\psi, \Delta}{\Gamma \Rightarrow \bigcirc\phi \wedge \bigcirc\bigcirc\psi, \Delta} \qquad \text{with } \phi \text{ first-order formula}$$

yielding

(9) $\mathsf{P}_1;\mathsf{P}_2, flag(x, C\_mb) \not\equiv$ "d" $\Rightarrow \quad \bigcirc flag(x, C\_mb) \equiv$ "r" $\wedge \bigcirc\bigcirc F;\bigcirc flag(x, C\_mb) \equiv$ "d"

At this point in the proof construction it becomes necessary to make the connection between P_1 and P_2 more concrete using rule *rule*4:

$$rule4 \quad \frac{\phi, P;\psi \wedge Q \Rightarrow \Gamma \qquad \phi, P \Rightarrow \Diamond[\bigcirc F \wedge \psi]}{\phi, P;Q \Rightarrow \Gamma}.$$

The intention in using this rule is to force P_1 to cause an effect which is the precondition of P_2. Sequents (10) and (11) are the result of applying *rule*4 to sequent (9).

(10) $flag(x, C\_mb) \not\equiv$ "d", $\mathsf{P}_1;\mathsf{pre} \wedge \mathsf{P}_2 \Rightarrow \quad \bigcirc flag(x, C\_mb) \equiv$ "r" $\wedge \bigcirc\bigcirc F;$
$\bigcirc flag(x, C\_mb) \equiv$ "d"

(11) $flag(x, C\_mb) \not\equiv$ "d", $\mathsf{P}_1 \Rightarrow \Diamond[\bigcirc F \wedge \mathsf{pre}]$

Note, that we have introduced also a metavariable for a precondition here that has appropriately to be instantiated in the sequel.

First we go on with sequent (10) and split it with the *chop composition* rule above to get (12) and (13).

(12) $\mathsf{P}_1, flag(x, C\_mb) \not\equiv$ "d" $\Rightarrow \bigcirc flag(x, C\_mb) \equiv$ "r" $\wedge \bigcirc\bigcirc F$

(13) $\mathsf{pre} \wedge \mathsf{P}_2 \Rightarrow \bigcirc flag(x, C\_mb) \equiv$ "d"

Sequent (12) can be splitted into the sequents

(14) $\mathsf{P}_1, flag(x, C\_mb) \not\equiv$ "d" $\Rightarrow \bigcirc flag(x, C\_mb) \equiv$ "r"

(15) $\mathsf{P}_1, flag(x, C\_mb) \not\equiv$ "d" $\Rightarrow \bigcirc\bigcirc F$

Closing one part of the proof tree can be achieved by instantiating P_1 in sequent (14) with the predicate $EX(type(x, C\_mb))$. The resulting sequent is then an instance of the nonlogical axiom for the "type" command known from above.

The branch in sequent (15) can also be closed, because (15) only demands that P_1 is a plan of length one.

In sequent (13) the metavariables pre and P_2 can be instantiated with $flag(x, C\_mb) \not\equiv$ "d" and $EX(delete(x, C\_mb))$, respectively. Having carried out that substitution this part of the proof tree can also be closed because we end with a nonlogical axiom that is an instance of the "delete" command axiomatization. Now, all metavariables are instantiated and their substitution can be propagated through the proof tree. Sequents (8') and (8") also close branches of the proof tree because they only demand that plan Plan is not empty. The last

branch remaining still open is that ending in sequent (11); with the substitutions found it looks like:

(11') $EX(type(x, C\_mb)), flag(x, C\_mb) \not\equiv$ "d" $\Rightarrow \Diamond[\circ F \wedge flag(x, C\_mb) \not\equiv$ "d"]

Sequent (11') can easily be proved using rule *rule*1 and an appropriate frame instance of the "type" axiom which says that the property $flag(x, C\_mb) \not\equiv$ "d" is not destroyed by executing "type(x,C_mb)".

Then, all branches of the proof tree are closed and the resulting plan, the substitution for the metavariable Plan in (1), is:

$$EX(type(x, C\_mb)); EX(delete(x, C\_mb)).$$

The deductive planning system currently under implementation provides automatic strategies to guide the plan generation process according to the current specification. These strategies - besides those for sequential plans also strategies for deriving conditional and *while*-plans have been developed [BD91] - are implemented using concepts from tactical theorem proving.

5 Deductive Plan Reuse

Once a plan is generated it represents problem solving knowledge which is generally lost in classical planning systems after the plan has been successfully executed. Methods of *planning from second principles* try to reuse former problem solutions in order to make planning more efficient and flexible. In this section we demonstrate how plan reuse can be performed deductively.

5.1 The 4-Phase Model

To formalize planning from second principles a four-phase model of plan reuse has been proposed in [Koe91]:

1. In the *Plan Determination* phase a plan specification formula Φ is retrieved from the plan library to solve a new planning problem given as a plan specification formula Ψ.
2. In the phase of *Plan Interpretation* the formula Φ has to be interpreted in the current planning situation by investigating whether Φ can be instantiated to Φ_{inst} such that Ψ is obtained.
3. In the *Plan Refitting* phase the instantiated plan specification Φ_{inst} is compared with Ψ and refitting tasks for the planner are derived. Planner and plan reuse component interact in such a way that the reuse component generates subplan specifications for which the planner is activated to generate the subplans which have to be deleted from or incorporated into the plan to be reused.
4. The reuse process ends with a *Plan Library Update* in which the plan specification formula Ψ is generalized and compared with already stored plans. If Ψ is "worth" storing it is added to the plan library.

In the following we describe how plan interpretation and refitting, summarized as *plan modification* are realized deductively and demonstrate our method by means of an example.

5.2 A Deductive Approach to Plan Modification

For the following we assume that plan specification formulas $[\mathsf{Plan}_\psi \rightarrow \psi]$ are of form $[\mathsf{Plan}_\psi \rightarrow [\psi_i \rightarrow \psi_g]]$, where the subformulas ψ_i and ψ_g describe the facts holding before executing the plan and the facts that have to be reached by it, respectively.
Suppose, given a plan specification $[\mathsf{Plan}_\psi \rightarrow \psi]$ the plan determination process succeeds in finding an appropriate entry in the plan library and comes up with a specification formula $[\mathsf{Plan}_\phi \rightarrow \phi]$ and a plan formula P_ϕ that had been generated from this specification to replace the metavariable Plan_ϕ. To find out whether P_ϕ can be reused to replace even Plan_ψ in order to satisfy the current specification we try to prove the formula:

$$[\phi \rightarrow \psi]$$

This step is justified by the fact that $[P_\phi \rightarrow \psi]$ if $[\phi \rightarrow \psi]$, provided $[P_\phi \rightarrow \phi]$ holds.
If the proof of $[\phi \rightarrow \psi]$ succeeds the "old" plan P_ϕ can be reused without any modifications. If the proof fails information for successfully modifying P_ϕ can be extracted from it. $[\phi \rightarrow \psi]$ is attempted to be proved using a matrix calculus based on the connection method introduced by Bibel [Bib82] which has been extended to certain modal logics by Wallen in [Wal89]. He has extended the concept of complementary literals by considering even the modal context, i.e., the modal operators in the scope of which these literals occur. Modal contexts are represented by so-called *prefixes* of the literals concerned. They can be viewed as strings denoting possible worlds, or, in our case, intervals. Wallen then defines two literals to be *simultaneously complementary* iff they are first-order complementary and additionally their prefixes unify according to a modal substitution reflecting the property of the accessibility relation on worlds (intervals), cf. [Wal89].
We distinguish between constant and variable atomic prefixes and denote them by $\overline{a}_i$ and a_i, respectively. Suppose, we have $\sigma R^* \sigma'$ and $\sigma' R^* \sigma''$ for intervals σ, σ', and σ'' and consider the following correspondence between prefixes and these intervals:
$\overline{a}_0\overline{a}_1\overline{a}_2$; σ, $\overline{a}_1\overline{a}_2$; σ', $\overline{a}_2$; σ''.
According to the accessibility relation R^* the prefix $a_3\overline{a}_2$ can then denote any interval from which σ'' can be reached. Consequently, our prefix substitution function allows to map, for example, a_3 to $\overline{a}_0\overline{a}_1$.

5.3 Example

Suppose, we have to construct a plan to *"Read a mail, save it in a file, and then delete the mail"*, formally specified by the following formula:
$[\mathsf{Plan}_\psi \rightarrow \psi]$, where ψ abbreviates
$[flag(y, C\_mb) \not\equiv$ "d" $\rightarrow$
$\Diamond[flag(y, C\_mb) \equiv$ "r" $\wedge \Diamond[flag(y, C\_mb) \equiv$ "s" $\wedge \Diamond flag(y, C\_mb) \equiv$ "d"$]]]$
And suppose, the plan determination process having analyzed this specification formula comes up with a reuse candidate we know from the example in Section 3: the specification formula $[\mathsf{Plan}_\phi \rightarrow \phi]$ with ϕ abbreviating
$[flag(x, C\_mb) \not\equiv$ "d" $\rightarrow \Diamond[flag(x, C\_mb) \equiv$ "r" $\wedge \Diamond flag(x, C\_mb) \equiv$ "d"$]]$
and the plan formula P_ϕ: $EX(type(x, C\_mb)); EX(delete(x, C\_mb))$.

The plan modification process then starts with trying to prove formula $[\phi \rightarrow \psi]$, i.e.,

$[flag(x, C\_mb) \not\equiv$ "d" $\rightarrow \Diamond[flag(x, C\_mb) \equiv$ "r" $\wedge \Diamond flag(x, C\_mb) \equiv$ "d"]]
$\rightarrow [flag(y, C\_mb) \not\equiv$ "d" $\rightarrow$
$\Diamond[flag(y, C\_mb) \equiv$ "r" $\wedge \Diamond[flag(y, C\_mb) \equiv$ "s" $\wedge \Diamond flag(y, C\_mb) \equiv$ "d"]]].

The proof attempt consists of two steps.

First the matrix corresponding to that formula has to be built. Each matrix element consists of a prefixed literal (described by a prefixed atom and a sign $\in \{0,1\}$) and a label indicating whether the literal belongs to the i(nitial)- or g(oal)-part of one of the specification formulas, respectively.

Following Wallen the matrix representation of the formula is obtained by applying certain sequence rules in turn to eliminate logical and modal operators until no non-atomic formulas are left. The sequence rules used for building the matrix of a formula have to have the so-called *subformula property* [Wal89]. Prefixes are introduced when we apply rules that introduce modal operators. Applying, for example the rule

$$\frac{\Gamma \Rightarrow A, \Delta}{\Gamma \Rightarrow \Diamond A, \Delta}$$

to the sequent $a_0 a'(A) \Rightarrow a_0 a'(\Diamond B)$ leads to $a_0 \overline{a}_1 a''(A) \Rightarrow \overline{a}_1 a''(B)$, where a' and a'' are metavariables for (even empty) prefixes.

Proceeding in this way the matrix we finally obtain for our formula above consists of the following paths:

Path 1: $\{< a_1 a_2 flag(x, C\_mb) \equiv$ "r", $\phi_g, 1 >, < a_2 flag(x, C\_mb) \equiv$ "d", $\phi_g, 1 >,$
$< a_0 \overline{a}_1 \overline{a}_2 \overline{a}_3 flag flag(y, C\_mb) \not\equiv$ "d", $\psi_i, 1 >, < a_0 a_1 a_2 flag(x, C\_mb) \not\equiv$ "d", $\phi_i, 0 >\}$

Path 2: $\{< a_1 a_2 flag(x, C\_mb) \equiv$ "r", $\phi_g, 1 >, < a_2 flag(x, C\_mb) \equiv$ "d", $\phi_g, 1 >,$
$< a_0 \overline{a}_1 \overline{a}_2 \overline{a}_3 flag(y, C\_mb) \not\equiv$ "d", $\psi_i, 1 >, < \overline{a}_1 \overline{a}_2 \overline{a}_3 flag(y, C\_mb) \equiv$ "r", $\psi_g, 0 >\}$

Path 3: $\{< a_1 a_2 flag(x, C\_mb) \equiv$ "r", $\phi_g, 1 >, < a_2 flag(x, C\_mb) \equiv$ "d", $\phi_g, 1 >,$
$< a_0 \overline{a}_1 \overline{a}_2 \overline{a}_3 flag(y, C\_mb) \not\equiv$ "d", $\psi_i, 1 >, < \overline{a}_2 \overline{a}_3 flag(y, C\_mb) \equiv$ "s", $\psi_g, 0 >\}$

Path 4: $\{< a_1 a_2 flag(x, C\_mb) \equiv$ "r", $\phi_g, 1 >, < a_2 flag(x, C\_mb) \equiv$ "d", $\phi_g, 1 >,$
$< a_0 \overline{a}_1 \overline{a}_2 \overline{a}_3 flag(y, C\_mb) \not\equiv$ "d", $\psi_i, 1 >, < \overline{a}_3 flag(y, C\_mb) \equiv$ "d", $\psi_g, 0 >\}$

In the second step we have to determine the paths that contain simultaneously complementary literals. To do this we consider the following pairs of elements in the paths:

In path 1: $\{< a_0 \overline{a}_1 \overline{a}_2 \overline{a}_3 flag(y, C\_mb) \not\equiv$ "d", $\psi_i, 1 >,$
$< a_0 a_1 a_2 flag(x, C\_mb) \not\equiv$ "d", $\phi_i, 0 >\}$

In path 2: $\{< a_1 a_2 flag(x, C\_mb) \equiv$ "r", $\phi_g, 1 >, < \overline{a}_1 \overline{a}_2 \overline{a}_3 flag(y, C\_mb) \equiv$ "r", $\psi_g, 0 >\}$

In path 4: $\{< a_2 flag(x, C\_mb) \equiv$ "d", $\phi_g, 1 >, < \overline{a}_3 flag(y, C\_mb) \equiv$ "d", $\psi_g, 0 >\}$

They are complementary under the first order substitution $\rho = \{y/x\}$ and the modal substitution $\rho_M = \{\overline{a}_3/a_2, \overline{a}_1 \overline{a}_2/a_1\}$.

The complementary paths 1, 2 and 4 describe a valid formula $[\phi \rightarrow \psi']$ that can be constructed from $[\phi \rightarrow \psi]$ where ψ' is a part of ψ, i.e., the specification of P_ϕ, the plan to be reused, contains not all of the subgoals of the new specification ψ. This is found out as follows:

The complementary literals above are characterized as pairs of kind (ϕ_i, ψ_i) or (ϕ_g, ψ_g), respectively. For the remaining path *path 3* no complementary connection can be found and that causes the proof of $[\phi \rightarrow \psi]$ to fail. At this point the refitting phase starts.

Since ψ' is a part of ψ and the only literal in the matrix which is not part of a complementary connection is of kind ψ_g this indicates the "difference" between the two plan specifications: Compared to the current one there is a subgoal ϕ_g "missing" in the plan specification for

P_ϕ. As a consequence an additional subplan P_{new} has to be included into P_ϕ reaching P'_ϕ which is then the desired substitution for Plan_ψ.
Therefore, from *path 3* we extract literals of kind ψ_i and ψ_g and build a specification formula for the remaining subgoal.
The literals are:
$< a_0\overline{a}_1\overline{a}_2\overline{a}_3 flag(y, C\_mb) \not\equiv$ "d", $\psi_i, 1 >$ *and* $< \overline{a}_2\overline{a}_3 flag(y, C\_mb) \equiv$ "s", $\psi_g, 0 >$,
and the specification formula then reads:
$\mathsf{Plan}_{new} \rightarrow [flag(y, C\_mb) \not\equiv$ "d" $\rightarrow \Diamond flag(y, C\_mb) \equiv$ "s"$]$
The plan generation process produces P_{new}=$EX(save(y, file, C\_mb))$ as a substitution for Plan_{new}.
From the modal substitution information in ρ_M according to the correspondence between prefixes and intervals, the literals of kind ϕ_g and ψ_g in *path 3*, and the relation between formula ϕ and the Plan P_ϕ known from the plan generation process the position in P_ϕ where P_{new} has to be included can be derived. With that we reach the modified plan $P'_\phi = P_\psi$ as:

$$P_\psi = EX(type(y, C\_mb)); EX(save(y, file, C\_mb)); EX(delete(y, C\_mb))$$

It finally has to be verified that the modification of P_ϕ leads to a correct (i.e. executable) plan. That means we have to prove whether the effects of *type* and *save* imply the preconditions of *save* and *delete*, respectively.

6 Conclusion

We have introduced a deductive planning system that realizes planning from first as well as planning from second principles.
The system is intended to supply the kernel of an intelligent help system with a planning component. Hence, the planning domain is a command language, namely the language of the application system for which the help is provided. Planning in this command language environment suggests to view plans as programs and with that follow the *plans are programs* paradigm. As a consequence, deductive planning in this context is based on a programming logic.
We have introduced the *logical language for planning* LLP, an interval-based temporal logic that provides control structures, like ;, *if then else*, and *while*. An appropriate axiomatization of the application domain is obtained by describing basic actions like assignments in programming languages. With that we have only *one* axiom schema for each basic action characterizing both, its effects as well as the (*frame*) properties that are not affected by the action. Thus, the frame problem is addressed in a representational way.
Different kinds of plan specifications can be formulated in terms of special LLP formulas. Plans are then obtained by proving the specification formulas using a sequence calculus for LLP. The search for proofs is guided by several strategies that are implemented using concepts from the field of tactical theorem proving.
Planning from second principles is done by trying to reuse plans stored in a plan library. We have proposed a method for modifying a given reuse candidate in such a way that it also satisfies the current specification. The modification process is based on a special kind of subsumption test. If the test succeeds the plan can be reused without any modification. Otherwise, information is extracted from the failed proof and used to formally specify the modifications that have to be done.

References

[BBD+91] M. Bauer, S. Biundo, D. Dengler, M. Hecking, J. Köhler, and G. Merziger. Integrated plan generation and recognition: A logic-based approach. In *Proceedings of the 4. Internationaler GI-Kongress Wissensbasierte Systeme, München*, pages 266–277. Springer IFB 291, 1991.

[BD91] S. Biundo and D. Dengler. An interval-based temporal logic for planning. Research report, German Research Center for Artificial Intelligence, Saarbrücken, 1991. also submitted to the 1st International Conference on AI Planning Systems.

[Bib82] W. Bibel. *Automated Theorem Proving.* Vieweg, Braunschweig, 1982.

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

[Gre69] C. Green. Application of theorem proving to problem solving. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, pages 219–239, 1969.

[Har79] D. Harel. *First Order Dynamic Logic.* Springer LNCS 68, New York, 1979.

[Koe91] J. Koehler. Approaches to the reuse of plan schemata in planning formalisms. Technical Memo TM-91-01, German Research Center for Artificial Intelligence, January 1991.

[Kow79] R. Kowalski. *Logic for Problem Solving.* North-Holland Publishing Company, Amsterdam, New York, Oxford, 1979.

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

[MW87] Z. Manna and R. Waldinger. How to clear a block: Plan formation in situational logic. *Journal of Automated Reasoning*, 3:343–377, 1987.

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

[Wal89] L. A. Wallen. *Automated Deduction in Non-classical Logics.* MIT-Press, Cambridge, London, 1989.

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

DFKI
-Bibliothek-
PF 2080
D-6750 Kaiserslautern
FRG

DFKI Publikationen

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

DFKI Publications

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

DFKI Research Reports

RR-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
39 pages

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

RR-91-33
Franz Baader, Klaus Schulz:
Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures
33 pages

RR-91-34
Bernhard Nebel, Christer Bäckström:
On the Computational Complexity of Temporal Projection and some related Problems
35 pages

RR-91-35
Winfried Graf, Wolfgang Maaß: Constraint-basierte Verarbeitung graphischen Wissens
14 Seiten

RR-92-02
Andreas Dengel, Rainer Bleisinger, Rainer Hoch, Frank Hönes, Frank Fein, Michael Malburg:
Π_{ODA}: The Paper Interface to ODA
53 pages

RR-92-03
Harold Boley:
Extended Logic-plus-Functional Programming
28 pages

RR-92-04
John Nerbonne: Feature-Based Lexicons:
An Example and a Comparison to DATR
15 pages

RR-92-05
Ansgar Bernardi, Christoph Klauck,
Ralf Legleitner, Michael Schulte, Rainer Stark:
Feature based Integration of CAD and CAPP
19 pages

RR-92-07
Michael Beetz:
Decision-theoretic Transformational Planning
22 pages

RR-92-08
Gabriele Merziger: Approaches to Abductive
Reasoning - An Overview -
46 pages

RR-92-11
Susane Biundo, Dietmar Dengler, Jana Koehler:
Deductive Planning and Plan Reuse in a Command
Language Environment
13 pages

RR-92-15
Winfried Graf: Constraint-Based Graphical Layout
of Multimodal Presentations
23 pages

DFKI Technical Memos

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 (Ed.):
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

TM-91-14
Rainer Bleisinger, Rainer Hoch, Andreas Dengel:
ODA-based modeling for document analysis
14 pages

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

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

DFKI Documents

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

D-91-02
Jörg P. Müller: Design and Implementation of a Finite Domain Constraint Logic Programming System based on PROLOG with Coroutining
127 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 Seiten

D-91-09
David Powers, 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 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

D-91-15
Harold Boley, Philipp Hanschke, Martin Harm, Knut Hinkelmann, Thomas Labisch, Manfred Meyer, Jörg Müller, Thomas Oltzen, Michael Sintek, Werner Stein, Frank Steinle:
µCAD2NC: A Declarative Lathe-Worplanning Model Transforming CAD-like Geometries into Abstract NC Programs
100 pages

D-91-16
Jörg Thoben, Franz Schmalhofer, Thomas Reinartz:
Wiederholungs-, Varianten- und Neuplanung bei der Fertigung rotationssymmetrischer Drehteile
134 Seiten

D-91-17
Andreas Becker:
Analyse der Planungsverfahren der KI im Hinblick auf ihre Eignung für die Abeitsplanung
86 Seiten

D-91-18
Thomas Reinartz: Definition von Problemklassen im Maschinenbau als eine Begriffsbildungsaufgabe
107 Seiten

D-91-19
Peter Wazinski: Objektlokalisation in graphischen Darstellungen
110 Seiten

Deductive Planning and Plan Reuse in a Command Language Environment

Susanne Biundo, Dietmar Denger, Jana Koehler

RR-92-11
Research Report