ELSEVIER

Artificial Intelligence 87 (1996) 145–186

Artificial Intelligence

# Planning from second principles

Jana Koehler *

*Department of Computer Science, Albert Ludwigs University, Am Flughafen 17, D-79110 Freiburg, Germany*

Received June 1994; revised June 1995

**Abstract**

Planning from second principles by reusing and modifying plans is one way of improving the efficiency of planning systems. In this paper, we study it in the general framework of deductive planning and develop a logical formalization of planning from second principles, which relies on a systematic decomposition of the planning process.

Deductive inference processes with clearly defined semantics formalize each of the subtasks a second principles planner has to address. Plan modification, which comprises matching and adaptation tasks, is based on a deductive approach yielding provably correct modified plans. Description logics are introduced as query languages to plan libraries, which leads to a novel and efficient solution to the indexing problem in case-based reasoning.

Apart from sequential plans, this approach enables a planner to reuse and modify complex plans containing control structures like conditionals and loops.

## 1. Introduction

*Planning from first principles* generates plans from "scratch". The planner inspects its set of available actions and tries to construct a plan that achieves the desired goal with respect to specified preconditions. A serious limitation of first principles planners is the invariable nature of the planning process over time: If a planner receives exactly the same planning problem, it will repeat exactly the same planning operations. In other words, it is unable to benefit from experience that can be drawn from previous planning processes.

Approaches to *planning from second principles* try to overcome this limitation of planning from scratch by reusing and modifying previously generated plans. From a

* E-mail: koehler@informatik.uni-freiburg.de.


*SSDI* 0004-3702(95)00113-1

complexity theory point of view, we cannot hope to prove efficiency gains in the worst case [44], since the reuse of plans comprises subtasks that are not computationally easier than plan generation. But in many practical applications it seems to be more reasonable to reuse existing plans than generating a new one.

The current state of the art comprises a variety of approaches that tackle the problems from a cognitive point of view (cf. [34] for a summary of approaches) or in the framework of STRIPS-based planning, cf. [21,28,54].

In using a deductive framework, we present a formal approach to planning from second principles, which makes no commitments to particular planning formalisms and application domains. We formalize the whole reuse process in a unique logical framework including the *retrieval* of candidate plans from a library as well as the problem of plan *modification*.

Plan modification is based on deductive inference processes that yield provably correct modified plans. It comprises two subtasks: First, the task of *matching* an old and a new planning problem in order to determine their similarities and differences. Secondly, the task of *refitting* the reused plan to accommodate new requirements. As a new issue in plan modification, we discuss the reuse and refitting of control structures occurring in plans, like case analyses and loops, which introduce qualitatively new problems.

As for the plan library, we propose a *hybrid* knowledge representation formalism linking the planning logic with a *description logic*. In this approach, description logics are introduced as query languages to large knowledge bases or case libraries. Their use leads to well-defined abstraction, retrieval, and update procedures that possess theoretical and practical properties of interest. In particular, description logics enable us to develop efficient and complete approximation algorithms for the matching problem [44], which are guaranteed to retrieve all plans from the library that solve a planning problem.

Finally, the formal framework allows us to prove important properties like the correctness and completeness of the underlying inference procedures. Besides this, the approach provides the foundation for the implemented plan reuse system MRL,[1] which has been developed as an integrated part of the PHI planner [5].

The paper is organized as follows: We begin in Section 2 with a summary of the logical formalisms that are used by MRL. The section also contains a short introduction into deductive planning as well as a short overview of the PHI planner. Section 3 introduces a four-phase model as the foundation of second principles planning. The model supports a temporal view as well as a task specific view of the second principles planning process. A logical formalization of each phase provides the theoretical basis for the system MRL that is described in the subsequent sections. Sections 4 and 6 are devoted to the inference procedures working on the plan library, while in Section 5 the deductive approach to plan modification is presented. Finally, in Section 7 we review related work and propose a systematic categorization of the various principles and design decisions underlying second principles planners. We summarize the main properties of MRL in the light of this categorization.

[1] MRL stands for modification and reuse in logic.

## 2. Formal preliminaries

Deductive planning is a longstanding variant of artificial intelligence planning, whose origins go back to QA3 [18]. To generate plans deductively, constructive proofs of formal plan specifications are performed, i.e., "to construct a plan that will meet a specified condition, one proves the existence of a state in which the condition is true", cf. [37, p. 14]. Usually, this requires us to constructively prove plan specifications of the form

$$\forall s_0 \, \forall a \, \exists z \; Q[s_0, a, z]$$

where $s_0$ denotes the initial state, $a$ is an argument or input parameter, and $z$ is a planvariable representing the plan term that has to be constructed [37].

Two properties of deductive planners are particularly interesting when studying planning from second principles: First, plans are provably correct. Once provided with a correct axiomatization of a particular application domain and the actions which can be performed in this domain, a deductive planner generates plans that are guaranteed to work. Preserving this property during plan modification is a real challenge—how can we ensure that removing, reordering or adding actions leads to a sound plan, that solves the planning problem at hand?

Secondly, deductive planning has been closely related to program synthesis right from its origins. Plans are viewed as programs and consequently, they contain control structures like *if-then-else* and *while* loops. While it is very rare that classical planners generate such complex plans, many deductive planning systems are able to generate control structures in plans. The retrieval and modification of plans containing conditionals and loops is therefore another challenge we are going to address.

A deductive planning system—like any other classical planning system—is faced with a search space of enormous complexity and controlling search is a difficult problem. For a deductive planner, this means controlling the inferences in the underlying logic in such a way that plans can be constructed automatically. This involves enabling a system to correctly "guess" appropriate instantiations of planvariables and to provide it with mechanisms that decide which inference rules apply to certain logical formulae. The difficulty of this task led to a temporary abandonment of deductive techniques. Many (but not all) classical planners, which have been developed in the meanwhile, sacrifice soundness by ignoring frame or ramification problems in order to reduce search complexity.

Recently, deductive planning has seen a renaissance. On one hand, various planning logics that allow for an efficient solution to the frame problem have been carefully devised, e.g., [8,47]. On the other hand, new ways of controlling a deductive planner by using *tactics* have been developed, e.g., [8,53]. The use of tactics in deductive planning is inspired by *tactical theorem proving* [11,24,46]. Tactics support the declarative representation of control knowledge and guide the inference so that plans can be automatically constructed when proving the plan specification. In practice, this implies giving up completeness in order to purchase tractability. As we will demonstrate, tactics also play an important role in implementing plan modification.

The examples we are going to use to illustrate planning from second principles are taken from the PHI planner. PHI has been designed to perform plan generation and plan recognition tasks in command language environments, e.g., software systems. It provides a logic-based kernel that can be used to develop intelligent help systems supporting users of software. A prototype application of PHI is the UNIX *mail domain* where objects like *messages* and *mailboxes* are manipulated by actions like *read*, *delete*, and *save*.

The system uses the so-called logical language for planning (LLP) [8] as the underlying planning formalism. The logic LLP reflects the specific requirements of command language environments. For example, the basic actions which occur in plans are the elementary statements of the application system language. Furthermore, control structures like loops and conditionals are available as defined operators, which allows to describe complex user actions on the level of the logical formalism.

### 2.1. The logical language for planning LLP

LLP is a modal temporal logic with interval-based semantics, which combines features of *choppy logic* [49] with a *temporal logic for programs* [35]. Interval-based temporal logics have been proposed as appropriate formalisms to describe the behavior of programs or plans, e.g., [15, 40]. Plans can be decomposed into successively smaller periods or intervals of, e.g., subplans or actions. The intervals provide a convenient framework for introducing quantitative timing details. State transitions can be characterized by properties relating the initial and final values of variables over intervals of time.

The basis of LLP is a many sorted first-order language with equality. It distinguishes *local variables*, the value of which may vary from state to state and *global* variables which are the usual logical variables. Local variables are borrowed from *programming logics* where they correspond to program variables. LLP provides the modal operators $\bigcirc$ (*next*), $\Diamond$ (*sometimes*), $\Box$ (*always*), and the binary modal operator ; (*chop*). In the following, we shortly review the main properties of LLP as introduced in [8].

#### 2.1.1. Syntax and semantics

A state $\sigma_i$ is a pair $\sigma_i = (\sigma_i^1, \sigma_i^2)$ where $\sigma_i^1$ is a valuation assigning domain elements to local variables, while $\sigma_i^2$ indicates the command to be executed in state $\sigma_i$. Note that only the values of local variables may change from state to state. Function and predicate symbols are *rigid*, i.e., their interpretation does not vary over time. An interval $w$ is a nonempty finite or infinite sequence of states $\langle \sigma_0 \sigma_1 \ldots \rangle$. $W$ denotes the set of all intervals. The length of an interval $w$ is defined as

$$|w| = \begin{cases} \omega, & \text{if } w \text{ is infinite,} \\ n, & \text{if } w = \langle \sigma_0 \sigma_1 \ldots \sigma_n \rangle. \end{cases}$$

Observe that $|w| = 0$ iff $w = \langle \sigma_0 \rangle$ is a *singleton* containing only one state. Intuitively, the length of an interval does not represent the number of states this interval contains, but the number of possible state transitions. The *immediate accessibility* on intervals is defined as the subinterval relationship $R$ with

$$w\,R\,w' \quad \text{iff} \quad w = \langle \sigma_0\sigma_1\sigma_2 \ldots \rangle \text{ and } w' = \langle \sigma_1\sigma_2 \ldots \rangle.$$

The relation $R$ is not serial, i.e., $\forall w\ \exists w'\ w\,R\,w'$ does not hold since an interval of length zero has no successor. $R^*$ denotes the transitive and reflexive closure of $R$. The *composition* is defined as a partial function over the set of intervals $W$:

$$w \circ w' = \begin{cases} w, & \text{if } w \text{ is infinite,} \\ \langle \sigma_0 \ldots \sigma_{n-1}\sigma_n\sigma_{n+1} \ldots \rangle, & \text{if } w = \langle \sigma_0 \ldots \sigma_{n-1}\sigma_n \rangle \text{ and } w' = \langle \sigma_n\sigma_{n+1} \ldots \rangle. \end{cases}$$

Global variables are interpreted by mapping them to domain elements using a valuation function. The value of a local variable in an interval $w$ for a particular interpretation is given by its value in the initial state of the interval. The satisfiability relation $\models$ for modal-free formulae is defined as in classical first-order logic. $\mathsf{F}$ and $\mathsf{T}$ denote the propositional constants *false* and *true*, respectively. The special-purpose predicate *ex*, which takes a command term as the argument, denotes the action to be executed, i.e., $w \models_{\mathcal{I}} ex(t)$ iff $\mathcal{I}(t) = \sigma_0^2$. For the modal operators we define:

$$w \models_{\mathcal{I}} \bigcirc\phi \quad \text{iff} \quad w' \models_{\mathcal{I}} \phi \text{ for all } w' \in W \text{ with } w\,R\,w',$$

$$w \models_{\mathcal{I}} \Diamond\phi \quad \text{iff} \quad w' \models_{\mathcal{I}} \phi \text{ for some } w' \in W \text{ with } w\,R^*\,w',$$

$$w \models_{\mathcal{I}} \Box\phi \quad \text{iff} \quad w' \models_{\mathcal{I}} \phi \text{ for all } w' \in W \text{ with } w\,R^*\,w',$$

$$w \models_{\mathcal{I}} \phi;\psi \quad \text{iff} \quad \text{there are } w', w'' \in W, \text{ with } w = w' \circ w'', \; w' \text{ finite and } w' \models_{\mathcal{I}} \phi \text{ and } w'' \models_{\mathcal{I}} \psi.$$

For example, $\bigcirc\mathsf{F}$ holds in an interval $w$ iff $w$ has length 0, i.e., it is a singleton. More generally, $\bigcirc^n\mathsf{F}$ holds in $w$ iff $w$ has at most $n$ states, that is iff $w$ has at most length $n-1$. A formula $\phi \wedge \neg\bigcirc\mathsf{F} \wedge \bigcirc\bigcirc\mathsf{F};\bigcirc\Box\psi$ holds in an interval $\langle \sigma_0\sigma_1\sigma_2\sigma_3 \ldots \rangle$ if

- $\phi$ holds in the subinterval $\langle \sigma_0\sigma_1 \rangle$ and
- $\bigcirc\Box\psi$ holds in the subinterval $\langle \sigma_1\sigma_2\sigma_3 \ldots \rangle$, i.e., $\psi$ holds in all subintervals $\langle \sigma_n \ldots \rangle$ with $n \geqslant 2$.

### 2.1.2. *Plans and plan specifications in LLP*

Certain types of formulae in LLP are viewed as plans and plan specifications.

**Definition 1.** Let $t$ be a command term, $\varepsilon$ an atomic formula, $\phi$ and $\psi$ consistent plan formulae. *Plans* are all formulae of the form:[2]

- $ex(t)$,
- $\phi;\psi$,
- **if** $\varepsilon$ **then** $\phi$ **else** $\psi$,
- **while** $\varepsilon$ **do** $\phi$ **od** $;\psi$.

[2] Currently, PHI uses an extended class of plan formulae comprising nonlinear plans that contain temporal abstractions ("*sometimes* execute an action"). For the purpose of this paper, we restrict the class of plan formulae to those plans that are used in the examples.

The conditional **if** $\varepsilon$ **then** $\phi$ **else** $\psi$ stands for the formula $[\varepsilon \rightarrow \phi] \wedge [\neg\varepsilon \rightarrow \psi]$. The **while** operator is defined by the following axiom:[3]

$$\textbf{while } \varepsilon \textbf{ do } \phi \textbf{ od}; \psi \leftrightarrow \textbf{if } \varepsilon \textbf{ then } \phi; [\textbf{while } \varepsilon \textbf{ do } \phi \textbf{ od}; \psi] \textbf{ else } \psi.$$

The atomic actions available to the planner are the elementary commands of the UNIX mail system. They are axiomatized like assignment statements in programming logics. Changes of state caused by executing an action are reflected in a change of the values of local variables, which represent the mailboxes in the domain under consideration. For example, the axiomatization of the *delete*-command which deletes a message $x$ in a mailbox *mbox*[4] reads

$$\forall x\, [open\_flag(mbox) = T \wedge delete\_flag(msg(x, mbox)) = F \wedge ex(delete(x, mbox)) \rightarrow \bigcirc\, delete\_flag(msg(x, mbox)) = T].$$

The state of a mailbox is represented with the help of *flags*. As a precondition, the *delete*-command requires that the mailbox *mbox* is open, i.e., its *open_flag* yields the value *true* ($T$) and that the message $x$ has not yet been deleted, i.e., its *delete_flag* yields the value *false* ($F$).[5] As an effect, the action sets the *delete_flag* of message $x$ in mailbox *mbox* to the value *true* in the next state. In general, PHI uses more general second-order axiom schemata that are instantiated during the proof process. The axiom schemata describe effects as well as frame conditions, which leads to a representational solution of the frame problem, since only one axiom schemata is necessary for each action [8].

**Definition 2.** Plan specifications are universally quantified LLP formulae of the form

$$\mathsf{Plan} \wedge preconditions \rightarrow \Diamond\, goal,$$

i.e., if the Plan is carried out in the initial state where the *preconditions* hold then a state will be achieved satisfying *goal*. Plan is a metalogical variable standing for the plan formula that has to be generated by constructively proving the plan specification, i.e., Plan meets the specification iff $\mathsf{Plan} \wedge preconditions \rightarrow \Diamond\, goal$ is true.

Plan specification formulae have to obey various syntactic restrictions. Preconditions are described by a modal operator free first-order formula containing negation, conjunction, disjunction, and a limited form of implication and universal quantification. Goals may be described by formulae containing nested *sometimes* operators, like $\Diamond [\phi \wedge \Diamond \psi]$, conjunction, and also a limited form of implication and universal quantification.

[3] To complete the recursive definition of **while**, an extra semantic condition is necessary, which amounts to a smallest fixpoint construction as in [52] for example. Since in our application the formula $\phi$ is restricted to be a valid plan formula not containing any **while** structure, the operational view of the **while** definition is sufficient.

[4] Constants begin with capital letters, while variables are written in lower case.

[5] The reader may note the difference between the *boolean* constants $T$ and $F$, which are assigned as values to local variables and the *propositional* constants $\mathsf{T}$ and $\mathsf{F}$, which are formulae as in $\bigcirc\bigcirc\mathsf{F}$ for example.

Let us consider three specifications and example plans that will be used throughout this paper. The first specification $S_{P1}$ specifies the planning problem: "read and delete a message $m$ in the mailbox *mybox*". As preconditions, we assume that the mailbox *mybox* has already been opened and that the message $m$ has not yet been deleted.

$$\begin{aligned} S_{P1} \quad & \mathsf{Plan}_{P1} \wedge \\ & open\_flag(mybox) = T \wedge delete\_flag(msg(m, mybox)) = F \\ & \rightarrow \Diamond\, [\, read\_flag(msg(m, mybox)) = T \wedge \\ & \qquad \Diamond\, [delete\_flag(msg(m, mybox)) = T]\,]. \end{aligned}$$

$S_{P1}$ specifies temporary goals with the help of nested *sometimes* operators, i.e., goals that have to be achieved at some point and not necessarily in the end. It requires the message to be read first and then deleted. The plan **P1** solving this specification is a simple sequence containing the actions *type* and *delete*.

$$\mathbf{P1} \quad ex(type(m, mybox)) \,;\, ex(delete(m, mybox)).$$

In the second specification $S_{P2}$, we have the same goals as specified in $S_{P1}$, but formulated here as a conjunctive goal.

$$\begin{aligned} S_{P2} \quad & \mathsf{Plan}_{P2} \wedge delete\_flag(msg(x, mbox)) = F \\ & \rightarrow \Diamond\, [\, read\_flag(msg(x, mbox)) = T \wedge \\ & \qquad delete\_flag(msg(x, mbox)) = T]. \end{aligned}$$

As a precondition we only know that the message has not been deleted, but no information about the state of the mailbox is available, i.e., we do not know whether the mailbox is open or closed. Thus, the plan **P2** must contain a case analysis on the state of the mailbox *mbox*: If the mailbox is open, the message $x$ can be read and deleted. If the mailbox is closed, we have to open it first before the message $x$ can be read and deleted.

$$\begin{aligned} \mathbf{P2} \quad & \textbf{if } open\_flag(mbox) = T \textbf{ then } ex(empty\_action) \\ & \qquad\qquad\qquad\qquad \textbf{else } ex(open(mbox)) \,; \\ & ex(type(x, mbox)) \,;\, ex(delete(x, mbox)). \end{aligned}$$

The third specification $S_{P3}$ specifies an *iterative* plan reading all messages from sender *Joe* in the mailbox *mbox*. The specification of its preconditions and goals contains universally quantified formulae:

$$\begin{aligned} S_{P3} \quad & \mathsf{Plan}_{P3} \wedge open\_flag(mbox) = T \wedge \\ & \forall x\, [sender(msg(x, mbox)) = Joe \\ & \qquad \rightarrow delete\_flag(msg(x, mbox)) = F] \\ & \rightarrow \Diamond \forall x\, [sender(msg(x, mbox)) = Joe \\ & \qquad\qquad \rightarrow read\_flag(msg(x, mbox)) = T \wedge \\ & \qquad\qquad\quad delete\_flag(msg(x, mbox)) = T]. \end{aligned}$$

The plan **P3**, which solves this specification contains a *while* loop over the length of the mailbox.

**P3** $n := 1$ ;
**while** $n < length(mbox)$ **do**
**if** $sender(msg(n, mbox)) = Joe$
**then** $ex(type(n, mbox))$ ; $ex(delete(n, mbox))$
**else** $ex(empty\_action)$ ;
$n := n + 1$
**od**.

*2.1.3. The LLP sequent calculus*

All deductive inferences are performed in a sequent calculus, which has been developed for LLP.[6] Sequent calculi possess several advantages making deductive planning more efficient. They support a compositional proof guidance by tactics and allow for a natural problem decomposition [12]. The calculus contains different kinds of sequent rules: the rules for the standard S4 calculus as for example given in [55], LLP specific rules to handle the modal operators *next* and *chop*, and planning specific rules, which instantiate planvariables for example.

**Definition 3.** A sequent is an ordered pair $\langle \Gamma, \Delta \rangle$ of finite sets of formulae, written $\Gamma \Rightarrow \Delta$. It is valid iff, in all models $\mathcal{M}$, if when $\mathcal{M} \models A$, for all $A \in \Gamma$, then $\mathcal{M} \models B$, for some $B \in \Delta$.

In order to prove a formula in a sequent calculus, the theorem prover tries to find a derivation (tree) of the formula by applying sequent rules, which ends in a set of axioms (leaves) from which the formula follows.

**Definition 4.** A sequent rule consists of at least one upper sequent, the premise, and a bottom sequent, the conclusion. A sequent rule is correct iff the premise implies the conclusion.

LLP specific rules are for example the *chop_composition* rule

$$\frac{\phi_1 \Rightarrow \psi_1 \quad \phi_2 \Rightarrow \psi_2}{\phi_1 ; \phi_2 \Rightarrow \psi_1 ; \psi_2} \quad chop\_composition$$

and the *sometimes_to_next* rule

$$\frac{\Gamma \Rightarrow \bigcirc\phi \wedge \neg\bigcirc\mathsf{F}, \Delta}{\Gamma \Rightarrow \Diamond\phi, \Delta} \quad sometimes\_to\_next.$$

**Definition 5.** An axiom is a sequent of form $\Gamma, A \Rightarrow A, \Delta$, i.e., antecedent and succedent contain at least one common formula.

[6] A general introduction into sequent calculi can be found in [16, 55].

If a sequent derivation of a formula does not terminate in a formulae set comprising only axioms, then the proof of this formula has failed. But if there is a subtree of the derivation tree which contains only axioms as leaves, then we have a partial proof stating the validity of a subformula.

### 2.2. Description logics

A main problem during planning from second principles is the identification of an appropriate reusable plan. This can be achieved by *matching* a given plan specification formula against a set of plan specification formulae stored in a plan library. As we have seen, plan specifications are very complex logical descriptions. Therefore, we cannot hope to find an *efficient* matching algorithm comparing two plan specifications by a *partial* or *best match*. The main idea is therefore, to shift from the *source* formalism LLP to a *target* formalism, which can be used to represent *abstracted* plan specifications. As we will show, a *partial* match in the source formalism can be easily reduced to an *exact* match in the target formalism.

This approach uses *concept description logics* (also called terminological logics) as the ideal target formalism for the representation of abstract knowledge. Furthermore, we introduce description logics as *query languages* to plan libraries by formalizing the retrieval problem as a *classification* task. With that, description logics lead to a novel and efficient solution to the *indexing problem* in case-based reasoning.

#### 2.2.1. Syntax and semantics

Description logics comprise a whole family of logical languages with different degrees of expressivity and inference services of different computational complexity.[7] In the following, we define a language of restricted expressivity which is sufficient for the representation of abstracted plan specifications. It can be considered as a subset of various well-known description logics such as $\mathcal{ALC}$ [50], CLASSIC [9], and $\mathcal{KRIS}$ [2].

The basic "building blocks" are *concepts* and *roles*, which denote subsets of the objects of the application domain $\mathcal{D}$ and binary relationships over $\mathcal{D}$ connecting objects, respectively. The concept $\top$ (*top*) denotes the whole domain, while $\bot$ (*bottom*) denotes the empty set. Concepts are defined intensionally in terms of descriptions that specify the properties an object must satisfy to belong to the concept.

**Definition 6.** Let $C$ and $D$ be syntactical variables for concept descriptions and $R$, $R_1$, $R_2$ for role names. The following concept descriptions can be formed:

- $C \sqcap D$ (conjunction),
- $C \sqcup D$ (disjunction),
- $\neg C$ (negation),
- $\exists R.C$ (existential role restriction),
- $R_1 \circ R_2$ (role composition/role chain).

[7] For a good introduction into description logics see for example [41].

Besides the construction of complex concepts, new concept descriptions can be defined with the help of terminological axioms.

**Definition 7.** Let $A$ be a concept name and $D$ a concept description, then $A \doteq D$ is a terminological axiom.

Primitive concepts are all concepts which are not contained on left sides of terminological axioms. They remain undefined in the terminology and can be considered as the basic terms on which other concept descriptions can be built.

The formal model theoretic semantics of description logics uses the set of objects, the domain $\mathcal{D}$, for the interpretation of concept descriptions. Concepts denote a (sub)set of objects, while each role denotes a set of object pairs. These sets are called *extensions* of concepts and roles.

**Definition 8.** An interpretation $\mathcal{I}$ consists of a domain $\mathcal{D}$ and an extension function $\mathcal{E}$ mapping each concept name $A$ to a subset $\mathcal{E}[A]$ from $\mathcal{D}$ and each role name $R$ to a binary relation $\mathcal{E}[R]$ over $\mathcal{D}$.

The terminological relationships between concept descriptions can be used to determine the necessary relationships between their extensions.

**Definition 9.** Let $C$ be the set of concept symbols and $R$ be the set of role symbols. $\mathcal{D}$ is an arbitrary set and $\mathcal{E}$ a function:

$$\mathcal{E} = \begin{cases} C \to 2^{\mathcal{D}}, \\ R \to 2^{\mathcal{D}\times\mathcal{D}}. \end{cases}$$

$\mathcal{E}$ is an extension function iff the following equations hold:

$$\begin{aligned}
\mathcal{E}[\top] &= \mathcal{D}, \\
\mathcal{E}[\bot] &= \emptyset, \\
\mathcal{E}[C \sqcap D] &= \mathcal{E}[C] \cap \mathcal{E}[D], \\
\mathcal{E}[C \sqcup D] &= \mathcal{E}[C] \cup \mathcal{E}[D], \\
\mathcal{E}[\neg C] &= \mathcal{D} \setminus \mathcal{E}[C], \\
\mathcal{E}[\exists R.C] &= \{x \in \mathcal{D} \mid \text{exists } y \in \mathcal{D}\ \langle x, y\rangle \in \mathcal{E}[R] \text{ and } y \in \mathcal{E}[C]\}, \\
\mathcal{E}[R_1 \circ R_2] &= \{x, y \in \mathcal{D} \mid \text{exists } z \in \mathcal{D}\ \langle x, z\rangle \in \mathcal{E}[R_1] \text{ and } \langle z, y\rangle \in \mathcal{E}[R_2]\}.
\end{aligned}$$

The interpretation of a terminological axiom is the equation $\mathcal{E}[A] = \mathcal{E}[D]$.

**Definition 10.** A concept $C$ is satisfiable (also denoted as consistent) iff it has a nonempty extension $\mathcal{E}[C] \neq \emptyset$.

*2.2.2. Subsumption and classification*

The main inference procedure provided in terminological systems is *subsumption*, which determines whether one concept description is more general than another.

**Definition 11.** Let $\mathcal{T}$ be a terminology and $C$, $D$ be concepts. $D$ subsumes $C$ in $\mathcal{T}$ ($C \sqsubseteq_{\mathcal{T}} D$) iff $\mathcal{E}[C] \subseteq \mathcal{E}[D]$ holds in all models of $\mathcal{T}$.

The computation of all subsumption relationships between a set of concept descriptions is called *classification*. Since we intend to formalize retrieval as a classification task, we are interested in grounding it on a complete and efficient subsumption algorithm, which runs in polynomial time depending on the number of concept forming operators in a concept description. Due to the inherent intractability of subsumption [42], we can either give up completeness, or confine concept descriptions to a subset of admissible expressions, in order to obtain an algorithm with the desired properties. Giving up completeness is problematic in this application, because inability to detect existing subsumption relations may lead to incorrect behavior of the retrieval algorithm:

- Existing solutions may not be found in the library. This can lead to an undesirable computational overhead in second principles planning, because the system does not reuse the best available plan, which may result in needless modification effort.
- Uncontrolled growth of the plan library may occur. Identical abstracted specifications are added to the library, because the incomplete subsumption algorithm is unable to recognize their equivalence.

Consequently, concept descriptions are restricted to so-called *admissible concepts* in conjunctive normal form, for which a sound, complete, and polynomial-time subsumption algorithm exists.

**Definition 12.** Let $R$ be a composition of primitive roles (role chain), and $C$ be a primitive concept atom. Concept terms of form $\exists R.C$ and $\exists R.\neg C$ are called primitive components.

**Definition 13.** A consistent concept description $D$ is admissible iff $D$ is a conjunction of concept descriptions $d_i$, where each $d_i$ is restricted to be a disjunction of primitive components.

Restricting the language to admissible concept descriptions makes the usual expansion and normalization steps unnecessary. In the general case of more expressive descriptions, the relevant part of a terminology has to be transformed into a normal form before the computation of subsumption can start. Defined concepts have to be expanded by their definition using terminological axioms. This step is not necessary for admissible concepts, because they are already given in a normal form, and all concept terms are restricted to be primitive. In fact, admissible concepts define a subset of propositional logic. Primitive components can be treated as atomic units during the computation of subsumption. The following rule set taken from [17] defines a sound and complete subsumption algorithm for admissible concepts that runs in polynomial time.

$$E \sqsubseteq_T C, E \sqsubseteq_T D \rightarrow E \sqsubseteq_T C \sqcap D,$$
$$C \sqsubseteq_T E \rightarrow C \sqcap D \sqsubseteq_T E,$$
$$C \sqsubseteq_T E, D \sqsubseteq_T E \rightarrow C \sqcup D \sqsubseteq_T E,$$
$$E \sqsubseteq_T C \rightarrow E \sqsubseteq_T C \sqcup D,$$
$$C \sqsubseteq_T C.$$

## 3. A four-phase model of second principles planning

Reasoning from second principles proceeds in a basic cycle of *problem input—activation of previous solutions—adaptation*, cf. [48]. Besides these three phases, we consider a fourth and final phase in the cycle, which updates the memory of the second principles planner.

*I. Plan determination*

The current plan specification is the input to the plan-determination phase. Taking this specification, the retrieval process starts by mapping the LLP formula into a concept description, which represents the search key to the plan library.

The plan library contains a collection of *plan entries* that are extracted from previously solved planning problems. A plan entry provides comprehensive information about a planning problem and its solution, e.g., the specification of the problem describing initial and goal states, the plan which was generated as a solution for it, and information that is extracted from the plan generation process. Each plan entry possesses an index, which is represented as a concept description. The position of a plan entry is determined by the position of its index in the subsumption hierarchy.

Retrieval classifies the current search key in the subsumption hierarchy of indices, and returns a set of reuse candidates. Ranking heuristics are applied in order to determine the best candidate.

*II. Plan interpretation*

Plan interpretation attempts to prove that the current plan specification formula $S_{\text{new}}$ is a logical consequence of the reused plan specification formula $S_{\text{old}}$. If the attempt succeeds, the reused plan specification is sufficient for the current one, which means solving the old planning problem will solve the current planning problem. This means that the reused plan solves $S_{\text{new}}$ directly, and no modification of it is necessary. If the attempt fails, a plan skeleton is constructed. A plan skeleton provides an entry point into the search space of possible plans. It represents an incomplete solution to the current planning problem, because it may contain "placeholders" for subplans achieving open subgoals, which the reused plan is unable to achieve. The plan skeleton keeps any actions of the reused plan that were determined as reusable during plan interpretation, and in which variables are appropriately instantiated with object parameters taken from the current plan specification.

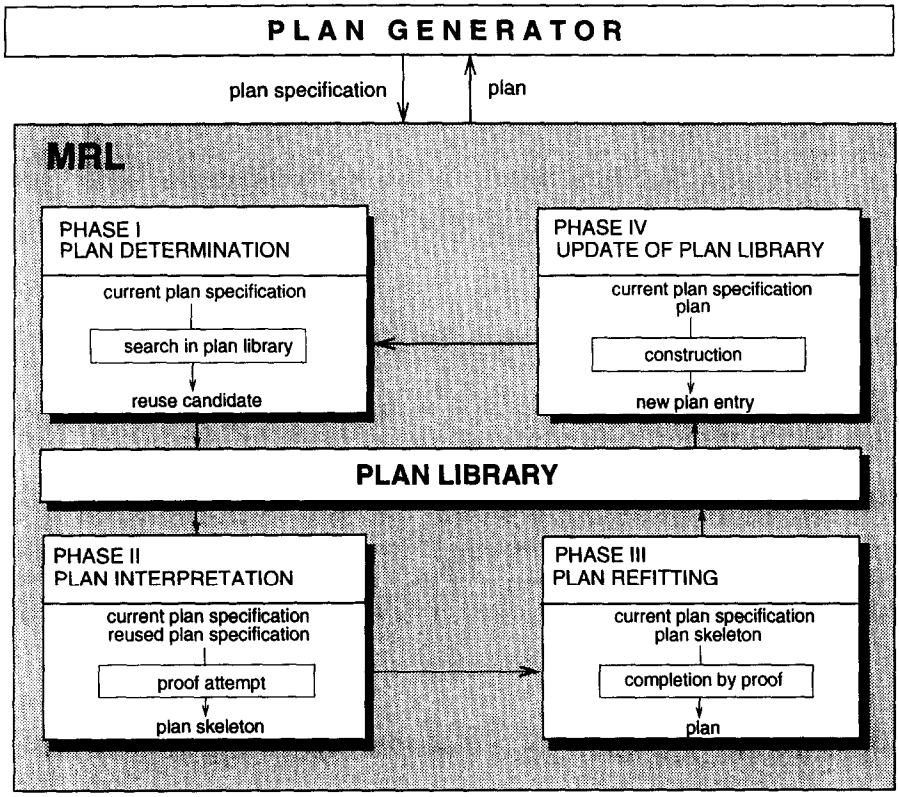


Fig. 1. Architecture of the MRL system.

*III. Plan refitting*

The third phase completes the plan skeleton to a *correct plan* with the help of an interleaved process of plan verification and generative planning. Similar to a generative planner, plan refitting selects a subgoal to work on and successively reuses subplans from the plan skeleton to construct the final plan. During this process, actions and control structures in the plan skeleton may be deleted, added or reordered.

*IV. Plan-library update*

Planning from second principles terminates with a *plan-library update*, during which a new plan entry is constructed from three sources of information: the current plan specification, the plan which was generated by modifying an existing plan, and information that is extracted from the proof tree which was constructed as a result of plan refitting. The plan entry is related to the current search key, which serves as its index in the plan library. The modified plan is now available to subsequent planning processes.

The four-phase model describes a temporal view on the reuse process. The phases I–III are necessary to generate a plan by reusing an existing one. They are also distinguished by other authors who sometimes denote them as *retrieval*, *matching*, *adaptation*

phases, cf. [22]. The fourth phase comprises the maintenance of the plan library. The formalization groups those phases together, which perform similar tasks. Operations on the plan library provide the basis for phases I and IV, while plan interpretation and refitting (phases II and III) work on plan specifications and are summarized as *plan modification*.

Fig. 1 shows the architecture of the MRL system. The system comprises four modules, each of which implements a phase of second principles planning.

### 3.1. Formalizing plan modification

The input to plan modification is the current plan specification formula $S_{\text{new}}$ of form

$$\mathsf{Plan}_{\text{new}} \wedge pre_{\text{new}} \rightarrow \Diamond\, goal_{\text{new}}$$

and the plan specification formula $S_{\text{old}}$ from the reuse candidate of form

$$\mathsf{Plan}_{\text{old}} \wedge pre_{\text{old}} \rightarrow \Diamond\, goal_{\text{old}}.$$

The planvariable $\mathsf{Plan}_{\text{new}}$ stands for the plan (formula) we want to determine by reusing the existing plan (formula) $\mathbf{P}_{\text{old}}$ that is available as a correct instantiation of the planvariable $\mathsf{Plan}_{\text{old}}$. Plan modification has now to answer the question of whether $\mathbf{P}_{\text{old}}$ meets $S_{\text{new}}$, i.e., if

$$\mathbf{P}_{\text{old}} \wedge pre_{\text{new}} \rightarrow \Diamond\, goal_{\text{new}}$$

is true. In this case, $\mathbf{P}_{\text{old}}$ can be reused immediately, otherwise it has to be refitted to meet new requirements. Instead of directly proving the validity of this formula, we show sufficient conditions between both plan specifications according to Theorem 14.

**Theorem 14.** $\mathbf{P}_{\text{old}}$ *meets* $S_{\text{new}}$ *if* $Ax \models S_{\text{old}} \rightarrow S_{\text{new}}$.

This means, if the new plan specification formula is a logical consequence of the old one under the given domain theory $Ax$, solving the old planning problem with plan $\mathbf{P}_{\text{old}}$ is sufficient for solving $S_{\text{new}}$. $Ax$ contains the set of action axiom schemata as well as the domain constraints.[8]

Since plan specifications contain formal descriptions of initial and goal states, we can show $Ax \models S_{\text{old}} \rightarrow S_{\text{new}}$ by proving sufficient relations between preconditions and goals according to Theorem 15:

**Theorem 15.** $Ax \models S_{\text{old}} \rightarrow S_{\text{new}}$ *if*

$$Ax \models pre_{\text{new}} \rightarrow pre_{\text{old}} \quad \textit{and} \quad Ax \models \Diamond\, goal_{\text{old}} \rightarrow \Diamond\, goal_{\text{new}}.$$

This means, we have to prove that the preconditions required by the old plan are satisfied in the current initial state and that the goals achieved by the old plan are

[8] An example of a domain constraint in the mail domain is the formula $\forall x\, [open\_flag(mailbox) = F \rightarrow delete\_flag(msg(x, mailbox)) = F]$, i.e., no closed mailbox can contain deleted messages.

sufficient for the currently required goals. If these relationships between initial and goal state specifications hold, we know that

- $\mathbf{P}_{\text{old}}$ is applicable in $pre_{\text{new}}$ and
- $\mathbf{P}_{\text{old}}$ achieves at least all of the goals required in $goal_{\text{new}}$.

The validity of both theorems can easily be demonstrated by a sequent proof of $Ax \models S_{\text{old}} \rightarrow S_{\text{new}}$. We start with the initial sequent

$$Ax, \mathbf{P}_{\text{old}} \wedge pre_{\text{old}} \rightarrow \Diamond goal_{\text{old}} \Rightarrow \mathbf{P}_{\text{old}} \wedge pre_{\text{new}} \rightarrow \Diamond goal_{\text{new}}$$

in which both planvariables are already instantiated with the reused plan formula. Applying sequent rules for the logical connectives $\wedge$ and $\rightarrow$ to both sides of the sequent leads to the following three proof tasks[9]

(1a) $Ax, \mathbf{P}_{\text{old}}, pre_{\text{new}} \Rightarrow \mathbf{P}_{\text{old}}, \Diamond goal_{\text{new}}$,

(1b) $Ax, \mathbf{P}_{\text{old}}, pre_{\text{new}} \Rightarrow pre_{\text{old}}, \Diamond goal_{\text{new}}$,

(1c) $Ax, \mathbf{P}_{\text{old}}, pre_{\text{new}}, \Diamond goal_{\text{old}} \Rightarrow \Diamond goal_{\text{new}}$.

Sequent (1a) is a logical axiom since antecedent and succedent contain the plan $\mathbf{P}_{\text{old}}$ as a common formula. Sequents (1b) and (1c) lead to the two subproof tasks as required in Theorem 15. Of course, sequents (1b) and (1c) would also be valid if $Ax, pre_{\text{new}} \Rightarrow goal_{\text{new}}$ can be proved, i.e., if the current plan specification is a tautology where the goal is already satisfied in the initial state. But this will seldom hold and therefore, there is not much sense in trying to prove this.

The reader may now wonder why it makes more sense to prove relations between preconditions and goals instead of directly proving the validity of $\mathbf{P}_{\text{old}} \wedge pre_{\text{new}} \rightarrow \Diamond goal_{\text{new}}$. Both approaches are possible in principle. We decided for the former since the subproofs are easier. For preconditions in particular, we in many cases have to perform a first-order proof involving formulae of a rather simple syntactic structure. This allows us to use complete proof tactics that run in polynomial time relative to the length of the formulae. Directly proving the instantiated plan specification formula means to split the plan formula into subformulae in such a way, that each of the subgoals from $goal_{\text{new}}$ can be shown to follow from a (sub)plan of $\mathbf{P}_{\text{old}}$ and the current preconditions $pre_{\text{new}}$. This requires the application of correct sequent rules for the splitting of plan formulae and goals, which often requires additional complicated proof tasks involving frame axioms.

The reader may note that an analogue to Theorem 15 can be obtained for syntactically different plan specifications, where plans are represented as terms as is usual in deductive planning. In this case, planvariables and plan terms occur as additional arguments in the goal state specifications.

### 3.2. *Formalizing library retrieval and update*

In principle, the inference procedures working on the plan library can be formalized in the same way as plan modification. A plan solving the current planning problem can be determined by finding a candidate that is applicable in the current initial state and achieves all of the current goals, i.e., by proving sufficient conditions between

[9] The sequent proof can be found in detail in Appendix A.

preconditions and goals. But apart from the complexity problems we would encounter, this is too restrictive because such a search process can only retrieve solutions. A failed proof would not tell us which of the plans is the best candidate for plan modification.

Research in case-based reasoning proposes to solve this problem by computing so-called *partial matches*.[10] Partial matches are computed between the indices of a case, which are "labels . . .that designate under what conditions each of the cases can be used to make useful inferences" [34, p. 20]. A pre-indexing technique identifies a privileged set of features to organize cases in the case library. Usually, indices are restricted to be vectors of propositional atoms in order to make the retrieval and matching problem tractable.[11]

There are several disadvantages of pre-indexing and partial matching. As Anderson et al. [1] have observed, indexing based on a fixed vocabulary hinders flexibility in retrieval and flattens the representation of cases into simple feature vectors. Besides this, these indexing schemes restrict the case library to have a *tree*-like structure usually represented as a *discrimination network* [13]. Finally, partial matches require a "relaxation" on the inference relation for which clear semantics can rarely be given and thus soundness of the retrieval algorithm is impossible to prove.

To overcome these serious limitations, we developed a novel solution to the indexing problem based on description logics. As a main advantage, description logics offer a more expressive representation language beyond sets of atoms. Simple feature vectors can be replaced by logical formulae, which may involve relational and functional descriptions. As shown in [9], such concept descriptions can be interpreted as labeled, directed multigraphs. Subsumption as the basic inference procedure in description logics formalizes the matching of these graph structures when indices have to be compared. The formal properties of the "matcher" can easily be investigated by proving formal properties of the subsumption algorithm. *Soundness* guarantees that the retrieved candidate meets the search criterion. *Completeness* ensures that all matching candidates are in the retrieval set. The *complexity* of the subsumption algorithm decides whether an efficient retrieval algorithm is available. As a further advantage, case libraries are indexed on a more general *lattice structure* provided by the subsumption hierarchy instead of a *tree* structure [29].

Finally, the pre-indexing problem can be resolved, because no set of features has to be identified in advance. Instead, the indexing vocabulary is computed from the logical representation of cases (here plan specification formulae) with the help of an *encoding scheme* $\omega$. The encoding scheme is defined as a *mapping* from LLP formulae to concept descriptions. The formal basis for $\omega$ is the model theoretic semantics of both logics. In a first step, an LLP plan specification formula is *equivalently* translated into a first-order formula using the method described in [33]. The resulting first-order formula is then replaced by an *abstracted* first-order formula.

**Definition 16.** Given two first-order formulae $\phi$ and $abs(\phi)$, the formula $abs(\phi)$ is an abstraction of $\phi$ iff

[10] See [34] for a summary of the state of the art in case-based reasoning.

[11] See [44] for a complexity theoretic analysis of the matching problem.

$$\mathcal{M} \models \phi \rightarrow abs(\phi)$$

holds in all models $\mathcal{M}$.

For example, the proposition $A$ is an abstraction of $A \wedge B$. The result of abstraction is a formula in a sublanguage of first-order logic which can be translated into a concept description by preserving equivalence again. Here, we exploit the fact that description logics can be seen as sublanguages of first-order logic [10]. A very desirable property of encoding schemes is the so-called *monotonicity property*:

**Definition 17.** Given two plan specification formulae $S_{\text{old}}$, $S_{\text{new}}$ and their respective encoding concepts $\omega(S_{\text{old}})$, $\omega(S_{\text{new}})$, an encoding scheme $\omega$ satisfies the monotonicity property iff

$$Ax \models S_{\text{old}} \rightarrow S_{\text{new}} \text{ then } \omega(S_{\text{old}}) \sqsubseteq_{\mathcal{T}} \omega(S_{\text{new}}).$$

This means, if a plan specification $S_{\text{new}}$ is a logical consequence of a plan specification $S_{\text{old}}$ in a domain axiomatization $Ax$, then $\omega(S_{\text{new}})$ must subsume $\omega(S_{\text{old}})$. In other words, the encoding scheme preserves an existing subset relationship between the set of models of the plan specification formulae as a subset relationship between the extensions of the concept descriptions. The monotonicity property ensures that existing solutions are found in the plan library, when a *complete* subsumption algorithm is used.

The *abstraction* of formulae as part of the encoding guarantees that an *exact* match as computed by the subsumption algorithm corresponds to a *partial* match between the original formulae. To compute an abstraction, we define a set of abstraction rules of form $\phi \rightarrow abs(\phi)$, which replace a formula $\phi$ by a *weaker* formula $abs(\phi)$. Examples of abstraction rules are $A \wedge B \rightarrow A$ or $\forall x\, p(x) \rightarrow \exists x\, p(x)$. An analysis of the abstraction rules allows us to draw conclusions about the degree of partial matches. Furthermore, when using different sets of abstraction rules, different degrees of partial matches can be performed by a retrieval algorithm.

Finally, an encoding scheme provides a formalization of *reasoning by approximation*. The retrieval algorithm approximates the relationship of logical consequence between plan specifications when computing subsumption between their encoding concept descriptions.

Fig. 2 summarizes the formal framework. It bases planning from second principles on deductive inference processes. Plan modification is formalized by proving sufficient conditions between preconditions and goals in the underlying planning logic. A formalization of the inference procedures working on the plan library is obtained by computing their approximation in a description logic.

The idea of exploiting relationships between preconditions and goals can be found in other approaches as well. Hanks and Weld [21] write that "retrieval takes the problem's *initial* and *goal conditions* and finds in the plan library a plan that has worked under circumstances *similar* to those posed by the current problem". The basic approach described by Hammond [19] is "to find a past plan in memory that satisfies as many of the *most important goals* as possible". Plan modification as formalized by Kambhampati

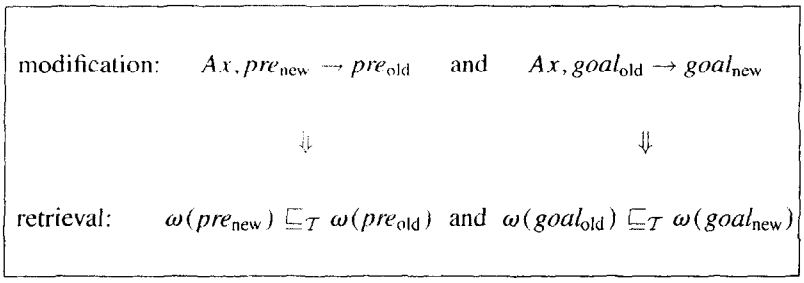


Fig. 2. The logical framework for planning from second principles.

and Hendler [28] relies on marking "the *differences* between the *initial* and *goal state specifications*".

In the remaining part of the paper we show how this formal framework serves as a theoretical foundation for the implementation of well-defined inference procedures in the second principles planner MRL. We obtain a system of predictable behavior, since theoretical properties like soundness, completeness, and efficiency of the inference procedures are provable.

## 4. Efficient retrieval of candidate plans

In order to illustrate the use of description logics as indexing and query languages let us return to the example plans, which have been introduced in Section 2. Assume that the plan **P2** solving $S_{P2}$

**P2** **if** *open_flag*(*mbox*) = *T* **then** *ex*(*empty_action*)
**else** *ex*(*mail*(*mbox*)) ;
*ex*(*type*(*x*, *mbox*)) : *ex*(*delete*(*x*, *mbox*))

has to be generated from second principles, i.e., by reusing the plans **P1** or **P3** that are stored in the plan library.

**P1** *ex*(*type*(*m*, *mybox*)) : *ex*(*delete*(*m*, *mybox*))

**P3**
*n* := 1 ;
**while** *n* < *length*(*mbox*) **do**
**if** *sender*(*msg*(*n*, *mbox*)) = *Joe*
**then** *ex*(*type*(*n*, *mbox*)) : *ex*(*delete*(*n*, *mbox*))
**else** *ex*(*empty_action*) ;
*n* := *n* + 1
**od**

Plan determination has to answer the question of whether **P1** or **P3** are appropriate candidates to guide the planning process for **P2** and which of the two candidates should

be preferred. A closer look at the plans reveals that they have a sequential subplan in common, cf. the framed formula. Apart from this, the plans differ mainly in the control structures they contain. The comparison of such complex structures is a difficult problem even for human experts. It is by no means obvious whether we should either take plan **P1** and *add a case analysis* or take plan **P3** and *remove the loop* in order to work towards **P2**.

The identification of **P1** and **P3** as appropriate reusable plans requires *abstraction* from:

- specific objects occurring in the specifications,
- temporary subgoal states,
- universally quantified goals.

The basic effects of actions which cause a mailbox's features to be changed have to be preserved during the abstraction process. These requirements have to be accomplished by defining a particular encoding scheme $\omega$, which is used in MRL to map LLP plan specifications to concepts in a description logic.

### 4.1. An example encoding scheme

The definition of a particular encoding scheme depends on three factors:

- the representation formalism for plan specifications and plans,
- the choice of a particular description logic,
- the application domain.

In order to map LLP plan specifications to concept descriptions we use three types of abstraction rules corresponding to the three different forms of abstraction that we want to perform: Abstraction from *universal goals* is accomplished by the rule

$$\forall x\, p(x) \rightarrow \exists x\, p(x).$$

Abstraction from *specific objects* is implemented by the rule

$$p(A) \rightarrow \exists x\, p(x).$$

For example, applying this rule to a formula $sender(msg(x, mbox)) = Joe$ leads to $\exists s\, sender(msg(x, mbox)) = s$ which abstracts from the specific sender of a mail. Abstraction from *temporal information* is slightly more complicated. To give the reader an impression of the temporal abstraction process we consider the formula

$$\Diamond\,[\,read\_flag(msg(m, mybox)) = T \wedge \Diamond\,[\,delete\_flag(msg(m, mybox)) = T\,]\,].$$

The goal of temporal abstraction is to abstract from the ordering of subgoal states. In the example formula, the nested *sometimes* operators specify such an ordering. A simple rule, which accomplishes the desired abstraction is

$$\Diamond\,[p \wedge \Diamond q] \rightarrow \Diamond p \wedge \Diamond q.$$

Such a modal formula can be equivalently translated into a first-order formula following the relational translation method by Ohlbach [45]. The translation adds an additional

predicate representing the accessibility relation on intervals into the object language. Local variables, which are the only fluents in LLP, are translated into unary functions that are equipped with an interval argument $w_i$ [33]. Another abstraction rule is applied to eliminate the additional predicates introduced by the translation. In the example, we would obtain ($w_1$, $w_2$ are interval variables)

$$read\_flag(msg(m, mybox(w_1))) = T \wedge$$
$$delete\_flag(msg(m, mybox(w_2))) = T.$$

The result of the abstraction process is a first-order formula containing negation, conjunction, and disjunction. Each literal is then mapped to a primitive component, i.e., an existential role restriction of the form $\exists R.C$ or $\exists R.\neg C$, while $\wedge$ and $\vee$ are mapped to $\sqcap$ and $\sqcup$, respectively.

The structure of a term like $read\_flag(msg(m, mybox(w_1)))$ is reflected in the composition of roles. The unary function *mybox* is of type *interval* $\rightarrow$ *mailbox* and is abstracted by a binary relation *interval* $\times$ *mailbox*. The binary function *msg* is of type *mailbox* $\times$ *integer* $\rightarrow$ *message*, i.e., it takes a mailbox and an integer as arguments and returns the message that can be found in the mailbox at the position indicated by the integer. Thus, this function is abstracted by the composition of binary relations *mailbox* $\times$ *integer* $\circ$ *integer* $\times$ *message*. The unary function *read_flag* is of type *message* $\rightarrow$ *boolean*, i.e., we abstract it by a binary relation of type *message* $\times$ *boolean*. Consequently, for the formula $read\_flag(msg(m, mybox(w_1))) = T$ we obtain the concept description

$$\exists \text{ mailbox} \circ \text{position} \circ \text{message} \circ \text{read\_flag.TRUE}.$$

After the encoding process has been completed, the conjunctive normal form of the concept description is computed. Of course, the computational effort for this operation grows exponentially with the formula length. But remember that the subsumption algorithm is only complete for concepts in conjunctive normal form. Nevertheless, for pragmatic reasons it is less costly to compute the normal form only once during the encoding process instead of computing it several times during the classification of an index. In many cases, the normalization will not be necessary because many planning tasks involve only *sets* of atomic subgoals.

The encoding scheme $\omega$ used in MRL leads to the following encodings of the specifications $S_{P1}$ to $S_{P3}$.

$\omega(pre_{S_{P1}})$ $\exists$ mailbox $\circ$ open_flag.TRUE $\sqcap$
$\exists$ mailbox $\circ$ position $\circ$ message $\circ$ delete_flag.FALSE

$\omega(goal_{S_{P1}})$ $\exists$ mailbox $\circ$ position $\circ$ message $\circ$ read_flag.TRUE $\sqcap$
$\exists$ mailbox $\circ$ position $\circ$ message $\circ$ delete_flag.TRUE

$\omega(pre_{S_{P2}})$ $\exists$ mailbox $\circ$ position $\circ$ message $\circ$ delete_flag.FALSE

$\omega(goal_{S_{P2}})$ $\exists$ mailbox $\circ$ position $\circ$ message $\circ$ read_flag.TRUE $\sqcap$
$\exists$ mailbox $\circ$ position $\circ$ message $\circ$ delete_flag.TRUE

$\omega(pre_{S_{P3}})$ $\exists$ mailbox ∘ open_flag.TRUE ⊓
[ $\exists$ mailbox ∘ position ∘ message ∘ sender.¬SENDER ⊔
$\exists$ mailbox ∘ position ∘ message ∘ delete_flag.FALSE ]

$\omega(goal_{S_{P3}})$ [ $\exists$ mailbox ∘ position ∘ message ∘ sender.¬SENDER ⊔
$\exists$ mailbox ∘ position ∘ message ∘ delete_flag.TRUE ] ⊓
[ $\exists$ mailbox ∘ position ∘ message ∘ sender.¬SENDER ⊔
$\exists$ mailbox ∘ position ∘ message ∘ read_flag.TRUE ]

The expressiveness of admissible concepts is sufficient to represent the mail domain adequately. The reader may note that this property may not generalize to other application domains for which different encoding schemes must be defined. In some cases, this can require to use more expressive concept languages for which subsumption becomes intractable. In this situation, we have either to give up completeness, or perform further abstractions leading to simplified formulae remaining within a tractable sublanguage.

The general idea of using description logics as query languages to case libraries seems to be widely applicable. Given a logical description of a case, i.e., a logical formula, it is possible to map it to some weaker logical formula, which can be interpreted as a concept description. Nevertheless, the development of encoding schemes mapping logical formulae to concept descriptions is a creative process. Its mechanization is an interesting subject for further research.

The encoding scheme used in MRL satisfies the monotonicity property as formulated by Definition 17. Thus, the retrieval algorithm is guaranteed to find existing solutions. Note that the inverse of the monotonicity property does not hold in general. A plan retrieved from the library will not, with certainty, provide a solution to the new planning problem. This reflects *reasoning by approximation*. The retrieval algorithm approximates the relationship between the plan specifications when comparing their abstractions. Thereby, it extends the computed set of candidates.

### 4.2. Weakening retrieval

The retrieval algorithm takes the encoding of preconditions and goals of the current plan specification in order to determine its position in the subsumption hierarchy. By testing

$$\omega(pre_{\text{new}}) \sqsubseteq_{\mathcal{T}} \omega(pre_{\text{old}}) \quad \text{and} \quad \omega(goal_{\text{old}}) \sqsubseteq_{\mathcal{T}} \omega(goal_{\text{new}})$$

a set of subsumed plan specifications is determined. The plans which meet these specifications are possible reuse candidates. If no plan is returned, i.e., if the encoded current specification only subsumes the bottom concept, no directly reusable plan is contained in the plan library. In this situation, a second principles planner is faced with the decision of either:

- to give up further reuse attempts and plan from scratch, or
- to weaken the retrieval criterion and accept that any reuse candidate has to be modified.

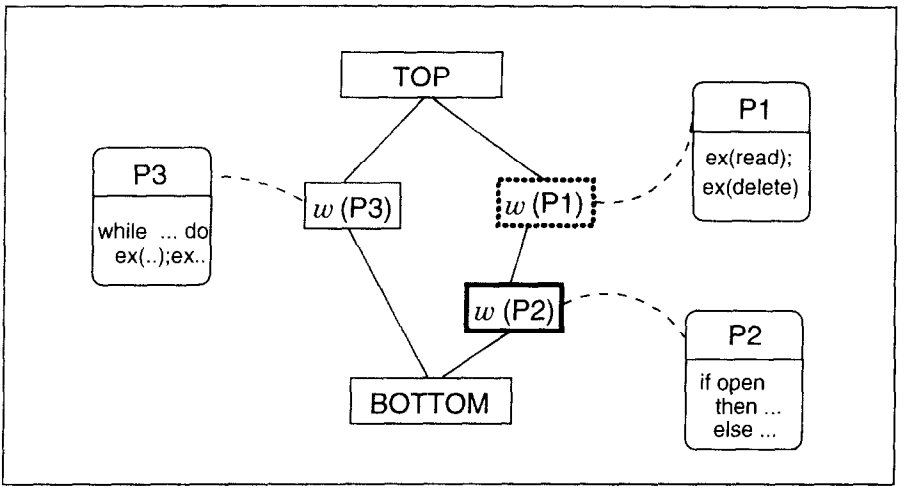


Fig. 3. The example library.

The principal problem is to anticipate the modification effort. Planning from second principles makes sense when the effort for retrieval and modification is lower than the effort for planning from scratch. Unfortunately, it is impossible to decide this *before* we start retrieval. However, practical experiments demonstrated that modifying a plan is often less costly than generating one from scratch [22,30]. Weakening retrieval and searching for plans that have to be modified is therefore a practical decision even under efficiency considerations.

Retrieval based on classification offers two principle ways to weaken the search criterion, which are to classify according to goals *or* preconditions *only*:

$$\omega(pre_{\text{new}}) \sqsubseteq \omega(pre_{\text{old}}) \quad \text{or} \quad \omega(goal_{\text{old}}) \sqsubseteq \omega(goal_{\text{new}}).$$

In the first case, retrieval searches for a plan achieving the current goals by accepting that its preconditions may not be satisfied in the current initial state. In the latter case, retrieval searches for a plan which is applicable in the current initial state, but will not achieve all of the currently required goals. [12]

Fig. 3 shows the small example library obtained for the three plan specifications under consideration. Obviously, $\omega(S_{\text{P2}})$ subsumes only the bottom concept, i.e., no directly reusable plan can be retrieved. Therefore, a weaker retrieval algorithm is activated searching for an $\omega(pre_{\text{old}_i})$ that subsumes $\omega(pre_{S_{\text{P2}}})$ or for an $\omega(goal_{\text{old}_i})$ that is subsumed by $\omega(goal_{S_{\text{P2}}})$. Classification of the encoded preconditions fails in retrieving a candidate as well, while classification of the encoded goals is successful for $\omega(goal_{S_{\text{P1}}})$ since $\omega(goal_{S_{\text{P1}}}) \sqsubseteq_{\mathcal{T}} \omega(goal_{S_{\text{P2}}})$ holds. Therefore, the plan **P1** attached to $\omega(S_{\text{P1}})$ is

[12] In a working system it seems to be a good restriction to implement only one of the possible approaches to weak retrieval in order to improve retrieval efficiency. Here, we discuss both possibilities in order to demonstrate how retrieval based on classification works. MRL applies weak retrieval only to preconditions, i.e., it requires plans to be applicable in the current initial state as a heuristic to reduce the refitting effort for control structures during plan modification.

activated as a reuse candidate. Since strong retrieval failed, we know that **P1** cannot represent a solution to the current planning problem $S_{P2}$. We expect it to achieve all of the current goals, but we know that its preconditions are not satisfied in the current initial state. Thus, plan refitting has to start as will be described in Section 5.

### 4.3. Ranking of plans

Only one candidate plan has been retrieved from the plan library in the example under consideration. But in general, retrieval will determine several appropriate reuse candidates. Consequently, a *ranking* is needed for the candidates in order to determine the best one.

The subsumption hierarchy, which is a directed acyclic graph, provides an easy way to rank candidates by computing their distance to the current concept description. The best candidate has the shortest path to $\omega(S_{\text{new}})$ in the graph. If several candidates have a shortest path, ranking heuristics are used to approximate the optimization and modification effort, respectively.

*Strong retrieval* returns plans that are supposed to be applicable in the initial state and to achieve *at least* all of the current goals. This implies that the candidate set may contain plans which achieve superfluous goals, i.e., goals that are currently unnecessary. Actions achieving these goals can be eliminated from the reused plan by making attempts at optimizing it. Thus, the ranking of candidates is based on an estimation of the *optimization effort* for each candidate, i.e., the number of superfluous actions that have to be eliminated from the candidate plan. The heuristic estimates the number of atomic subgoals that are achieved by a candidate plan but that are not required in the current plan specification. It assumes that this number reflects the minimal number of primitive actions that have to be eliminated from the candidate plan. Therefore, the plan with the smallest number is selected as the best reuse candidate and sent to plan modification. If several candidates receive the same ranking value, one of them is selected arbitrarily.

**Definition 18.** Let $\omega(S_{\text{old}_1}), \ldots, \omega(S_{\text{old}_n})$ be the indices of candidates retrieved by strong retrieval for $\omega(S_{\text{new}})$. The heuristic compares the encoding of goals of the candidates $\omega(goals_{S_{\text{old}_1}}), \ldots, \omega(goals_{S_{\text{old}_n}})$ with the encoding of the current goal $\omega(goals_{S_{\text{new}}})$. The set of primitive components that occur in a concept $C$ is denoted by $\mathsf{P}[C]$. The cardinality of the set $\mathsf{P}[C]$ is as usually denoted by $|\mathsf{P}[C]|$.

The optimization effort for each candidate is defined as

$$OPT_{\omega(goals_{S_{\text{old}_i}})} = |\mathsf{P}[\omega(goals_{S_{\text{old}_i}})] \setminus \mathsf{P}[\omega(goals_{S_{\text{new}}})]|.$$

The ranking heuristic selects a plan needing minimal optimization effort.

*Weak retrieval* returns candidate plans that are either supposed to be applicable in the initial state or to achieve the desired goals, i.e., we have to expect that every candidate has to be modified. Consequently, the heuristic estimates the effort to *modify* each candidate in the retrieval set by computing the intersection of $\omega(goals_{S_{\text{new}}})$ with $\omega(goals_{S_{\text{old}_1}}), \ldots, \omega(goals_{S_{\text{old}_n}})$, which approximates the number of *current* atomic subgoals that are achieved by each candidate, cf. [29].

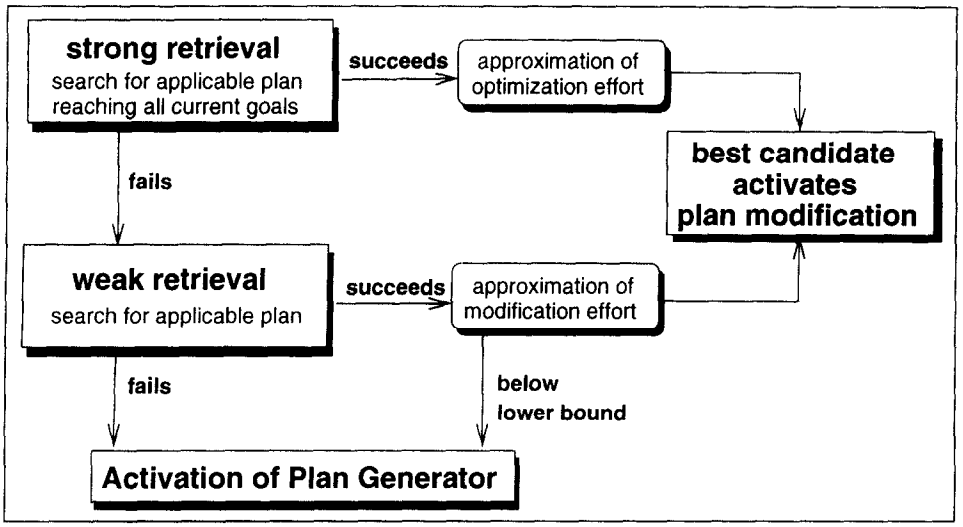


Fig. 4. Interaction between first and second principles planning.

Furthermore, the ranking heuristic verifies whether the ranking value of the best candidate exceeds a lower bound: it requires that at least half of the primitive components in $\omega(goal_{S_{new}})$ must be contained in $\omega(goal_{S_{old_i}})$. If this condition is satisfied, the ranking heuristic assumes that the best candidate achieves at least half of the current atomic subgoals. If no candidate receives a ranking value which exceeds the lower bound, all candidates are rejected because their modification effort is estimated as too expensive. In this situation, plan determination reports a failure and planning from scratch with the PHI planner is activated.

The ranking heuristics guide the interaction between planning from first and second principles, see Fig. 4.

## 5. Correct modification of complex plans

Plan modification is based on deductive inference processes which lead to modified plans that are provably correct. As introduced in Section 3, it proceeds in two phases. First, *plan interpretation* computes a plan skeleton and secondly, *plan refitting* completes the plan skeleton to a correct plan that meets the current specification.

In the following, we apply the formal approach to plan modification as defined in Section 3 to the example under consideration and discuss deductive plan modification in MRL.

### 5.1. Plan interpretation

Plan interpretation receives two sources of input:

(i) The current plan specification for which a plan has to be generated.

(ii) The best reusable plan, which the determination phase could identify in the plan library, together with its specification.

It takes the two plan specifications and tries to prove the required relations between preconditions and goals:

$$Ax \models pre_{\text{new}} \rightarrow pre_{\text{old}} \quad \text{and} \quad Ax \models \Diamond goal_{\text{old}} \rightarrow \Diamond goal_{\text{new}}.$$

In this example, proving the applicability of the reused plan **P1** in the current initial state as specified in $S_{P2}$ requires the proof of sequent (1):

$$\begin{aligned} &delete\_flag(msg(x, mbox)) = F \\ &\Rightarrow open\_flag(mybox) = T \wedge delete\_flag(msg(m, mybox)) = F. \end{aligned} \tag{1}$$

Starting point for the *goal proof* is sequent (2):

$$\begin{aligned} &\Diamond [read\_flag(msg(m, mybox)) = T \wedge \\ &\quad \Diamond [delete\_flag(msg(m, mybox)) = T]] \\ &\Rightarrow \Diamond [read\_flag(msg(x, mbox)) = T \wedge \\ &\quad delete\_flag(msg(x, mbox)) = T]. \end{aligned} \tag{2}$$

In the following, we discuss both sequent proofs as they are performed by the proof tactics used during plan interpretation [31]. The tactics run in polynomial time on the length of the input formula. On one hand, this enables plan interpretation to *efficiently* compute an *entry point* into the search space of plans. On the other hand, this implies that the tactic is *incomplete* in the sense that it cannot compute a *maximal plan skeleton* which has been shown to be a PSPACE-hard problem [44].

The precondition proof for the example sequent is very simple because of the simple syntactic structure of the formulae. The first rule that is successfully applied to sequent (1) is rule $r\wedge$.[13]

$$\bullet \ \frac{\Gamma, A, B \Rightarrow \Delta}{\Gamma, A \wedge B \Rightarrow \Delta} \, l\wedge \qquad \bullet \ \frac{\Gamma \Rightarrow A, \Delta \quad \Gamma \Rightarrow B, \Delta}{\Gamma \Rightarrow A \wedge B, \Delta} \, r\wedge.$$

The rule splits sequent (1) into sequents (3) and (4):

$$delete\_flag(msg(x, mbox)) = F \Rightarrow open\_flag(mybox) = T, \tag{3}$$

$$delete\_flag(msg(x, mbox)) = F \Rightarrow delete\_flag(msg(m, mybox)) = F. \tag{4}$$

While sequent (3) cannot be reduced to an axiom, sequent (4) can be closed, i.e., it leads to an axiom under the substitution $\{x/m, mbox/mybox\}$. In order to obtain an appropriate instantiation of the reused plan, variables in the reused specification $S_{\text{old}}$ are substituted by terms which occur in the current specification $S_{\text{new}}$. Furthermore, different variables must be mapped to different terms, i.e., the substitutions must be *injective*. Injectivity may not always be required, but it is a safe condition ensuring that a proper instantiation of the reuse candidate is computed during the proof. The reader

[13] A survey of all sequent rules that are used in this paper can be found in Appendix B.

may note that an instantiation of variables in sequents during a sequent proof is only possible when quantifier rules are applied. Plan specification formulae are implicitly universally quantified, i.e., when proving $S_{\text{old}} \Rightarrow S_{\text{new}}$ in the sequent calculus we remove the universal quantifiers using the rules[14]

$$\bullet\ \frac{\Gamma, A[c/x] \Rightarrow \Delta}{\Gamma, \forall x\, A \Rightarrow \Delta}\, l\forall \qquad \bullet\ \frac{\Gamma \Rightarrow \Delta, A[a/x]}{\Gamma \Rightarrow \Delta, \forall x\, A}\, r\forall$$

and have to "guess" the appropriate instantiation. Of course, this is unacceptable in an implemented prover due to the resulting computational overhead. Therefore, the instantiation is delayed until we know which instantiation is appropriate, i.e., which one will lead to a proof of the sequent. The restrictions we pose on the instantiations of the leaf sequents ensure that only those instantiations are computed that can be introduced with the help of quantifier rules.

Sequents (3) and (4) are the leaves of the derivation tree, because no further rules are applicable. Since only one of the leaves is an axiom, the tactic did not find a valid proof of the reused plan's preconditions.

When trying to prove that the reused plan achieves all of the current goals the prover has to cope with *sometimes* operators and therefore additionally uses the following sequent rules:

$$\bullet\ \frac{\Gamma^*, A \Rightarrow \Delta^*}{\Gamma, \Diamond A \Rightarrow \Delta}\, l\Diamond \qquad \bullet\ \frac{\Gamma \Rightarrow A, \Delta}{\Gamma \Rightarrow \Diamond A, \Delta}\, r\Diamond$$

with $\Gamma^*$ and $\Delta^*$:

$$\Gamma^* \stackrel{\text{def}}{=} \{\Box B \mid \Box B \in \Gamma\} \quad \text{and} \quad \Delta^* \stackrel{\text{def}}{=} \{\Diamond B \mid \Diamond B \in \Delta\}.$$

The proof proceeds recursively over the *sometimes* operators in both goal specifications in order to compare every temporary subgoal state specified in $goal_{\text{old}}$ with each of the temporary subgoal states from $goal_{\text{new}}$. First, the tactic applies rule $l\Diamond$ to sequent (2) followed by rule $r\Diamond$ and obtains sequent (5):

$$\begin{aligned} &read\_flag(msg(m, mybox)) = T \wedge \\ &\quad \Diamond [delete\_flag(msg(m, mybox)) = T] \\ &\Rightarrow read\_flag(msg(x, mbox)) = T \wedge delete\_flag(msg(x, mbox)) = T. \end{aligned} \tag{5}$$

Now, rule $l\wedge$ is applied to sequent (5) followed by rule $r\wedge$, which leads to sequents (6) and (7):

$$\begin{aligned} &read\_flag(msg(m, mybox)) = T, \\ &\quad \Diamond [delete\_flag(msg(m, mybox)) = T] \\ &\Rightarrow read\_flag(msg(x, mbox)) = T, \end{aligned} \tag{6}$$

[14] Eigenvariable condition: $a$ must not occur in the conclusion of $r\forall$.

$$read\_flag(msg(m, mybox)) = T,$$
$$\Diamond\,[delete\_flag(msg(m, mybox)) = T]$$
$$\Rightarrow delete\_flag(msg(x, mbox)) = T. \quad (7)$$

Sequent (6) can also be closed under the substitution $\{x/m, mbox/mybox\}$, i.e., the current subgoal $read\_flag(msg(x, mbox)) = T$ has been successfully proved. The system proceeds on sequent (7) and removes the remaining $\Diamond$ operator with the help of rule $l\Diamond$ which leads to sequent (8):

$$delete\_flag(msg(m, mybox)) = T \Rightarrow \emptyset. \quad (8)$$

The formula $delete\_flag(msg(x, mbox)) = T$ from the succedent of sequent (7) disappears in sequent (8) because it does not occur in the scope of a $\Diamond$ operator. Thus, the tactic fails in proving the remaining subgoal. The reason for this failure is obvious: the current goal specification requires the two subgoals to be achieved in the same state, while the reused goal specification only requires the two subgoals to be achieved one after the other. Of course, deleting a message preserves the effect that the message has been read, i.e., the reused plan that first reads the message and then deletes it also leads to a final state where the message has been read and deleted. But we have no way to derive this fact from the original plan specification formula. This is a motivation for a completion process of plan specification formulae that is described in Section 6.

### 5.2. *Plan refitting*

Proof tactics are always designed to terminate. In addition, they are considered as decision procedures: If a tactic does not result in a proof tree, it is assumed that no proof is possible and that a falsifying valuation for some of the leaves has been obtained. Two situations are possible after the termination of a proof tactic in the sequent calculus:

(i) A proof tree has been constructed, i.e., the leaves of the tree describe a set of logical axioms from which the original formula follows. In this case the original formula was proved to be valid.

(ii) No proof tree has been found and the assumption is made that no proof is possible and that a counter-example tree has been constructed.

This assumption is a safe condition ensuring the soundness of plan modification. Remember that the tactics are incomplete, i.e., when a tactic terminates with a failure it might either be the case that the formula is invalid or that the formula is valid, but the tactic failed to find a proof.

Assuming that the formula is invalid ensures that the correctness of a plan is verified during plan refitting. Thereby, it prevents the reuse of plans that are not provably correct with respect to the current plan specification.

The proof tactics guarantee that the leaves of a counter-example tree contain only atomic formulae. The falsifying valuation makes:

- All *old* atomic goals (in the example from $S_{P1}$) true, however some of the atomic formulae which describe *current goals* (in the example from $S_{P2}$) are valued as false. These falsified goals are interpreted as those current goals that are not achieved by the reused plan (in the example by **P1**).

- All atomic formulae describing *current* preconditions (in the example from $S_{P2}$) true, but some of the *old* preconditions (in the example from $S_{P1}$) false. These falsified preconditions are interpreted as those preconditions of the reused plan (in the example of **P1**) that do not hold in the current initial state.

Plan **P1** must be modified by constructing a plan skeleton from it, because it was neither possible to prove that its preconditions are satisfied nor that it achieves all of the currently required goals. First, the reused plan is instantiated with the substitutions $\{x/m, mbox/mybox\}$ computed during plan interpretation leading to

$$\mathbf{P1}' \quad ex(type(x, mbox)) ; ex(delete(x, mbox)).$$

Plan refitting concludes from the failed proofs that the (instantiated) precondition $open\_flag(mbox) = T$ required by **P1** does not hold in the initial state and that the current goal $delete\_flag(msg(x, mbox)) = T$ is not achieved by it. Furthermore, **P1** achieved a subgoal

$$delete\_flag(msg(m, mybox)) = T$$

that is not contained in the axiom that was obtained from sequent (6), which was constructed during the goal proof. Thus, plan refitting concludes that the action $ex(delete(m, mybox))$ achieving this subgoal is (at least at the current position where it occurs) superfluous and can be removed from the plan skeleton.

This analysis of the result of plan interpretation leads to the following modification operations that have to be performed on the instantiated plan $\mathbf{P1}'$:

(i) A planvariable $\mathsf{Plan}_1$ has to be introduced in front of the reused plan. It represents a subplan achieving the missing *precondition*

$$open\_flag(mbox) = T.$$

(ii) The *superfluous* action $ex(delete(x, mbox))$ is removed from the plan skeleton.

(iii) A planvariable representing the subplan for the open *subgoal*

$$delete\_flag(msg(x, mbox)) = T$$

must be introduced into the plan skeleton. In order to determine the position in the skeleton where this planvariable has to be added, the current goal state specification must be analyzed with the help of the PHI planner.

The planvariable in the current plan specification $\mathbf{S}_{P2}$ is instantiated with the preliminary plan skeleton $\mathbf{P1}''$. It serves as a starting point for plan refitting:

$$\mathbf{P1}'' \quad \mathsf{Plan}_1 ; ex(type(x, mbox)).$$

$$S_{P2'} \quad \mathsf{Plan}_1 ; ex(type(x, mbox)) \wedge delete\_flag(msg(x, mbox)) = F \rightarrow \Diamond [read\_flag(msg(x, mbox)) = T \wedge delete\_flag(msg(x, mbox)) = T].$$

In a first step, a subplan to replace $\mathsf{Plan}_1$ has to be generated. Plan refitting applies the rule *effect_intro* [7] to $S_{P2'}$ and introduces the missing precondition

$$open\_flag(msg(mbox)) = T$$

as the new subgoal $goal_{new}$:

$$\frac{pre, \mathsf{Plan}_1 \Rightarrow \Diamond\,[goal_{new} \wedge \mathsf{OF} \wedge pre'] \quad pre', \mathsf{Plan}_2 \Rightarrow \Diamond\, goal}{pre, \mathsf{Plan}_1 ; \mathsf{Plan}_2 \Rightarrow \Diamond\, goal} \; \textit{effect\_intro.}$$

It obtains two subplan specifications (9) and (10) where $\mathsf{Plan}_2$ is instantiated with the action $ex(type(x, mbox))$ taken from the plan skeleton. The precondition $pre'$ can be instantiated after a plan for $\mathsf{Plan}_1$ has been generated and frame conditions have been computed.

$$\begin{aligned} &delete\_flag(msg(x, mbox)) = F, \mathsf{Plan}_1 \\ &\Rightarrow \Diamond\,[open\_flag(msg(mbox)) = T \wedge \mathsf{OF} \wedge pre'] \end{aligned} \tag{9}$$

$$\begin{aligned} &pre', ex(type(x, mbox)) \\ &\Rightarrow \Diamond\,[read\_flag(msg(x, mbox)) = T \wedge \\ &\qquad delete\_flag(msg(x, mbox)) = T]. \end{aligned} \tag{10}$$

The proof of subplan specification (9) leads to a conditional plan because there is no atomic action available in the domain axiomatization that achieves the required goal under the given precondition. Plan refitting applies the rule *if_intro* [7] to instantiate the planvariable $\mathsf{Plan}_1$ with a case analysis:

$$\frac{pre, \textbf{if } (cond, \mathsf{Plan}_A, \mathsf{Plan}_B) \Rightarrow \Diamond\, goal}{pre, \mathsf{Plan} \Rightarrow \Diamond\, goal} \; \textit{if\_intro.}$$

In the example, the conditional *cond* is instantiated with the missing precondition $open\_flag(mbox) = T$ that plan interpretation failed to prove:

$$\begin{aligned} \mathsf{Plan}_1 := \textbf{if } &open\_flag(mbox) = T \textbf{ then } \mathsf{Plan}_3 \\ &\textbf{else } \mathsf{Plan}_4. \end{aligned}$$

Applying the rule *if_splitting* [7]

$$\frac{pre, cond, \mathsf{Plan}_A \Rightarrow \Diamond\, goal \quad pre, \neg cond, \mathsf{Plan}_B \Rightarrow \Diamond\, goal}{pre, \textbf{if } (cond, \mathsf{Plan}_A, \mathsf{Plan}_B) \Rightarrow \Diamond\, goal} \; \textit{if\_splitting}$$

to sequent (9), plan refitting obtains the following subplan specifications:

$$\begin{aligned} &delete\_flag(msg(x, mbox)) = F, open\_flag(mbox) = T, \mathsf{Plan}_3 \\ &\Rightarrow \Diamond\,[open\_flag(mbox) = T \wedge \mathsf{OF} \wedge pre'], \end{aligned} \tag{11}$$

$$\begin{aligned} &delete\_flag(msg(x, mbox)) = F, \neg open\_flag(mbox) = T, \mathsf{Plan}_4 \\ &\Rightarrow \Diamond\,[open\_flag(mbox) = T \wedge \mathsf{OF} \wedge pre']. \end{aligned} \tag{12}$$

$\mathsf{Plan}_3$ can be instantiated with the empty action $ex(empty\_action)$, because the desired subgoal holds already in the initial state.

The formula $\neg open\_flag(mbox) = T$ in sequent (12) is equivalent to the formula $open\_flag(mailbox) = F$ and thus, $\mathsf{Plan}_4$ is instantiated with the action instance $ex(open(mbox))$ which opens the mailbox and starts a mail session:

$$\forall mailbox\ [open\_flag(mailbox) = F \wedge ex(open(mailbox)) \rightarrow \bigcirc\, open\_flag(mailbox) = T].$$

With that, the following conditional plan is obtained as an instantiation of $\mathsf{Plan}_1$:

$$\mathsf{Plan}_1 := \textbf{if } open\_flag(mbox) = T \textbf{ then } ex(empty\_action) \textbf{ else } ex(open(mbox)).$$

Now, the precondition $pre'$ in sequent (10) can be instantiated with the formula

$$delete\_flag(msg(x, mbox)) = F \wedge open\_flag(mbox) = T.$$

The proof of subplan specification (10) proceeds as an interleaved process of plan generation and plan verification. A tactic for the ordering of conjunctive goals is activated [7] which decides to achieve the subgoal $read\_flag(msg(x, mbox)) = T$ before the subgoal $delete\_flag(msg(x, mbox)) = T$, since deleting a message destroys the possibility of reading it subsequently. The first subgoal is isolated with the help of the *effect_split* rule [7]:

$$\frac{pre, \mathsf{Plan}_A \Rightarrow \Diamond [goal_1 \wedge \bigcirc \mathsf{F} \wedge pre' \mid \quad pre', \mathsf{Plan}_B \Rightarrow \Diamond [goal_2]}{pre, \mathsf{Plan}_A ; \mathsf{Plan}_B \Rightarrow \Diamond [goal_1 \wedge goal_2]} \quad effect\_split.$$

This rule requires a sequential composition of two planvariables that can be split such that the first planvariable represents a subplan achieving the first subgoal, while the second planvariable represents a *subplan* achieving the remaining subgoals. But the planvariable $\mathsf{Plan}_2$ introduced by the *effect_intro* rule has been instantiated with the *single* atomic action $ex(type(x, mbox))$ in specification (10). Thus, this instantiation must be withdrawn and plan refitting sets $\mathsf{Plan}_2$ to $\mathsf{Plan}_5 ; \mathsf{Plan}_6$.

We obtain two subplan specifications (13) and (14):

$$open\_flag(msg(mbox)) = T \wedge delete\_flag(msg(x, mbox)) = F,\ \mathsf{Plan}_5 \Rightarrow \Diamond [read\_flag(msg(x, mbox)) = T \wedge \bigcirc \mathsf{F} \wedge pre''], \tag{13}$$

$$pre'', \mathsf{Plan}_6 \Rightarrow \Diamond [delete\_flag(msg(x, mbox)) = T]. \tag{14}$$

The first subgoal $read\_flag(msg(x, mbox)) = T$ (sequent (13)) has successfully been proven during plan interpretation. Consequently, the action from the plan skeleton $ex(type(x, mbox))$ achieving this subgoal is reused as an instantiation of the planvariable $\mathsf{Plan}_5$:

$$open\_flag(msg(mbox)) = T \wedge delete\_flag(msg(x, mbox)) = F,$$
$$ex(type(x, mbox))$$
$$\Rightarrow \Diamond [read\_flag(msg(x, mbox)) = T \wedge \mathsf{OF} \wedge pre''] . \quad (15)$$

The instantiation can be successfully verified by plan refitting. The precondition $pre''$ in sequent (14) is instantiated with

$$open\_flag(msg(mbox)) = T \wedge delete\_flag(msg(x, mbox)) = F$$

because this precondition "survives" the execution of the *type* action.

$$open\_flag(msg(mbox)) = T \wedge delete\_flag(msg(x, mbox)) = F,$$
$$\mathsf{Plan}_6$$
$$\Rightarrow \Diamond [delete\_flag(msg(x, mbox)) = T] . \quad (16)$$

Sequent (16) isolates the open subgoal $delete\_flag(msg(x, mbox)) = T$ that plan interpretation failed to prove. Plan refitting concludes that the reused plan provides no instantiation and relies on planning from scratch. It generates the action $ex(delete(x, mbox))$ that instantiates the remaining planvariable $\mathsf{Plan}_6$. With this, all planvariables have been successfully instantiated and a correct proof of the plan specification has been constructed by plan refitting. The result is the desired plan **P2** that is obtained by reusing the sequential plan **P1**:

**P2** **if** $open\_flag(mbox) = T$ **then** $ex(empty\_action)$
**else** $ex(open(mbox))$ ;
$ex(type(x, mbox))$ ; $ex(delete(x, mbox))$.

The planning process benefits from the reuse of plan **P1** in two situations:

- When a conditional control structure has to be introduced; here planning from second principles "knows" on which formula the case analysis has to be performed.
- When the subgoal $read\_flag(msg(x, mbox)) = T$ has to be addressed; here planning from second principles reuses an action instantiation that achieved the same goal in the candidate plan.

The search space during planning can be dynamically restricted in both cases, which leads to a speed up of the second principles planner when compared to the generative planner.[15] A maximal reuse of the candidate plan is not possible according to the complexity results in [43]. In the example, this leads to some overhead during plan refitting where the action instance $ex(delete(m, mybox))$ is eliminated from the original plan, but subsequently re-introduced as the action instance $ex(delete(x, mbox))$. This demonstrates "that it is not possible to determine efficiently (i.e., in polynomial time) a maximal reusable plan skeleton *before* plan generation starts to extend this skeleton", see [43, p. 1440].

The example demonstrated the generation of a conditional plan by reusing a sequential plan. MRL is the first system that is able to correctly reuse and modify plans containing

[15] A summary of the results of an empirical study can be found in [30,32].

control structures. Usually, these plans are much more complex than those shown in the simple example. Refitting of such plans with a large number of atomic actions and nested control structures can involve several hundred deduction steps.

### 5.3. *Reuse of control structures*

The reuse and modification of plans with control structures leads to qualitatively new problems that do not occur in approaches restricting themselves to sequential plans. The modification of sequential plans comprises operations like the instantiation, deletion, addition or reordering of atomic actions. The modification of complex plans raises the question of whether these operations can be extended to control structures. Two main decisions have to be made:

(i) Are control structures reused? *versus*
Are only those sequential subplans reused that occur in the scope of control structures?

(ii) Are control structures introduced by the modification strategy if this is required by the refitting process? *versus*
Are control structures only introduced if the current planning problem requires a plan containing control structures?

The treatment of control structures in a second principles planner requires to make these decisions carefully and to take into consideration specific requirements from the application domain. The MRL system provides the reuse component of the PHI planner which is working in a help system application. Here, plans are generated to provide active help to users of software environments [5]. This means that plans are required to meet exactly the user's goals and to be as simple as possible. Therefore, control structures are only reused in a restricted way in the implemented system MRL. They are introduced into the modified plan or preserved in the plan skeleton only if the current planning problem requires the generation of a plan containing control structures.

An unrestricted reuse of control structures can lead to the following problems:

- Reused control structures are not guaranteed to correspond to the requirements of the current planning situation. This can result in *over-complicated plans*. For example, a case analysis makes the execution of a plan more complicated because a test on the conditional has to be performed during execution time. Thus, a case analysis should only be introduced into a plan skeleton when the current planning problem requires us to generate a conditional plan.
- Plans can achieve *unintended side-effects*. Plan refitting makes some attempts at optimizing a reused plan by removing superfluous actions from it, but it is not able to generate *optimal* plans because this is usually harder than generating an arbitrary plan. Superfluous control structures render the problem worse. For example, an iterative plan which achieves a particular goal for all objects satisfying a precondition could in principle be reused to satisfy the goal for only one of the objects. As an example, the reader may think of reusing a plan achieving the goal: "delete all my files in directory $x$". This also achieves the goal: "delete file $x.ps$ in directory $x$". Without any attempts at optimizing the reused plan by removing the superfluous iterative control structure a drastic and harmful side-effect is achieved.

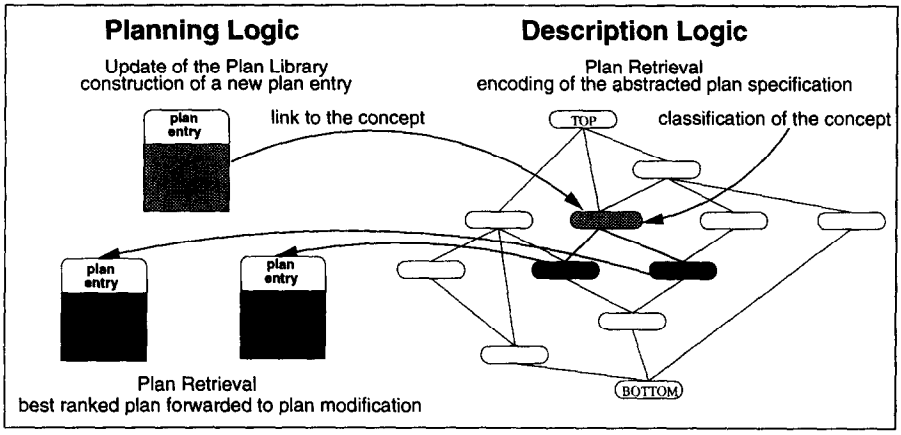


Fig. 5. Plan entries in the plan library.

Restricting the reuse of control structures as in MRL is one way of coping with these problems. Further research is necessary in order to identify other solutions.

## 6. Updating the plan library

The plan library is updated dynamically in MRL. Fig. 5 summarizes the hybrid representation of the library based on a description logic and a planning logic. The description logic provides the library with an indexing structure based on the subsumption hierarchy of encoded plan specifications, which are represented as concept descriptions and serve as *indices* to *plan entries*. A *plan entry* contains information about a successfully solved planning problem: the plan, the plan specification, and information extracted from the planning process that has led to this plan. This information is represented in the planning logic. A plan entry can be retrieved from and inserted into the library over the *index* to which it is linked. Each index represents an abstract class of planning problems in the application domain. Several plan entries can be linked to the same index when their specifications are encoded by equivalent concept descriptions. In the current implementation, only one plan entry is stored on a "first come-first serve" basis as an instance of the abstract class of planning problems represented by the index. This avoids redundant entries in the plan library. Planning problems belonging to the same abstract class can be solved by modifying the plan stored in the plan entry.

The system starts with the initial plan library containing only the indices *top* and *bottom*. A new plan entry is added to the library under the following conditions:

- no reusable plan has been found and the planner has to generate a plan from first principles,
- the reused plan had to be modified.

The plan library is not updated when:

- a library plan directly solves a current planning problem,
- the index of the current planning problem is already contained in the library.

Let us continue the example from Section 5. According to the above-mentioned conditions, the plan library is updated because the reused plan has been modified. Three sources of information are available for the construction of the plan entry:

(i) the current plan specification,

(ii) the modified plan that meets the specification,

(iii) the proof tree that has been constructed during plan refitting.

The index of the plan entry has already been computed during plan determination. The current plan specification is *completed* before it is added to the plan entry. By completion we understand the computation of the *weakest preconditions* and *strongest goals* of a plan. This means, those preconditions are eliminated from the plan specification which are not necessary for the plan and those goals are added to the plan specification that a plan can achieve as side-effects.

To determine the weakest preconditions and strongest goals of a plan, the planning process that has led to the plan is analyzed. In particular, action axiom schemata that have been applied during the proof are investigated. They specify the *necessary preconditions* of an action and the *effects* it achieves, see Section 2. If the completion process leads to a changed plan specification formula, the encoding of the plan specification is repeated, because a different concept description may result from it and thus, the position of the plan in the library can change caused by altered subsumption relationships.

In the example under consideration, the completion of plan specification $S_{P2}$ leads to a disjunctive precondition reflecting the complete case analysis that has been introduced into the plan with the help of the *if_intro* rule:

$$\begin{aligned}&\mathsf{Plan}_{P2} \wedge \\ &\quad [\mathit{delete\_flag}(\mathit{msg}(x, \mathit{mbox})) = F \wedge \mathit{open\_flag}(\mathit{mbox}) = T] \vee \\ &\quad [\mathit{delete\_flag}(\mathit{msg}(x, \mathit{mbox})) = F \wedge \mathit{open\_flag}(\mathit{mbox}) = F] \\ &\quad \rightarrow \Diamond [\mathit{read\_flag}(\mathit{msg}(x, \mathit{mbox})) = T \wedge \\ &\qquad \mathit{delete\_flag}(\mathit{msg}(x, \mathit{mbox})) = T].\end{aligned}$$

An explicit representation of the possible preconditions for plan **P2** supports the identification of applicable subplans during the plan-interpretation phase. A recomputation of the encoding is not necessary because the conjunctive normal form of the completed precondition formula is logically equivalent to the originally specified precondition in $S_{P2}$.

A major part of a plan entry comprises information that is extracted from the proof tree leading to a plan:

- relation of sequential subplans occurring in conditional plans to their weakest preconditions,
- extraction of sequential body plans occurring in iterative plans,
- relation of atomic actions to the atomic goals achieved by the plan.

In order to relate sequential subplans to their weakest preconditions, the proof tree is analyzed for applications of the rule *if_intro*, see Section 5. In the example, plan refitting has led to the conditional plan

$\text{Plan}_1$ **if** $open\_flag(mbox) = T$ **then** $ex(empty\_action)$
**else** $ex(open(mbox))$

preceding the sequential plan $ex(type(x, mbox)) ; ex(delete(x, mbox))$. Two possible preconditions for this plan are explicitly represented in the completed plan specification formula. Now, each of them is related to the sequential plan that belongs to one of the preconditions. Consequently, the sequential subplan resulting from the **then** branch [16]

$$ex(empty\_action) ; ex(type(x, mbox)) ; ex(delete(x, mbox))$$

is related to the preconditions

$$delete\_flag(msg(x, mbox)) = F \wedge open\_flag(mbox) = T$$

and the sequential subplan resulting from the **else** branch

$$ex(open(mbox)) ; ex(type(x, mbox)) ; ex(delete(x, mbox))$$

is related to its preconditions

$$delete\_flag(msg(x, mbox)) = F \wedge open\_flag(mbox) = F.$$

Plan refitting relies furthermore on information about the relationship between atomic actions and atomic subgoals. When a *current* atomic subgoal has successfully been proved with the help of an *old* subgoal during plan interpretation, plan refitting tries to reuse the action or subplan which achieved the old subgoal in order to achieve the current subgoal. The action instances which achieve atomic goals are extracted from the leaves of the proof tree resulting from the application of action axiom schemata. For the example plan, the following relationships are stored in the plan entry:

- $ex(mail(mbox))$ achieves $open\_flag(mbox) = T$,
- $ex(type(x, mbox))$ achieves $read\_flag(x, mbox) = T$,
- $ex(delete(x, mbox))$ achieves $delete\_flag(msg(x, mbox)) = T$.

The construction of a plan entry is completed by a systematic renaming of variables with internal designators and by a sort-preserving abstraction of constants like sender *Joe* with existentially quantified variables.

Finally, the plan entry is related to its index which uniquely determines its position in the plan library. It is now available to subsequent planning from second principles.

## 7. Related work

The implementation of a second principles planner based on the formal framework as introduced in Section 3 requires design decisions that specify how planning from

[16] The *emptyaction* in this subplan is normally eliminated. It is shown here only for the sakeness of completeness and understandability of the example.

second principles proceeds in detail. In this section, we discuss the most important of these decisions underlying MRL and relate the system to other approaches.

*Meta level versus object level*

Planning from second principles can proceed on a *meta level* or on an *object level*. On the *object level*, previously generated plans are directly reused to solve the current planning problem. This means that the plans as the *objects* of the planning process provide the basis for planning from second principles. Reuse on the *meta level* means to "recycle" knowledge extracted from previous planning processes that represents planning experience in the form of planning tactics, heuristics or strategies.

The commitment of a particular planner to one of these levels is a fundamental design decision. A commitment to the object level leads to case-based planners and reuse systems, e.g., PRIAR [27], CHEF [19] and SPA [21]. A commitment to the meta level leads to adaptive and reactive systems based on learning techniques, e.g., PRODIGY [39] and GRASSHOPPER [36].

MRL proceeds mainly on the object level because it relies on the reuse of stored plans. Meta-level knowledge is reused, e.g., when plan refitting is supplied with information about preconditions on which case analyses have to be performed, see the example in Section 5.

*Skeletal plan refinement versus flexible modification*

When planning from second principles proceeds on the object level, plans are modified in order to construct the desired plan from them. Plan modification can be implemented as *skeletal plan refinement* [14] or as *flexible modification* [21,26].

Skeletal plan refinement computes an appropriate ground-level instantiation for each operator occurring in the abstract skeleton. The admissible modification operations are restricted to *instantiation*, but they can proceed in several hierarchical steps and backtracking may occur. The modified plan is obtained as an instance of the skeleton.

*Flexible modification* as implemented in MRL admits a variety of operations on plans, like the *deletion* and *addition* of operators and control structures.

Skeletal refinement occurs in MRL when the current plan specification has successfully been proved to be a logical consequence of the reused plan specification. In this situation, an instantiation of the library plan will solve the current planning problem and plan modification can be restricted to easy-to-compute substitutions.

*Transformation based versus generation based*

The modification of a plan can be done with the help of transformations [6,19,38] or by extending a first principles planner with the ability to modify plans [21,25,54].

Transformation-based approaches execute a plan in a simulated environment. Failures are classified in a failure hierarchy and resolved by activating transformations on the plan. This approach requires a prediction of all possible failures, i.e., a proof of the *completeness* of the failure hierarchy and the available transformation rules, which is hard to achieve. [17] Further problems are related to the *soundness* and *termination* of the

[17] As an example see the incompleteness proof of CHEF in [20].

transformations. Transformations resolving a failure may introduce other failures, which makes it difficult to ensure that the transformation process does not loop and that the transformed plan is sound, i.e., that it solves the current planning problem.

To overcome these problems, a generation-based approach has been introduced in the PRIAR system [25]. The proof of the completeness of plan modification with respect to the planner is trivial since plan modification can rely on plan generation as a "fall-back" possibility. Soundness and termination are also easy to ensure if the underlying first principles planner possesses these properties.

The modification of a plan in MRL proceeds *generation based*. MRL computes a *plan skeleton* and sends it to plan refitting for completion, which interacts with the generative PHI planner. The plan skeleton preserves those control structures and actions that are assumed to be reusable. The extension of a skeleton to a correct plan requires *flexible* modification operations, which *add*, *delete* or *reorder* operators and control structures. The correctness of deductive plan refitting, which completes the skeleton, ensures that the modified plan is *sound*. Planning knowledge represented by the plan skeleton guides plan refitting and dynamically constrains the search space.

MRL is "complete" with respect to the planner because plan refitting can "fall back" on plan generation. The system is incomplete in the sense that it will not always find a plan if there is one because the use of tactics makes the underlying LLP theorem prover incomplete.

*Conservative versus nonconservative*

A desirable property of plan modification is *conservatism*, which means to "produce a plan ... by minimally modifying [the original plan]" [28]. Minimal modification of a plan implies to preserve the maximal number of applicable operators in a plan skeleton. A critical analysis of conservatism in [43] shows that the computation of such maximal plan skeletons is PSPACE-hard. Therefore, implemented systems including MRL are nonconservative. In order to ensure efficiency of the plan modification process, they rely on polynomial approximations, for example proof tactics for plan interpretation that run in polynomial time, which compute an "entry point" into the search space of possible plans as made explicit by Hanks and Weld [21]. This entry point cannot be guaranteed to be the best, but it is the best the approximation algorithm can compute. It is an open problem whether the maximal applicable subplan is efficiently approximable within a constant ratio. Recent results for similar problems [4,51] seem to hint at a negative result.

*The plan library*

Recently, the representation of plans based on terminological knowledge representation systems has led to several approaches, which extend description logics with new application-oriented representational primitives for the representation of actions and plans.

One such extension is the system RAT [23] which is based on $\mathcal{KRIS}$ [3]. RAT implements reasoning about plans by inferences in the underlying description logic. The system simulates the execution of plans, verifies their applicability in particular situations, and solves tasks of temporal projection.

An application of description logics to tasks of plan recognition is developed in T-REX [56]. Plans in T-REX may contain conditions and iterations as well as nondeterminism in the form of disjunctive actions.

The plan library can be *static* as well as *dynamic* in MRL. A static library comprises user-predefined typical plans. The system retrieves these plans for reuse, but does not add new plans to the library. A dynamic plan library grows during the lifetime of the system, i.e., MRL starts with an empty library and incrementally adds new plan entries to it.

The main advantage in using a description logic as a query language to the plan library as in MRL lies in the novel solution to the indexing problem and in the theoretically well-founded properties of the retrieval algorithm. For the first time, retrieval guarantees that *solutions* are found in a library in *polynomial* time. This contrasts to approaches that are restricted to retrieve "reasonable similar past cases ... within limited bounded resources" (cf. [54, p. 103]). Furthermore, an indexing of plan libraries based on the lattice structure provided by the *subsumption* hierarchy overcomes problems occurring in indexing schemes based on discrimination networks. On one hand, discrimination networks fail in indexing complex plan specifications because they are restricted to cope with conjunctions of literals. On the other hand, retrieval algorithms working on discrimination networks are often faced with an exponentially growing input set. For example, given a goal state containing $n$ atomic subgoals, the retrieval algorithm developed in [54] first searches a plan covering these $n$ subgoals. If this fails, it computes all subsets of subgoals of cardinality $n-1$, then $n-2$ and so on until it takes the atomic subgoals as input. This means, the retrieval algorithm takes the power set of $n$ except the empty set as input in the worst case, which is $2^n - 1$. Retrieval based on concept descriptions avoids such problems because existing relationships between sets of subgoals are reflected in the subsumption hierarchy.

## 8. Conclusion

We have presented a logic-based approach to planning from second principles, which relies on a systematic decomposition of the planning process with the help of a four-phase model. Deductive inference processes with clearly defined semantics formalize each phase. The formal model is independent of a particular planning formalism and makes no commitments to application domains, implementational details, the nature of plans or planning situations.

Plan modification yields provably correct modified plans and enables a second principles planner to reuse plans containing control structures like conditionals and iterations.

Reusable plans are retrieved from a dynamically updated plan library using a description logic as query language to the library. The plan library can be indexed basing on a lattice structure and retrieval is formalized using a KL-ONE-like classifier which is guaranteed to find existing solutions.

The formal framework has led to an implemented system with predictable behavior. Furthermore, in contrast to heuristic approaches, theoretical properties like the correctness, completeness and efficiency of the inference procedures can be proved.

**Acknowledgements**

This work was supported by the German Ministry for Research and Technology (BMFT) under contract ITW 9000 8 as part of the PHI project. I want to thank the members of the PHI group, Mathias Bauer, Susanne Biundo, Dietmar Dengler, and Gabriele Paul for their interest in my work and for fruitful discussions. Furthermore, I am indebted to Wolfgang Wahlster for his advice and support, and to Bernhard Nebel and Hans-Jürgen Ohlbach for commenting on the draft version. The anonymous reviewers made very detailed comments helping me during the revision of the paper.

## Appendix A. Proofs of theorems

**Theorem 14.** $\mathbf{P}_{\text{old}}$ *meets* $S_{\text{new}}$ *if* $Ax \models S_{\text{old}} \rightarrow S_{\text{new}}$.

**Theorem 15.** $Ax \models S_{\text{old}} \rightarrow S_{\text{new}}$ *if*

$$Ax \models pre_{\text{new}} \rightarrow pre_{\text{old}} \quad \textit{and} \quad Ax \models \Diamond goal_{\text{old}} \rightarrow \Diamond goal_{\text{new}}.$$

**Proof.** We have to prove the following sequent

$$Ax, \mathbf{P}_{\text{old}} \wedge pre_{\text{old}} \rightarrow \Diamond goal_{\text{old}} \Rightarrow \mathbf{P}_{\text{old}} \wedge pre_{\text{new}} \rightarrow \Diamond goal_{\text{new}} \tag{A.1}$$

in which both planvariables are already instantiated with the reused plan formula $\mathbf{P}_{\text{old}}$. We start by applying the sequent rule $r\rightarrow$ which leads to sequent (A.2). To this sequent, the rule $l\rightarrow$ is applied leading to sequents (A.3) and (A.4).

$$\bullet\ \frac{\Gamma \Rightarrow A, \Delta \quad \Gamma, B \Rightarrow \Delta}{\Gamma, A \rightarrow B \Rightarrow \Delta}\ l\rightarrow \qquad \bullet\ \frac{\Gamma, A \Rightarrow B, \Delta}{\Gamma \Rightarrow A \rightarrow B, \Delta}\ r\rightarrow$$

$$Ax, \mathbf{P}_{\text{old}} \wedge pre_{\text{old}} \rightarrow \Diamond goal_{\text{old}}, \mathbf{P}_{\text{old}} \wedge pre_{\text{new}} \Rightarrow \Diamond goal_{\text{new}}, \tag{A.2}$$

$$Ax, \mathbf{P}_{\text{old}} \wedge pre_{\text{new}} \Rightarrow \mathbf{P}_{\text{old}} \wedge pre_{\text{old}}, \Diamond goal_{\text{new}}, \tag{A.3}$$

$$Ax, \mathbf{P}_{\text{old}} \wedge pre_{\text{new}}, \Diamond goal_{\text{old}} \Rightarrow \Diamond goal_{\text{new}}. \tag{A.4}$$

Sequents (A.3) and (A.4) have to be expanded further. We start with sequent (A.3) and apply the rule $l\wedge$, followed by $r\wedge$ and obtain sequents (A.5) and (A.6).

$$Ax, \mathbf{P}_{\text{old}}, pre_{\text{new}} \Rightarrow \mathbf{P}_{\text{old}}, \Diamond goal_{\text{new}}, \tag{A.5}$$

$$Ax, \mathbf{P}_{\text{old}}, pre_{\text{new}} \Rightarrow pre_{\text{old}}, \Diamond goal_{\text{new}}. \tag{A.6}$$

To sequent (A.4), the rule $l\wedge$ is applied leading to sequent (A.7).

$$Ax, \mathbf{P}_{\text{old}}, pre_{\text{new}}, \Diamond goal_{\text{old}} \Rightarrow \Diamond goal_{\text{new}}. \tag{A.7}$$

The original sequent is proved iff the three sequents are axioms, i.e., antecedent and succedent contain a common formula. This holds for sequent (A.5) immediately, while sequents (A.6) and (A.7) lead to the two subproofs

$$Ax, pre_{\text{new}} \Rightarrow pre_{\text{old}} \quad \text{and} \quad Ax, \Diamond goal_{\text{old}} \Rightarrow \Diamond goal_{\text{new}}. \qquad \square$$

## Appendix B. Summary of sequent rules

$$\Gamma, A \Rightarrow A, \Delta,$$

$$\frac{\Gamma \Rightarrow A, \Delta \quad \Gamma, B \Rightarrow \Delta}{\Gamma, A \rightarrow B \Rightarrow \Delta}\ l\rightarrow \qquad \frac{\Gamma, A \Rightarrow B, \Delta}{\Gamma \Rightarrow A \rightarrow B, \Delta}\ r\rightarrow$$

$$\frac{\Gamma, A, B \Rightarrow \Delta}{\Gamma, A \wedge B \Rightarrow \Delta}\ l\wedge \qquad \frac{\Gamma \Rightarrow A, \Delta \quad \Gamma \Rightarrow B, \Delta}{\Gamma \Rightarrow A \wedge B, \Delta}\ r\wedge$$

$$\frac{\Gamma, A[c/x] \Rightarrow \Delta}{\Gamma, \forall x A \Rightarrow \Delta}\ l\forall \qquad \frac{\Gamma \Rightarrow \Delta, A[a/x]}{\Gamma \Rightarrow \Delta, \forall x A}\ r\forall$$

$$\frac{\Gamma^*, A \Rightarrow \Delta^*}{\Gamma, \Diamond A \Rightarrow \Delta}\ l\Diamond \qquad \frac{\Gamma \Rightarrow A, \Delta}{\Gamma \Rightarrow \Diamond A, \Delta}\ r\Diamond$$

$$\Gamma^* \stackrel{\text{def}}{=} \{\Box B \mid \Box B \in \Gamma\} \qquad \Delta^* \stackrel{\text{def}}{=} \{\Diamond B \mid \Diamond B \in \Delta\}$$

$$\frac{pre, \mathsf{Skeleton} \Rightarrow \Diamond goal}{pre, \mathsf{Plan} \Rightarrow \Diamond goal}\ seq\_inst$$

$$\frac{pre, \mathsf{Plan}_1\,;\mathsf{Plan}_2 \Rightarrow \Diamond goal}{pre, \mathsf{Plan} \Rightarrow \Diamond goal}\ sequence\_assumption$$

$$\frac{pre, \mathsf{Plan}_1 \Rightarrow \Diamond [goal_1 \wedge \mathsf{OF} \wedge pre'] \quad pre', \mathsf{Plan}_2 \Rightarrow \Diamond goal_2}{pre, \mathsf{Plan}_1\,;\mathsf{Plan}_2 \Rightarrow \Diamond [goal_1 \wedge goal_2]}\ effect\_split$$

$$\frac{pre, \mathsf{Plan}_1 \Rightarrow \Diamond [goal_{\text{new}} \wedge \mathsf{OF} \wedge pre'] \quad pre', \mathsf{Plan}_2 \Rightarrow \Diamond goal}{pre, \mathsf{Plan}_1\,;\mathsf{Plan}_2 \Rightarrow \Diamond goal}\ effect\_intro$$

$$\frac{pre, \mathbf{if}\ (cond, \mathsf{Plan}_A, \mathsf{Plan}_B) \Rightarrow \Diamond goal}{pre, \mathsf{Plan} \Rightarrow \Diamond goal}\ if\_intro$$

$$\frac{pre, cond, \mathsf{Plan}_A \Rightarrow goal \quad pre, \neg cond, \mathsf{Plan}_B \Rightarrow \Diamond goal}{pre, \mathbf{if}\ (cond, \mathsf{Plan}_A, \mathsf{Plan}_B) \Rightarrow \Diamond goal}\ if\_splitting$$

## References

[1] W. Anderson, J. Hendler, M. Evett and B. Kettler, Massively parallel matching of knowledge structures, in: H. Kitano and J. Hendler, eds., *Massively Parallel Artificial Intelligence* (MIT Press, Cambridge, MA, 1994) 53–71.

[2] F. Baader and B. Hollunder, KRIS: knowledge representation and inference system, *SIGART Bull.* **2** (2) (1991) 8–15.

[3] F. Baader, B. Hollunder, B. Nebel, H.-J. Profitlich and E. Franconi, An empirical analysis of optimization techniques for terminological representation systems, or making KRIS get a move on, in: *Proceedings 3rd International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, MA (1992) 270–281.

[4] C. Bäckström, Finding least constrained plans and optimal parallel executions is harder than we thought, in: C. Bäckström and E. Sandewall, eds., *Current Trends in AI Planning* (IOS Press, Amsterdam, 1994).
[5] M. Bauer, S. Biundo, D. Dengler, J. Koehler and G. Paul, PHI—a logic-based tool for intelligent help systems, in: *Proceedings IJCAI-93*, Chambery (1993) 460–466.
[6] M. Beetz and D. McDermott, Improving robot plans during their execution, in: *Proceedings 2nd International Conference on Artificial Intelligence Planning Systems* (1994) 7–12.
[7] S. Biundo and D. Dengler, The logical language for planning LLP , Research Rept., German Research Center for Artificial Intelligence, Saarbrücken (1996).
[8] S. Biundo, D. Dengler and J. Koehler, Deductive planning and plan reuse in a command language environment, in: *Proceedings ECAI-92*, Vienna (1992) 628–632.
[9] A. Borgida and P. Patel-Schneider, A semantics and complete algorithm for subsumption in the CLASSIC description logic, *J. Artif. Intell. Res.* **1** (1994) 277–308.
[10] R. Brachmann and H. Levesque, The tractability of subsumption in frame based description languages, in: *Proceedings AAAI-84*, Austin, TX (1984) 34–37.
[11] R. Constable, *Implementing Mathematics with the Nuprl Proof Development System* (Prentice Hall, Englewood Cliffs, NJ, 1986).
[12] D. Dengler, An adaptive deductive planning system, in: *Proceedings ECAI-94*, Amsterdam (1994) 610–614.
[13] E. Feigenbaum, The simulation of natural learning behavior, in: E. Feigenbaum and J. Feldman, eds., *Computers and Thought* (McGraw-Hill, New York, 1963).
[14] P. Friedland and Y. Iwasaki, The concept and implementation of skeletal plans, *J. Automated Reasoning* **1** (1985) 161–208.
[15] D. Gabbay, Declarative past and imperative future: executable temporal logic for imperative systems, in: H. Barringer and A. Pnueli, eds., *Proceedings Colloquium on Temporal Logic in Specification*, Altrinchham, Lecture Notes in Computer Science **398** (Springer, New York, 1989) 402–450.
[16] J. Gallier, *Logic for Computer Science: Foundations of Automatic Theorem Proving* (Wiley, New York, 1987).
[17] R. Givan and D. McAllester, New results on local inference relations, in: *Proceedings 3rd International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, MA (1992) 403–412.
[18] C. Green, Application of theorem proving to problem solving, in: *Proceedings IJCAI-69*, Washington, DC (1969) 219–239.
[19] K. Hammond, Explaining and repairing plans that fail, *Artif. Intell.* **45** (1990) 173–228.
[20] S. Hanks and D. Weld, The systematic plan adaptor: a formal foundation of case-based planning, Tech. Rept. 92-09-04, Department of Computer Science and Engineering, University of Washington, Seattle, WA (1992).
[21] S. Hanks and D. Weld, Systematic adaptation for case-based planning, in: *Proceedings 1st International Conference on Artificial Intelligence Planning Systems* (1992) 96–105.
[22] S. Hanks and D. Weld, A domain-independent algorithm for plan adaptation, *J. Artif. Intell. Res.* **2** (1995) 319–360.
[23] J. Heinsohn, D. Kudenko, B. Nebel and H.-J. Profitlich, Integration of action representation in terminological logics, in: C. Peltason, K. Luck and C. Kindermann, eds., *Proceedings Terminological Logic Users Workshop*, KIT-Report 95, TU Berlin (1991).
[24] M. Heisel, W. Reif and W. Stephan, Tactical theorem proving in program verification, in: *Proceedings 10th International Conference on Automated Deduction*, Kaiserslautern, Lecture Notes in Artificial Intelligence **449** (Springer, Berlin, 1990) 117-131.
[25] S. Kambhampati, Flexible reuse and modification in hierarchical planning: a validation-structure-based approach, PhD thesis MD 207 42-3411, Center for Automation Research, Computer Vision Laboratory, University of Maryland, College Park, MD (1989).
[26] S. Kambhampati, A theory of plan modification, in: *Proceedings AAAI-90*, Boston, MA (1990) 176–182.
[27] S. Kambhampati and J. Hendler, Control of refitting during plan reuse, in: *Proceedings IJCAI-89*, Detroit, MI (1989) 943-949.
[28] S. Kambhampati and J. Hendler, A validation-structure-based theory of plan modification and reuse, *Artif. Intell.* **55** (1992) 193–258.

[29] J. Koehler, An application of terminological logics to case-based reasoning, in: *Proceedings 4th International Conference on Principles of Knowledge Representation and Reasoning*, Bonn (1994) 351–362.
[30] J. Koehler, Avoiding pitfalls in case-based planning, in: *Proceedings 2nd International Conference on Artificial Intelligence Planning Systems* (1994) 104–109.
[31] J. Koehler, Correct modification of complex plans, in: *Proceedings ECAI-94*, Amsterdam (1994) 605–609.
[32] J. Koehler, Flexible plan reuse in a formal framework, in: C. Bäckström and E. Sandewall, eds., *Current Trends in AI Planning* (IOS Press, Amsterdam, 1994) 171–184.
[33] J. Koehler and R. Treinen, Constraint deduction in an interval-based temporal logic, in: M. Fisher and R. Owen, eds., *Executable Modal and Temporal Logic*, Lecture Notes in Artificial Intelligence **897** (Springer, Berlin, 1995) 103–117.
[34] J. Kolodner, *Case-Based Reasoning* (Morgan Kaufman, Los Altos, CA, 1993).
[35] F. Kröger, *Temporal Logic of Programs* (Springer, Heidelberg, 1987).
[36] C. Leckie and I. Zuckerman, An inductive approach to learning search control rules for planning, in: *Proceedings IJCAI-93*, Chambery (1993) 1100–1105.
[37] Z. Manna and R. Waldinger, A theory of plans, in: M. Georgeff and A. Lansky, eds., *Reasoning about Actions and Plans: Proceedings 1986 Workshop* (Morgan Kaufmann, Los Altos, CA, 1987) 11–45.
[38] D. McDermott, Planning and acting, *Cognit. Sci.* **2** (1978) 71–109.
[39] S. Minton, Quantitative results concerning the utility of explanation-based learning, in: *Proceedings AAAI-88*, St. Paul, MN (1988) 564–569.
[40] B. Moszkowski and Z. Manna, Reasoning in interval temporal logic, in: E. Clarke and D. Kozen, eds., *Proceedings Conference on Logics of Programs*, Lecture Notes in Computer Science **164** (Springer, New York, 1983).
[41] B. Nebel, *Reasoning and Revision in Hybrid Representation Systems*, Lecture Notes in Artificial Intelligence **422** (Springer, Berlin, 1990).
[42] B. Nebel, Terminological reasoning is inherently intractable, *Artif. Intell.* **43** (1990) 235–249.
[43] B. Nebel and J. Koehler, Plan modification versus plan generation: a complexity-theoretic perspective, in: *Proceedings IJCAI-93*, Chambery (1993) 1436–1441.
[44] B. Nebel and J. Koehler, Plan reuse versus plan generation: a theoretical and empirical analysis, *Artif. Intell.* **76** (1995) 427–454.
[45] H.-J. Ohlbach, Semantics-based translation methods for modal logics, *J. Logic Comput.* **1** (1991) 691–775.
[46] L. Paulson, Isabelle: the next 700 theorem provers, in: P. Odifredi, ed., *Logic and Computer Science* (Academic Press, New York, 1990).
[47] R. Reiter, Proving properties of states in the situation calculus, *Artif. Intell.* **64** (1993) 337–351.
[48] C. Riesbeck and R. Schank, *Inside Case-Based Reasoning* (Lawrence Erlbaum, Hillsdale, NJ, 1989).
[49] R. Rosner and A. Pnueli, A choppy logic, in: *Proceedings Symposium on Logic in Computer Science*, Cambridge, MA (1986).
[50] M. Schmidt-Schauß and G. Smolka, Attributive concept descriptions with complements, *Artif. Intell.* **48** (1991) 1–26.
[51] B. Selman, Near-optimal plans, tractability, and reactivity, in: *Proceedings 4th International Conference on Principles of Knowledge Representation and Reasoning*, Bonn (1994) 521–529.
[52] W. Stephan and S. Biundo, Multilevel refinement planning in an interval-based temporal logic, in: *Proceedings 7th Portuguese Conference on Artificial Intelligence*, Lecture Notes in Artificial Intelligence **990** (Springer, Berlin, 1995) 483–487.
[53] P. Traverso, A. Cimatti and L. Spalazzi, Beyond the single planning paradigm: introspective planning, in: *Proceedings ECAI-92*, Vienna (1992) 643–647.
[54] M. Veloso, *Planning and Learning by Analogical Reasoning*, Lecture Notes in Artificial Intelligence **886** (Springer, Berlin, 1994).
[55] L. Wallen, *Automated Deduction in Nonclassical Logics* (MIT Press, Cambridge, 1989).
[56] R. Weida and D. Litman, Subsumption and recognition of heterogeneous constraint networks, in: *Proceedings Tenth IEEE Conference on Artificial Intelligence for Applications* (1994).