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, FRG, e-mail: <last name>@dfki.uni-sb.de

Abstract

We introduce a deductive planning system intended to supply intelligent help systems. It consists of a deductive planner and a plan reuse component, providing planning from first as well as planning from second principles. Both components rely on an interval-based temporal logic. The deductive formalisms realizing plan formation from formal specifications and the reuse of already existing plans are presented and demonstrated by examples taken from the domain of operating systems.

1 Introduction

Intelligent help systems aim at supporting users of complex software systems. Advanced active help can 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 he pursues. On the other hand, plans have to be generated and supplied to the user based on this information, thus enabling him to reach these goals properly.

Consequently, the PHI-System [1] currently being developed as the kernel of an intelligent help system provides both a *plan recognizer* to anticipate the user's goals and a *plan generation component* that supports the user by providing plans to reach these goals.

One of PHI's main characteristics is the close mutual cooperation between the plan recognition and plan generation components. One mode of cooperation uses (abstract) *plans* as a basis for plan recognition: Starting from a formal plan specification the generation component produces a set of hypothetical plans. These hypotheses are used to identify the user's plan by trying to map the observed actions on an instance of any of the plan hypotheses. *Abstract* plans are those that contain variables, abstract commands, and control structures. 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. [2] and [9]), 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* differs in several aspects from the deductive planning approaches known from the literature (cf. [4], [7], [2], and [9]). It relies on an interval-based temporal logic that combines features of traditional programming logics, such as, e.g., *Dynamic Logic* [5] or *Temporal Logic of Programs* [8]. 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*.

2 The Planning System

The planning system, shown in Figure 1, consists of a *deductive plan generator*, the *reuse component*, and a *plan interpreter*. 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 from second principles. It works in the following way:

A formal plan specification Φ given to the deductive planner is forwarded to the reuse component. If the reuse component succeeds in finding a plan in the library that (perhaps after minor modifications) can be used to solve Φ, the *plan modification* process starts. 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*) out of the library. If the plan has to

*This paper has been published in the Proceedings of the 10th European Conference on Artificial Intelligence, Ed. by B. Neumann, pages 628-632, John Wiley & Sons, Chichester, New York, 1992.

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 *constructive proof* of the specification formula. A so-called *plan formula* results that represents the specified plan. Besides linear plans, which are sequences of basic actions, even conditional and *while*-plans can be derived [3]. If executable plans are required to be produced in a certain situation, the plan interpreter is finally activated to eliminate these control structures if necessary.

Figure 1: The Planning System

3 The Logical Framework

The *logical language for planning* (LLP) we have developed to realize deductive planning in our help system context is an interval-based modal temporal logic that combines features of *Choppy Logic* [10] with the *Temporal Logic for Programs* [8]. LLP relies on a many-sorted first-order language and, besides the normal logical variables, 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 changing the values of certain local variables.
The modal operators provided by LLP are $\circ$(next), $\Diamond$ (sometimes), $\Box$ (always), and a sequential composition of formulas by the two-place modal operator ; (chop). Besides these operators *control structures* are also available, as in programming logics. 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 in a similar way. 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*. These *plan formulas* are *basic plan formulas*, i.e., $EX(c)$, where c is a term of type *command*, sequential plans $\alpha;\beta$, where α and β are plan formulas. Conditional and *while*-plans are omitted for lack of space.

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 operators $\{\neg, \wedge, \forall, \circ, \Box, ;\}$. We use the abbreviations $\Diamond\phi \leftrightarrow \neg\Box\neg\phi$, $[\phi \rightarrow \psi] \leftrightarrow [\neg[\phi \wedge \neg\psi]]$, and $\neg\,\mathrm{T} \leftrightarrow \mathrm{F}$.

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: $\sigma \circ \sigma' = \sigma$, if σ is infinite, and $< \sigma_0 \ldots \sigma_n \ldots >$

if $\sigma = < \sigma_0 \ldots \sigma_n >$ and $\sigma' = < \sigma_n \ldots >$.
We call the triple (W, R, $\circ$) a *frame* and the six-tuple (D, I, β, W, R, $\circ$) a Σ-*interpretation*. 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 $x \in VG$, and $\mathcal{I}_\Sigma(a) = \sigma_0(a)$ for every $a \in VL$. The interpretation is extended in the usual way to function expressions.
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:
$\sigma \models_\mathcal{I}$ T, $\sigma \models_\mathcal{I} EX(t)$ iff $\mathcal{I}_\Sigma(t) = \sigma_0^2$
$\sigma \models_\mathcal{I} \circ\phi$ 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 $\sigma'' \models_\mathcal{I} \psi$

Calculus

The calculus we use for LLP is based on the complete sequent calculus for S4 modal logic given in [11]. We have extended this calculus by giving additional rules for handling the modalities $\circ$, ; and *while*. Due to lack of space we present only the *next* and the *chop composition* rules and introduce other basic as well as derived rules when we use them in the examples. Remember that a *sequent* 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: $\dfrac{\Gamma^* \Rightarrow \phi, \Delta^*}{\Gamma \Rightarrow \circ\phi, \Delta}$ with

  $\Gamma^* = \{\psi \mid \circ\psi \in \Gamma\} \cup \{\Box\psi \mid \Box\psi \in \Gamma\}$, and
  $\Delta^* = \{\psi \mid \circ\psi \in \Delta\} \cup \{\Diamond\psi \mid \Diamond\psi \in \Delta\}$

- *chop composition*-rule: $\dfrac{\phi_1 \Rightarrow \psi_1 \quad \phi_2 \Rightarrow \psi_2}{\phi_1;\phi_2 \Rightarrow \psi_1;\psi_2}$

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 behavior by the use of local variables for identifying objects of type *mailbox* or *message*. The axioms describing the different mail commands as basic actions are given like axioms for assignment statements in programming logics. The "type" command for reading a message, for example, is given by the following axiom scheme:

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

The symbol P is a *metavariable* for formulas; the substitution instructions correspond to the *effect* of the "type" command: "type" does nothing other than change the *flag* of the i-th message in C_mbox to "r" and increases the $Current$-counter by 1. Using the "type" axiom during the plan generation process is done by building an appropriate instance of the above axiom schema and applying it to the actual sequent.
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 have also obtained a representational solution of the frame problem [3].
Basic actions are required to terminate. This fact is expressed by special axioms. We have:

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

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 sequent proof that provides appropriate instantiations for these variables. Based on the specification we develop a proof tree applying several sequent 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 start from the specification formula and 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* [8]). They read $\mathsf{Plan} \rightarrow [\phi_i \rightarrow \Diamond\phi_g]$ stating that ϕ_g holds at 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 and delete it". Then the input to the plan generator is the formula:

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

Now a sequent proof of formula (1) will be constructed 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) $\mathsf{Plan},\ flag(x,C) \not\equiv$ "d" $\Rightarrow$
$\Diamond[flag(x,C) \equiv$ "r" $\wedge \Diamond flag(x,C) \equiv$ "d"$]$

Applying $rule1$: $\dfrac{\Gamma \Rightarrow \circ\phi \wedge \neg\circ F, \Delta}{\Gamma \Rightarrow \Diamond\phi, \Delta}$
and the standard $\wedge{:}right$ rule we obtain sequents (3) and (4):

(3) $\mathsf{Plan}, flag(x,C) \not\equiv$ "d" $\Rightarrow$
$\circ[flag(x,C) \equiv$ "r" $\wedge \Diamond flag(x,C) \equiv$ "d"$]$

(4) $\mathsf{Plan}, flag(x,C) \not\equiv$ "d" $\Rightarrow \neg\circ F$

Sequent (4) represents a so-called *assertion* stating that Plan is not the empty plan. While this assertion can only be proved when an instantiation of Plan has been found, sequent (3) is the subgoal we proceed with to obtain this instantiation. We apply the distributivity law of *next* and then $rule2$: $\dfrac{\Gamma \Rightarrow \psi \wedge \Diamond[\circ\phi \wedge \neg\circ F], \Delta}{\Gamma \Rightarrow \psi \wedge \circ\Diamond\phi, \Delta}$ to reach sequent (5):

(5)
$\mathsf{Plan}, flag(x,C) \not\equiv$ "d" $\Rightarrow$
$\circ flag(x,C) \equiv$ "r"$\wedge \Diamond[\circ flag(x,C) \equiv$ "d" $\wedge \neg\circ F]$

Applying $rule1$ and some equivalence preserving transformations, sequent (6) is obtained. Note that we have also introduced a structure for the plan to be generated: Plan has been replaced by formula $\mathsf{P}_1;\mathsf{P}_2$. The intention behind this step is that we have to reach a conjunction of goals. We plan to solve the first conjunct by P_1 and the second by P_2, respectively.

(6) $flag(x,C) \not\equiv$ "d", $\mathsf{P}_1;\mathsf{P}_2 \Rightarrow$
$\circ flag(x,C) \equiv$ "r" $\wedge \circ\circ flag(x,C) \equiv$ "d"

$rule3$: $\dfrac{\Gamma \Rightarrow \circ\phi \wedge \circ\circ F;\circ\psi, \Delta}{\Gamma \Rightarrow \circ\phi \wedge \circ\circ\psi, \Delta}$ (ϕ first-order) now yields

(7) $\mathsf{P}_1;\mathsf{P}_2, flag(x,C) \not\equiv$ "d" $\Rightarrow$
$\circ flag(x,C) \equiv$ "r" $\wedge \circ\circ F;\circ flag(x,C) \equiv$ "d"

At this point in the proof construction it becomes necessary to make the connection between P_1 and P_2 more concrete by forcing P_1 to install the preconditions to be required for P_2. This is done using

$rule4$: $\dfrac{\phi, P;\psi \wedge Q \Rightarrow \Gamma \qquad \phi, P \Rightarrow \Diamond[\circ F \wedge \psi]}{\phi, P;Q \Rightarrow \Gamma}$

We obtain

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

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

Note that we have introduced the metavariable pre for the preconditions of P_2. pre has to be appropriately instantiated as the proof proceeds.

We turn to sequent (8) applying the *chop composition* rule above and the $\wedge{:}right$ rule to get (10), (11) and an additional assertion we omit for lack of space.

(10) $\mathsf{pre} \wedge \mathsf{P}_2 \Rightarrow \circ flag(x,C) \equiv$ "d"

(11) $\mathsf{P}_1, flag(x,C) \not\equiv$ "d" $\Rightarrow \circ flag(x,C) \equiv$ "r"

Closing one part of the proof tree can be achieved by instantiating P_1 in (11) with formula $EX(type(x,C))$. The resulting sequent is an instance of the axiom for the "type" command mentioned above.
In sequent (10) the metavariables pre and P_2 can be instantiated with $flag(x,C) \not\equiv$ "d" and $EX(delete(x,C))$, respectively. Having carried out that substitution, this part of the proof tree can also be closed because we end up with an instance of the axiom scheme for "delete". Now, all metavariables are instantiated and their substitution can be propagated through the proof tree.
With that the remaining assertions about the plan can now be easily proved. The substitution we have finally obtained for Plan reads:
$EX(type(x,C));EX(delete(x,C))$.
The deductive planning system currently under implementation provides automatic strategies to guide the plan generation process. These strategies are implemented using concepts from tactical theorem proving.

4 Deductive Plan Reuse

Once a plan is generated it represents problem solving knowledge that 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 [6] we have proposed a four-phase model of plan reuse. This section is devoted to plan modification that is indispensible if plans have to be reused. We introduce a deductive method and demonstrate it by means of an example.
Plan specification formulas are of the form $[\mathsf{Plan}_\psi \rightarrow [\psi_i \rightarrow \Diamond\psi_g]]$, where the subformulas ψ_i and ψ_g describe facts holding in the initial state before executing the plan, and in the goal state that is to be reached, respectively.
Suppose we intend to reuse a plan represented as the plan formula P_ϕ to solve the plan specification $[\mathsf{Plan}_\psi \rightarrow \psi]$. P_ϕ had been generated from a specification formula $[\mathsf{Plan}_\phi \rightarrow \phi]$ 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 $\phi_g \rightarrow \psi_g$ and $\psi_i \rightarrow \phi_i$ using a restricted substitution ρ with $DOM(\rho) \subseteq VAR(\phi)$ and

$COD(\rho) \subseteq TERM(\psi)$. If the proof succeeds, $\phi \rightarrow \psi$ holds and the "old" plan P_ϕ can be reused without any modifications. If it fails, information for successfully modifying P_ϕ can be extracted from it.

We use a matrix proof method for modal logics embedded into a sequent calculus [11]. Therefore, certain sequent rules are applied in turn to eliminate logical and modal operators until no non-atomic formulas are left. To represent modalities, so-called *prefixes* are introduced for every literal in the matrix. They can be viewed as strings denoting intervals. Two literals are defined to be *simultaneously complementary* iff they are first-order complementary and their prefixes unify according to a modal substitution reflecting the accessibility relation on intervals.

Suppose we have to construct a plan for the goal *"Read a mail, save it, and then delete the mail"*, formally specified by the following formula:

$[\mathsf{Plan}_\psi \rightarrow \psi]$, where ψ abbreviates

$[flag(y,C) \not\equiv$ "d" $\rightarrow \Diamond[flag(y,C) \equiv$ "r" $\wedge$

$\Diamond[flag(y,C) \equiv$ "s" $\wedge \Diamond flag(y,C) \equiv$ "d"$]]]$ and suppose we want to reuse the candidate known from the example in Section 3: the specification formula $[\mathsf{Plan}_\phi \rightarrow \phi]$ with ϕ abbreviating

$[flag(x,C) \not\equiv$ "d" $\rightarrow \Diamond[flag(x,C) \equiv$ "r" $\wedge$

$\Diamond flag(x,C) \equiv$ "d"$]]$ and the plan formula P_ϕ:

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

In this example no substitution ρ can be found that makes all paths in the matrix simultaneously complementary. Therefore, the proof fails and we have to start the modification of the plan to be reused. As a starting point for plan modification, we choose a substitution ρ that leads to a maximal number of simultaneously complementary paths. They describe a valid formula $[\phi \rightarrow \psi']$ where ψ' is a subformula of ψ. The "difference" between the two plan specifications is indicated by the syntactic structure of the formula represented by the noncomplementary path under ρ: 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, we build the following specification formula for the remaining subgoal:

$\mathsf{Plan}_{new} \rightarrow [flag(y,C) \not\equiv$ "d" $\rightarrow \Diamond flag(y,C) \equiv$ "s"$]$

The plan generation process produces

P_{new}=$EX(save(y,C))$ as a substitution for Plan_{new}. Now we have to determine where in P_ϕ the subplan P_{new} has to be positioned. It can be derived using the modal substitution information in ρ according to the correspondence between prefixes and intervals and the temporal ordering of actions in P_ϕ known from the plan generation process. With that we reach the modified plan $P'_\phi = P_\psi$ as:

$EX(type(y,C)); EX(save(y,C)); EX(delete(y,C))$.

Finally, it has to be verified that the modification of P_ϕ leads to an executable plan.

5 Conclusion

We have introduced a deductive planning system intended to supply intelligent help systems. Due to the specific planning domain plans are viewed as programs. Consequently, we have presented the temporal programming logic LLP that provides control structures for plans as well as an axiomatization of the application domain that is apt to address the frame problem in a quite convenient way. Besides realizing deductive planning from first principles, we have proposed a method for modifying an already existing plan in such a way that it can be reused for a new specification. The modification process is based on a proof attempt. If it 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 made.

References

[1] M. Bauer, S. Biundo, D. Dengler, M. Hecking, J. Koehler, and G. Merziger. Integrated plan generation and recognition: A logic-based approach. In *Proc. 4. Int. GI-Kongress,* Springer IFB 291, 1991.

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

[3] S. Biundo and D. Dengler. An interval-based temporal logic for planning. DFKI RR-92-12, 1992.

[4] C. Green. Application of theorem proving to problem solving. In *Proc. 1st IJCAI* , 1969.

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

[6] J. Koehler. Towards a logical treatment of plan reuse. In *Proc. 1st AIPS.* Los Altos/CA, 1992.

[7] R. Kowalski. *Logic for Problem Solving.* North-Holland Publishing Company, Amsterdam, 1979.

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

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

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

[11] L. A. Wallen. *Automated Deduction in Nonclassical Logics.* Cambridge, Mass., 1990.