PHI—A Logic-Based Tool
for Intelligent Help Systems *

M. Bauer and **S. Biundo** and **D. Dengler** and **J. Koehler** and **G. Paul**

German Research Center for Artificial Intelligence (DFKI)

Stuhlsatzenhausweg 3, 66123 Saarbrücken

e-mail: {lastname}@dfki.uni-sb.de

Abstract

We introduce a logic-based system which improves the performance of intelligent help systems by supplying them with plan generation and plan recognition components. Both components work in close mutual cooperation. There are two modes of cross-talk between them, one where plan recognition is done on the basis of abstract plans provided by the planner and the other where optimal plans are generated based on recognition results. The examples which are presented are taken from an operating system domain, namely from the UNIX mail domain.

1 Introduction

Intelligent help systems aim at providing advanced active help to the users of complex software systems (cf. [Breuker, 1990; Thies and Berger, 1992; Norvig *et al.*, 1993]). The performance of these help systems can be considerably improved if they are supplied with *plan recognition* and *plan generation* capabilities. Observing a user and recognizing his goals enables the system to help by taking into account the current state of the system as well as the user's level of education and current behavior. Moreover, if a planning capability is available user-specific support can be given by proposing appropriate plans which exactly is what the PHI system aims to achieve.

PHI (cf. the figure below) is a tool for intelligent help systems. It provides both a plan recognizer and a planning component and one of its main characteristics consists in the close mutual cooperation between the two components.

There are several *cross-talk modes*. The first one is devoted to realizing plan recognition on the basis of *abstract* plans produced by the planner. Abstract plans are those which represent a variety of "concrete" observable action sequences by admitting several degrees of freedom like *variables* (abstracting from the objects involved), *abstract commands* (abstracting from the names

of actions which have the same effects), or *temporal abstraction* (abstracting from the point in time at which an action occurs).

The generation of plans is based on standard assumptions concerning goals that typically occur or are specific to a certain user. Abstract plans are generated from these formal plan specifications. In doing so, the planner not only performs planning from first principles but is able to *reuse* already existing plans which are stored in a library (planning from second principles). The plans provided serve as plan hypotheses in the recognition process. Taking abstract plans instead of concrete ones keeps the hypothesis space of manageable size. The plan hypotheses are passed to the recognition component where they are provided with numerical values which reflect the probabilities of their being confirmed by the subsequent observations. These a priori probabilities mirror a specific user's behavior, and are taken from the user model. Having observed the user's actions step by step the plan recognizer consequently tries to confirm the plan hypotheses by proving that the action sequence observed up to now is an admissible "instance". Hypotheses which are not confirmed are rejected and with that the probability distribution of the hypothesis space changes dynamically.

In the first cross-talk mode the plan recognizer is able to determine the most likely plan a user follows by carrying out appropriate "instantiations" on valid plan hypotheses. Thus, services like *semantic plan completion* can be offered at any time during the observation process.

The second cross-talk mode is devoted to providing the user with *optimal* plans whenever suboptimal behavior has been recognized or aid has explicitly been sought.

The system is completely *logic-based*. It requires a proper axiomatization of the basic commands of the application system and certain domain constraints. The logic LLP which we have developed for that purpose combines features of both traditional programming and temporal logics. The plan generation and recognition components are special purpose inference procedures. Plan generation is done *deductively* using a sequent calculus for LLP, whereas plan recognition follows an *abductive* principle.

The application domain, from which we present examples, is a subset of the operating system UNIX, namely its
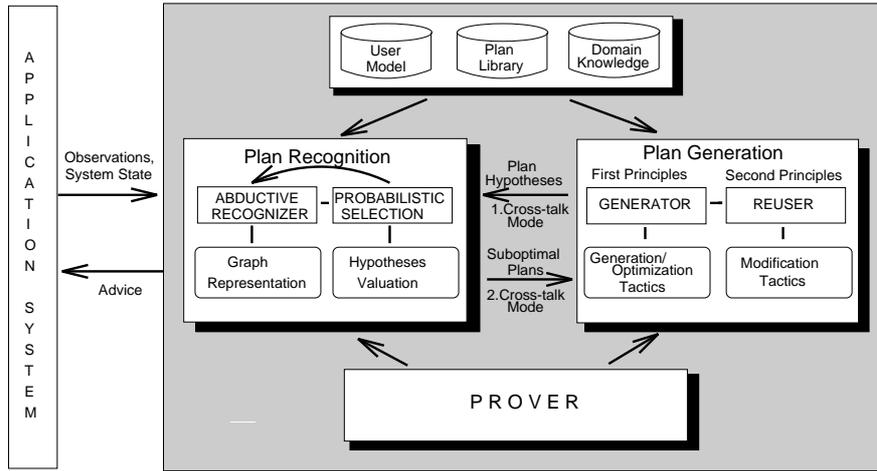
mail system, where commands like *type*, *delete*, or *save* manipulate objects like *messages* or *mailboxes*.

The paper is organized as follows: After a short introduction to the formal framework in section 2 we describe the plan generation and recognition components in sections 3 and 4, respectively. We demonstrate by means of an example how the system works in the first cross-talk mode. Finally, we conclude with some remarks in section 5.

2   The Formal Framework

Plan generation and plan recognition are carried out on a common logical basis. The *logical language for planning* (LLP) [Biundo and Dengler, 1993], which we have developed for this purpose, combines features of *Choppy Logic* [Rosner and Pnueli, 1986] with the *Temporal Logic for Programs* [Kröger, 1987]. This entails the consideration of plans as programs as has also been proposed by other authors (cf. [Green, 1969], [Bibel, 1986], [Manna and Waldinger, 1987]). LLP is an interval-based modal temporal logic. It provides the modal operators $\bigcirc$ (next), $\Diamond$ (sometimes), $\Box$ (always), and the binary modal operator **;** (chop), which expresses the sequential composition of formulas. Besides these operators control structures (e.g., conditionals) are also available, as in programming logics. Basic actions, which in our example domain are the elementary mail commands, are axiomatized like assignment statements in programming logics. The state changes which they perform are reflected in changing the values of certain variables. The *type* command, for example, is represented by the following axiom schema:

$$\forall x\, [\,[\, of(mb) = T \wedge df(x, mb) = F \wedge$$
$$\mathsf{P}\, \frac{rf(x, mb)}{T} \wedge EX(type(x, mb))\,] \rightarrow \bigcirc \mathsf{P}]$$

where P is a metavariable for formulas. The schema states: If the mailbox mb is open (i.e., the open-flag $of(mb)$ equals true) and message x is not yet deleted (i.e., the delete-flag $df(x, mb)$ equals false) and we execute the *type* command then formula P holds in the next state, provided the formula we obtain by replacing each ocurrence of the term $rf(x, mb)$ in P by T (i.e., the *weakest precondition* of P w.r.t. $type(x, mb)$) has held before.

The substitution instructions correspond to the effect of the *type* command. Replacing P by $rf(x, mb) = T$ we obtain the following action axiom:

$$\forall x\, [\,[\, of(mb) = T \wedge df(x, mb) = F \wedge$$
$$EX(type(x, mb))\,] \rightarrow \bigcirc rf(x, mb) = T\,]$$

It states that if a certain message x in a current mailbox mb has not yet been deleted and we execute the *type* command then the message is read in the next state (i.e., the read-flag $rf(x, mb)$ is set to *true*).

Plans are represented by a certain class of LLP formulas. Besides basic actions (denoted using the *execute* predicate EX) they contain the *chop* operator, control structures, and also temporal abstractions.

Plan specifications are LLP formulas of the form as follows: [*preconditions* $\wedge$ Plan] $\rightarrow$ *goals* i.e., if the *preconditions* hold in a situation where we carry out Plan, then the *goals* will be reached. For example the following formula

$$of(M) = T \wedge \mathsf{Plan} \quad \rightarrow \quad \Diamond[displ = headers(M)\, \wedge$$
$$\Diamond\, [rf(x, M) = T]] \qquad (1)$$

specifies the plan "Display the content of mailbox M on the screen and then read message x". Plan is a metavariable for a plan formula.

Plan generation (cf. section 3) is carried out by *constructively* proving the specification formula. While proving the specification formula the plan metavariable Plan is replaced by a plan (formula) which satisfies the specification. Proving (1), for example, results in the following plan:

$$EX(header(M))\ ;$$
$$\mathbf{if}\ df(x, M) = T\ \mathbf{then}\ EX(undelete(x, M))\ ;$$
$$EX(type(x, M)) \qquad\qquad (2)$$

As the UNIX mail language does not of course provide any control structures, this plan has to be considered as abstract, apart from the fact that the variable x occurs in it.

In section 3 we will see how the conditional plan above is generated from first principles. We can then demonstrate planning from second principles by reusing and

modifying this plan so that it meets a new specification:

$$of(M) = T \wedge df(x, M) = F \wedge \mathsf{Plan} \rightarrow$$
$$\Diamond [displ = headers(M) \wedge \Diamond [rf(x, M) = T \wedge$$
$$\Diamond [df(x, M) = T]]] \tag{3}$$

$$EX(header(M)); EX(type(x, M)) ;$$
$$EX(delete(x, M)) \tag{4}$$

is obtained as a result. In a plan recognition example
(cf. section 4) this plan can then serve as one of the plan
hypotheses.

3 Plan Generation

The planning system (cf. [Biundo *et al.*, 1992]) works by
using techniques of planning from both first and second
principles. Planning from first principles begins with a
plan specification. The plan is generated on the basis
of the *domain knowledge* provided. Planning from sec-
ond principles adds the ability to incorporate previously
generated plans and the problem solving knowledge ob-
tained thereby. In the first cross-talk mode, abstract
plans are generated in order to provide the plan recog-
nizer with plan hypotheses. To generate these hypothe-
ses, the planner works from second principles by reusing
formerly generated plans.

3.1 Planning from First Principles

By using a sequent calculus for LLP (cf. [Biundo and
Dengler, 1993]) the plan generator tries to find a con-
structive proof for the plan specification formula so that
an instantiation for the plan metavariable can be ob-
tained. We thus have a plan the execution of which is
sufficient to reach the goals specified, i.e., a plan which
meets the specification. Following the paradigm of *tac-
tical theorem proving* (cf. [Constable, 1986], [Heisel *et
al.*, 1990], [Paulson, 1990]) the proof is guided by special
planning tactics written in a metalogical tactic language.

As for plan specification (1), the proof is carried out by
dividing the specification formula into subformulas, i.e.,
those representing single subgoals which the plan has
to reach. We can simultaneously introduce a structure
into the plan metavariable Plan, which states that Plan
should consist of at least two subplans: $\mathsf{Plan} \equiv \mathsf{P}_1 ; \mathsf{P}_2$.

Let us now consider the generation of a plan for P_2.
The corresponding subgoal reads:

$$of(M) = T \wedge \mathsf{P}_2 \rightarrow \Diamond rf(x, M) = T$$

Usually subgoals of this type are proven by using non-
logical axioms which describe basic actions. Thereby,
the plan metavariable is instantiated by a basic plan for-
mula. The instance of the *type* axiom below is selected
because it can reach the desired goal of setting rf to T:

$$of(M) = T \wedge df(x, M) = F \wedge EX(type(x, M))$$
$$\rightarrow \bigcirc rf(x, M) = T$$

The preconditions of this action however must hold in
order to make the axiom applicable. One of these precon-
ditions is missing from the subgoal above. Following a
deductive version of the *means-ends analysis* (cf. [Fikes

and Nilsson, 1971], [Nilsson, 1980]) we therefore intro-
duce an additional subplan which produces the missing
precondition. Thus, P_2 becomes the composition of a
one-armed conditional and a subplan P_4, respectively:

$$\mathsf{P}_2 \equiv [\mathbf{if}\ df(x, M) = T\ \mathbf{then}\ \mathsf{P}_3]; \mathsf{P}_4$$

The new subgoal obtained is:

$$of(M) = T \wedge \mathbf{if}\ df(x, M) = T\ \mathbf{then}\ \mathsf{P}_3 \rightarrow$$
$$\bigcirc [of(M) = T \wedge df(x, M) = F]$$

To properly instantiate P_3 an instance of the *undelete*
action axiom can be used; this tells us that the execution
of $undelete(x, M)$ makes $df(x, M) = F$ true in the next
state, should it not have held before. In a similar way P_4
can now be instantiated by using the *type* action axiom.

The overall plan which results after the proof tree has
been completed and all plan metavariables have been
instantiated, is the plan given by formula (2) above. It
clearly meets the specification in (1).

In addition to subgoals whose proof leads to instan-
tiations of the plan metavariables, as in the above ex-
amples, so-called *plan assertions* must also be proven.
These represent certain properties which are required
by the plan to be generated. A typical example in our
case is the fact that the formula $of(M) = T$—which
acts as a precondition to the whole plan—does survive
the execution of subplan P_1. This fact is proven by re-
gression where we generate the weakest preconditions of
$of(M) = T$ w.r.t. all basic actions occurring in P_1. In
our system, planning from first principles is, like sev-
eral other approaches to deductive planning (cf. [Green,
1969], [Bibel, 1986], [Manna and Waldinger, 1987]) close-
ly related to work done on *deductive program synthesis*
where *programs* are generated by proofs (cf. [Manna and
Waldinger, 1980], [Heisel *et al.*, 1991], [Biundo, 1992]).

3.2 Planning from Second Principles

The ability of a planner to modify a plan is considered as
a valuable tool for improving the efficiency of planning
by avoiding the repetition of the same planning effort
because instead of generating a plan from scratch, plan
reuse tries to exploit knowledge stored in previously gen-
erated plans.

The reuser first takes the current specification and
searches in a *plan library* for a plan which can be reused
as its solution. Since we concentrate on plan modifica-
tion in this paper, we suppose that the search in the
plan library terminates successfully with a plan specifi-
cation and describe how the reuser verifies whether the
plan belonging to it provides a solution to the current
planning problem. The verification is carried out by a
formal proof in which the *prover* verifies that at least the
preconditions the plan requires hold in the current sit-
uation and that at most the goals achieved by the plan
are required as current goals.

If the proof succeeds, the plan provides a provably
sound solution to the current planning problem; if it fails,
the plan has to be modified.

The *modification tactics* analyze the failed proof and
modify the plan using information from the generation
process that lead to this plan.

Let us assume, for example, that specification (3) is given to the planner in the first cross-talk mode. Planning from second principle starts and tries to reuse plan (2).

Comparing it with specification (1) it is obvious that more preconditions are given, but even more goals are required in (3). In this case, the prover reports a failure because more goals are required in specification (3) than are achieved by the plan. The modification tactic identifies the missing subgoal $df(x, M) = T$ for which a subplan has to be generated from first principles. Furthermore, it has to inspect the temporal structure of the plan to be reused in order to determine the point in time at which this subplan has to be inserted. For this purpose, explicit representations of the temporal models of both specifications are constructed and compared during the proof.

A plan is a solution if it achieves at least all the goals that are required in the current specification, i.e., if the plan achieves some additional subgoals it is still considered to be a solution. In some applications however, plans have to be minimal in the sense of achieving exactly the goals required. The plan reuse component is able to perform the necessary optimizations in these cases.

In the example, the reuser detects that the case analysis in the reused plan is superfluous because the condition on which it depends is explicitly given in the specification. Therefore, the conditional can be deleted from the plan. The result of the modification process is a *plan skeleton* for a sequential plan

$$EX(header(Mbox)); EX(type(x, Mbox)); \mathsf{Plan}_1$$

containing the reusable subplan identified during the proof and a meta variable Plan_1 as a "placeholder" for the completing subplan which has to be generated in order to reach the additional goal.

The generator uses the plan skeleton as a partial instantiation of the plan metavariable Plan in specification (3). This simplifies the constructive proof of the specification: The partial proof tree for which an instantiation of the metavariable is already known can be easily expanded without further search effort. To replace the metavariable Plan_1 occurring in the skeleton, the generator has to plan from first principles leading to the instantiation $EX(delete(x, M))$. The interleaving of proof tree reconstruction and generation ensures that the modified plan provides a provably sound solution to the current plan specification that can be sent to the plan recognizer as a plan hypothesis.

The approach we follow investigates plan reuse in the general context of deductive planning and has been described in more detail in [Biundo *et al.*, 1992; Koehler, 1992]. Other current approaches investigate plan reuse and modification in the framework of classical STRIPS-like planners, e.g., the hierarchical planner and modification system PRIAR [Kambhampati and Hendler, 1992], or in the framework of case-based reasoning, e.g., the systems SPA [Hanks and Weld, 1992] or CHEF [Hammond, 1990]. The experiments reported by some of the authors give evidence that plan reuse might indeed be more efficient than planning from scratch. How far these results

generalize is studied in a complexity-theoretic analysis of plan modification vs. plan generation by Nebel and Koehler (cf. these proceedings). In contrast to practical experiences it turned out that plan modification is not uniformly as easy as planning from scratch.

3.3 Generating Optimal Plans

The second cross-talk mode is concerned with the generation of optimal and user-satisfactory plans. The generator receives a plan specification which either belongs to a plan recognized as suboptimal by the plan recognition component or is derived from a request for passive help.

Planning in this mode is based on a dynamically changing adjustment of the generation process triggered by *plan quality criteria* derived from the user model. The generator considers, e.g., the user's preferences, his knowledge about the domain, and his typical behavior in order to generate satisfactory plans for him. It produces a user-adapted concrete plan that meets the specification and is as short as possible according to the number of basic actions used. Since planning is done deductively the adjustment essentially places a restriction on the sets of nonlogical axioms and rules.

If, on the basis of the current plan quality criteria, no plan can be found, then the criteria must be minimally changed in order to generate a plan. The necessary deviations are recorded and can be used by a tutorial system to teach the user accordingly. In the case of a recognized suboptimal plan, the generated optimal plan is, e.g., the basis for an active user support of the help system. Generation of optimal plans is only carried out from first principles because the reuse of concrete plans requires consideration of dynamically changing plan quality criteria which can contradict the aim of making planning more efficient.

4 Plan Recognition

The recognition of plans in this logic-based context is realized by a *generalized abductive process* with a *probabilistic valuation* of hypotheses. Starting from plan hypotheses synthesized by the plan generation component and observations of user actions, an attempt is made to identify a hypothesis describing the user's pursued plan. The use of probabilistic reasoning allows us to determine *one* "best" hypothesis to offer user-specific help, e.g., by doing semantic plan completion.

4.1 The Abductive Recognizer

Plan recognition, which is the identification of a user's behavior given an observed goal or action, can be viewed as an inherently abductive problem, if a plan hypothesis P is interpreted as an assumption explaining the observed action a, i.e., $\mathcal{T} \cup \{P\} \models a$, where $\mathcal{T}$ describes the domain knowledge (e.g., [Appelt and Pollack, 1990], [Shanahan, 1989], [Helft and Konolige, 1990],[Wærn, 1992]). P is required to be a *ground instance* of an element of a set of predefined candidate explanations called *abducibles*.[1]

---

[1] For an introduction to abduction see, for example, [Peirce, 1931 1958] or [Fann, 1970]. An overview can be found

However, this "classical" abduction principle is insufficient for temporal hypotheses incorporating an implicit representation of time, because the correctness criterion is no longer satisfied (cf. [Paul, 1993b]). To overcome this deficiency the abductive process is divided into two phases: 1. A *guessing phase* where modal hypotheses are adopted—these abducibles are the abstract plans provided by the plan generation component—and 2. a *validation phase* where the hypotheses are verified with respect to the sequence of observed actions. This is the task of plan recognition.

In addition, the ground instance requirement has to be adapted to our modal logic context. It is generalized in such a way that hypotheses are made more precise not only by instantiating variables but in a temporal sense as well. We *concretize* the hypotheses according to the observations made in each recognition step.

This process of concretizing will be explained using a short example with two observation steps. It is assumed we are given the following plan hypothesis synthesized by the generation component as described in section 3.

$$P1 = EX(header(M)); EX(type(x, M));$$
$$EX(delete(x, M))$$

Suppose we have the additional hypotheses

$$P2 = \Diamond EX(header(P)); EX(type(x, P));$$
$$EX(delete(x, P))$$
$$P3 = \Diamond EX(header(P)); EX(next(x, P));$$
$$EX(delete(x, P))$$

i.e., contrary to the first hypothesis the *header* command is not expected to be the first action but may also occur at some later stage. In addition, a mailbox named P is supposed to be the current one here.

Observing the user execute the action $EX(folder(P, M))$, i.e., he moves from the mailbox M to the mailbox P, leads to the rejection of $P1$ and the following concretized hypotheses:

$$P2^1 = EX(folder(P, M)) \wedge \bigcirc \Diamond EX(header(P));$$
$$EX(type(x, P)); EX(delete(x, P))$$
$$P3^1 = EX(folder(P, M)) \wedge \bigcirc \Diamond (EX(header(P));$$
$$EX(next(x, P)); EX(delete(x, P))$$

Concerning $P1$ we definitely anticipated $EX(header(M))$. This cannot be concretized with the first observation and thus the hypothesis has to be rejected. In $P2$ and $P3$ the initially expected user action is not specified, we only know that the given action sequence is executed at some time. So, after observing an action not matching the first expectation, we know that this sequence cannot start before the next observation, i.e., we get as parts of the new hypotheses $P2^1$ and $P3^1$ the formula $\bigcirc \Diamond EX(header(P))$, stating that the action *header* is anticipated at some future time and not before the next state.

If the second observation is $EX(header(P))$, the concretization of both $P2$ and $P3$ is a disjunction of two hypotheses. Either we recognize the *header* command in the "sometimes" sequence, or we make a decision to expect it later. Thus, we have

in [Kakas *et al.*, 1992] and [Paul, 1993a].

$$P2_1^2 = EX(folder(P, M)) \wedge \bigcirc EX(header(P));$$
$$EX(type(x, P)); EX(delete(x, P))$$
$$P2_2^2 = EX(folder(P, M)) \wedge \bigcirc EX(header(P)) \wedge$$
$$\bigcirc \bigcirc \Diamond EX(header(P)); EX(type(x, P));$$
$$EX(delete(x, P))$$

As the hypotheses for $P3$ are similar we will omit them here.

A hypothesis is said to be *recognized* if the sequence of observations implies its concretization. Considering the example, this is the case for $P2$ if, e.g., the next observations are $EX(type(2, P))$ and $EX(delete(2, P))$.

Besides the theoretical foundations, a method was developed for computing concretized explanations. An algorithm solving this problem must be able to identify at each state of time the part of the considered hypotheses being affected at that moment. To do this in an efficient way, we use a *transformation* of LLP formulas into graphs that contain solely first-order formulas.[2] This transformation is carried out once before starting recognition. The plan recognition process is realized by moving through the graph according to the observations made. A theorem prover is used to deduce relations between abstract commands in the hypothesis and observed actions. Several properties of this generalized abduction principle have been proven. For details see [Paul, 1993b].

The method described above allows us to recognize temporal abstractions as well as abstract plans. We are able to retransform the graph in each iteration step, thus obtaining a history of the actions observed and a description of the expected continuation of the plan. By that we are able to offer semantic plan completion at any time as long as valid hypotheses are available. Probabilistic selection (cf. section 4.2) is the method used in determining the "best" hypothesis for that purpose.

4.2 Probabilistic Selection

The plan recognizer described so far manipulates *sets* of plan hypotheses each of which is considered equally plausible. If, however, a decision for *one* alternative is required (e.g., to offer semantic plan completion to the user at an early stage), we must be able to determine the "best" choice among them. To do so, we can exploit knowledge about user preferences stored in a kind of user model and encoded in a probabilistic mechanism.

Probabilistic reasoners (e.g., [Kruse *et al.*, 1991]) use a knowledge base as a back-up, which contains the set of all possible hypotheses together with numerical values assigned to them representing their specific probability. These numbers usually stem from long-term observations of the domain and from statistics. The situation in the first cross-talk mode, however, is somewhat different because the search space of the plan recognizer—the set of all plan hypotheses—is generated dynamically. So, we cannot expect to have any statistical information at hand that directly applies to it. Yet, we can evaluate the

---

[2]Graph- or so-called tableau-based methods are used for a variety of modal logics. See, for example, [Pratt, 1979], [Ben-Ari *et al.*, 1982], and [Wolper, 1985].

user behavior observed according to criteria like typical action sequences, frequently pursued goals, etc.

If we know, for example, that the user tends to delete a message immediately after reading it, and that he usually prefers the *type* command to *next*, we might come up with a set C of LLP formulas similar to abstract plans or plan specifications (cf. section 2) describing the user's default behavior. Each formula is assigned a numerical value representing the statistical information about it. In the example above, C might include the following entries

$C_1 : \langle \exists x, mb.\Diamond(EX(type(x, mb)); EX(delete(x, mb))),$
        $0.4 \rangle$

$C_2 : \langle \exists x, mb.\Diamond(EX(next(x, mb)); EX(delete(x, mb))),$
        $0.2 \rangle$

$C_3 : \langle \exists x, mb.\Diamond(EX(read\_mail(x, mb));$
        $EX(delete(x, mb))), 0.1 \rangle$ ...

The entry C_3 here means that for a small part of all observed cases, we know only that the user executed a *type* or a *next* command to read his messages, expressed by using the abstract command *read_mail*. The formula in C_3 is more general than those of C_1 and C_2. Such relations induce a hierarchical structure on C that is exploited during the numerical computations.

The numerical values distributed among the members of C sum up to 1 and form a *mass distribution* from Dempster-Shafer theory (DST) (cf. [Shafer, 1976], [Shafer and Pearl, 1990]). While interpreting the value 1 as perfect certainty, smaller numbers represent the degree of partial confidence we might have in the validity of the various propositions.

From a mass distribution m, we can derive the so-called *belief* and *plausibility* functions Bel_m and Pl_m, respectively. These two values make up a probability interval stating that the "true" probability of some proposition A lies somewhere between $Bel_m(A)$ and $Pl_m(A)$, but cannot be determined on the basis of the knowledge at hand. Thus, we are able to express *partial ignorance*.[3]

Classifying the plan hypotheses obtained from the plan generator according to the formulas C_i in C, a set of valuated hypotheses sets PH_{C_i} is obtained each of which inherits the numerical value originally attributed to its classification criterion in C. In our example, P_1 and P_2 become members of the class PH_{C_1}, P_3 becomes an element of PH_{C_2}. In addition, they are all placed in class PH_{C_3}. Thus the original hierarchy of classification criteria according to generality mentioned above mirrors itself in the subset/superset relation of the associated hypotheses sets that make up the *plan hierarchy PH*.

After this preprocessing, the probabilistic selection module in every recognition step obtains the most recent observation together with information about those hypotheses which are no longer valid. On the basis of this knowledge, we can compute an updated mass distribution on PH reflecting the impact of the new observation on the a-priori valuation of the hypotheses. Dempster's rule is the basis for this computation which is explained in more detail in [Bauer, 1993].

---

[3]For an examination of the relations between mass distributions, probabilities, and DST see [Kruse *et al.*, 1991].

Let us assume we are given the information that P_1 is no longer valid. Then, the observation $EX(header(Priv))$, for example, may lead to a new mass distribution where $PH_{C_1} = \{P_2\}$ and $PH_{C_2} = \{P_3\}$ are attributed the values 0.5 and 0.35, respectively, and $PH_{C_3} = \{P_2, P_3\}$ is valuated with 0.15. If we recall from DST that the *belief* in a set A of hypotheses is computed by summing up the respective mass values attributed to all of its subsets and the *plausibility* is the result of adding the mass values of all sets having a non-empty intersection with A, we can interpret these numbers as follows: Currently, hypothesis P_2 appears to be the most likely because its probability lies somewhere in the interval $[0.5, 1]$. If semantic plan completion is required, P_2 is the current offer of the system to the user. If we are forced to additionally provide a hypothesis *definitely* containing the plan pursued by the user, we can choose the disjunction of P_2 and P_3 because the set PH_{C_3} which contains these only is the smallest set with attributed belief 1, i.e., it represents the most specific hypothesis we *certainly* believe in. This property uniquely determines the disjunction of P_2 and P_3 because all other plan hypotheses that might be contained in PH are attributed mass 0 and thus are considered impossible according to the evidences obtained so far.

5 Conclusion

A new approach in implementing intelligent help systems has been introduced. It is based on a logic developed especially for the command language environments in which help systems are embedded. Plan recognition and plan generation components are special purpose inference processes. They work in close mutual cooperation (cross-talks). One cross-talk mode is devoted to plan recognition on the basis of abstract plans provided by the planner; another mode works on generating optimal plans on the basis of recognition results. Planning from first as well as from second principles is done deductively combining ideas borrowed from the logic-based treatment of programs with those of tactical theorem proving. The resulting plans are provably sound w.r.t. their specifications. Plan recognition is based on a new abductive principle for modal logics. The recognizer is additionally supplied with a probabilistic reasoner, a means to improve the help provided by taking into account user-specific characteristics as well as general heuristics.

Realizing plan generation and recognition in such a strictly logic-based way nevertheless does not cause any considerable inefficiencies, even for "real" *mail* plans that are much more complex than the examples shown in this paper. This suggests evaluating the PHI approach even for richer application domains. A prototype of the PHI system has been implemented in SICSTUS PROLOG on a SUN SPARC computer.

References

[Appelt and Pollack, 1990] D.E. Appelt and M. Pollack. Weighted abduction for plan ascription. Technical report, AI Center, SRI International, Menlo Park, 1990.

[Bauer, 1993] M. Bauer. Plan recognition under uncertainty. Research Report, DFKI, 1993. to appear.

[Ben-Ari et al., 1982] M. Ben-Ari, J.Y. Halpern, and A. Pnueli. Deterministic propositional dynamic logic: finite models, complexity, and completeness. *Comput. System Sci.*, 25:402–417, 1982.

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

[Biundo and Dengler, 1993] S. Biundo and D. Dengler. The logical language for planning LLP. Research Report, DFKI, 1993. to appear.

[Biundo et al., 1992] S. Biundo, D. Dengler, and J. Koehler. Deductive planning and plan reuse in a command language environment. In *Proc. of 10th ECAI*, pages 628–632, 1992.

[Biundo, 1992] S. Biundo. *Automatische Synthese rekursiver Programme als Beweisverfahren*. Springer IFB 302, Berlin, Heidelberg, New York, 1992.

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

[Constable, 1986] R.L. Constable. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, 1986.

[Fann, 1970] K.T. Fann. *Peirce's Theory of Abduction*. Martinus Nijhoff, The Hague, 1970.

[Fikes and Nilsson, 1971] R.E. Fikes and N.J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *AI*, 2:189–208, 1971.

[Green, 1969] C. Green. Application of theorem proving to problem solving. In *Proc. of 2th IJCAI*, pages 219–239, 1969.

[Hammond, 1990] K. J. Hammond. Explaining and repairing plans that fail. *AI*, 45:173 – 228, 1990.

[Hanks and Weld, 1992] S. Hanks and D. S. Weld. Systematic adaptation for case-based planning. In *Proc. of 1st AIPS*, pages 96–105, Washington, D.C., 1992. Morgan Kaufmann.

[Heisel et al., 1990] M. Heisel, W. Reif, and W. Stephan. Tactical theorem proving in program verification. In *Proc. of 10th CADE*. Springer LNCS 449, 1990.

[Heisel et al., 1991] M. Heisel, W. Reif, and W. Stephan. Formal software development in the KIV system. In *Automating Software Design, R. McCartney and M.R. Lowry (eds.)*. AAAI Press, 1991.

[Helft and Konolige, 1990] N. Helft and K. Konolige. Plan recognition as abduction and relevance. Draft version, AI Center, SRI International, Menlo Park, 1990.

[Kakas et al., 1992] A.C. Kakas, R.A. Kowalski, and F. Toni. Abductive logic programming. Draft version, Department of Computer Science, University of Cyprus, Nicosia, and Imperial College of Science, Technology and Medicine, London, 1992.

[Kambhampati and Hendler, 1992] S. Kambhampati and J. A. Hendler. A validation-structure-based theory of plan modification and reuse. *AI*, 55:193 – 258, 1992.

[Koehler, 1992] J. Koehler. Towards a logical treatment of plan reuse. In *Proc. of 1st AIPS*, pages 285–286, Washington, D.C., 1992. Morgan Kaufmann.

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

[Kruse et al., 1991] R. Kruse, E. Schwecke, and J. Heinsohn. *Uncertainty and Vagueness in Knowledge Based Systems*. Springer, 1991.

[Manna and Waldinger, 1980] Z. Manna and R. Waldinger. A deductive approach to program synthesis. *ACM Transactions on Programming Languages and Systems*, 2:90–121, 1980.

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

[Nilsson, 1980] N.J. Nilsson. *Principles of Artificial Intelligence*. Springer, New York, 1980.

[Norvig et al., 1993] P. Norvig, W. Wahlster, and R. Wilensky. *Intelligent Help Systems for UNIX - Case Studies in Artificial Intelligence*. Springer, Heidelberg, 1993. to appear.

[Paul, 1993a] G. Paul. Approaches to abductive reasoning – an overview. *AI Review*, 1993. to appear.

[Paul, 1993b] G. Paul. A generalized abductive principle for a modal temporal logic. Research Report, DFKI, 1993. to appear.

[Paulson, 1990] L. Paulson. Isabelle: The next 700 theorem provers. In P. Odifredi, editor, *Logic and Computer Science*. Academic Press, 1990.

[Peirce, 1931 1958] C.S. Peirce. *Collected Papers of Charles Sanders Peirce (eds. C. Hartshorne et al.)*. Harvard University Press, 1931-1958.

[Pratt, 1979] V. Pratt. Models of program logics. In *Proc. of 20th Annual IEEE Symposium on Foundations of Computer Science*, pages 115–122, 1979.

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

[Shafer and Pearl, 1990] G. Shafer and J. Pearl, editors. *Readings in Uncertain Reasoning*. Morgan Kaufmann Publishers, Los Altos, 1990.

[Shafer, 1976] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, 1976.

[Shanahan, 1989] M. Shanahan. Prediction is deduction but explanation is abduction. In *Proc. of 11th IJCAI*, pages 1055–1060, 1989.

[Thies and Berger, 1992] M.A. Thies and F. Berger. Plan-based graphical help in object-oriented user interfaces. In *Proc. of the International Workshop on 'Advanced Visual Interfaces'*, Rome, Italy, 1992.

[Wærn, 1992] A. Wærn. Reactive abduction. In *Proc. of 10th ECAI*, pages 159–163, 1992.

[Wolper, 1985] P. Wolper. The tableau method for temporal logic: an overview. *Logique et Anal.*, 28:119–136, 1985.