A Semi-Reactive Planner Based on a Possible Models Action Formalization

Sylvie Thiébaux*
Computer Science Department
FIT
Melbourne, FL 32901-6988, U.S.A.
e-mail: sylvie@cs.fit.edu

Joachim Hertzberg
AI Research Division
GMD
5205 Sankt Augustin 1, F.R.G.
e-mail: hertz@gmdzi.gmd.de

Abstract

We describe the basics of a semi-reactive planner that is based on a possible models action formalization. By semi-reactive planner, we mean a planning system that is able to incrementally generate plans in a classical way, thereby extending, when necessary, the compiled knowledge available for reactive execution. The action formalization, which is a correction of the possible worlds approach by Ginsberg and Smith, allows one to cope with lacking information, ambiguity and context dependency in the planning domain by considering different courses of action. This enables the planner to handle certain forms of uncertainty at planning time. Plans developed under this model are compiled into finite state automata for execution, as a basis for reacting to less expected effects at execution time. Moreover, we explain how replanning fits into the framework, thereby opening it towards handling unexpected effects which occur during plan execution, and for which no reaction plan is available.

1 MOTIVATION

Much of the more recent work in AI planning can be subsumed under the following two topics, both of which are in fact old, dating back to research on STRIPS [Fikes and Nilsson, 1971; Fikes *et al.*, 1972] and before.

Formalization, i.e., formal calculi for planning, for describing planning systems, or planning using formal calculi for more or less complete parts of a planner. (See [Bibel, 1986], [Lifschitz, 1987], [Reichgelt and Shadbolt, 1990; Chapman, 1987] for examples of the respective directions.) The idea is that formalization helps to understand or to communicate one's understanding of planning in detail; more practically, descriptions of some degree of formality might be a basis for sound software engineering. Moreover, there is the related work on formalizing action and change, which, however, is not always put into relation to planning as closely as possible. (An example for such work is [Ginsberg and Smith, 1988], which will be talked of later; see [Sandewall, 1991] for further references.)

Reactivity, i.e., reactive planning and reaction planning. ([Schoppers, 1989a], from which we borrow the terminology, contains further references.) The idea is the following. In general, it is wrong to assume that every conceivable course of things can be properly kept track of at planning time and that all and only the events considered at planning time will happen at plan execution time. Reaction planning tackles the problem by (sometimes automatically) encoding goal directed behavior not only for a single planning problem but for whole parts of the problem domain (e.g., [Georgeff and Lansky, 1987; Agre, 1988; Schoppers, 1989b]). Reactive planning follows the idea of providing rough plans that are used, reacting to unforeseen or unforeseeable events in the world at execution time (e.g., [Dean and Boddy, 1988]).

If you agree with the reasons for working on the individual topics, then it seems obvious that their combination is worth being examined; work in this direction is, e.g., [Downs and Reichgelt, 1991]. The objective of the work reported in our paper is to develop a planner exhibiting some degree of reactivity, and having a well defined semantics, not by an *ex post* formalization, but by virtue of being based on a well understood action calculus. You will even notice that our formalization resembles Winslett's [1988] and that the basics of our semi-reactive system is similar to, e.g., Drummond and Bresina's [1990] projection graphs and Kabanza's [1990] world automata. (The differences are sketched in the sections 2 and 3, respectively). However, the point here is not to develop yet another variant of well-known action formalizations or reactive systems, thereby pushing forward reaction planning or planner formalization individually, but to attempt exploring

*Work done while the author was at GMD

what Drummond and Bresina [1990] call the

> continuum of plan-guided systems from those ..., where complete plans are produced in advance and later used by independently competent execution systems, to those systems that are embedded in the situations for which their plans are generated.

In particular, we propose to build a semi-reactive planning system, i.e., a system that is able to generate reaction plans to extend its "compiled" knowledge, when no reaction plan is available. Our planner is based on a particular formalization of actions [Brewka and Hertzberg, 1991] that allows taking several forms of uncertainty into account at planning time. In accordance with a theory about the world stating some of the most frequent unwanted effects occuring when applying an action, it develops plans that include alternative actions. As it is unthinkable to envisage *all* possible cases in advance, the remaining part of uncertainty will be handled at execution time, by using the reaction plans available, or by generating new ones incrementally. The reaction plans are represented as classical finite automata.

The computer program PASCALE illustrates and implements the ideas presented here; [Thiébaux, 1991] describes it and elaborates on these ideas.

The paper is organized as follows. In section 2, we present the possible models formalization as the formal background for our approach. Section 3 describes the plan representation we use. In section 4, we describe the concept of a semi-reactive system built within the framework. Section 5 concludes with some notes about related work.

2 A POSSIBLE MODELS FORMALIZATION OF ACTION

To start with, we briefly present the key concepts of the formalization of actions our work is based on. The formalization is described in [Brewka and Hertzberg, 1991] to which we refer for details, a discussion of its strenghts and weaknesses, and a comparison to related work.[1] We first give the intuition of the approach informally, turning to formal definitions afterwards.

[1]In [Brewka and Hertzberg, 1991], what we call possible *model* here is called possible *world,* following the terminology of Ginsberg and Smith. To stress the difference to Ginsberg and Smith's approach and the similarity with Winslett's, we change the terminology. Moreover, in view of implementation, we make a few very slight modifications in the formalism described by Brewka and Hertzberg.

2.1 INTUITION

Starting point for our action model is the state based approach of Ginsberg and Smith [1988]. Their idea is to use standard first order logic to model action effects, where the frame problem is solved by making the STRIPS assumption explicit: Given a state, which is described by a set S of formulae (denoting their conjunction), and an action whose postconditions P are also given as a set of formulae, a possible world T for P in S is a maximal consistent subset of $S \cup P$ containing all the formulae in P. That means: Compute $S \cup P$, and then delete non-elements of P from the whole set until the rest is consistent. As there may be different ways to make $S \cup P$ consistent, T is not unique.

The appeal of this formalization is that it looks close to what may be used in a planner implementation. However, it may yield highly unintuitive results, as was first noticed by Winslett [1988] and described in more detail in [Brewka and Hertzberg, 1991]. Ginsberg and Smith's approach tends to have problems with obtaining the intended results of actions in the presence of lacking or ambiguous information about the situation or about action postconditions.

Consider an example. In a room, there are a door and a window, at least one of which is currently open, i.e., $open(d) \vee open(w)$. You are now told that someone has—by an action that may or may not have changed something—cared for the door being closed. Intuitively, only $\neg open(d)$ should be concluded now, as nothing is known about the window. According to Ginsberg and Smith's approach, however, closing the door would magically open the window: as $open(d) \vee open(w)$ and $\neg open(d)$ are consistent, the new information would just be added to the situation description, and $open(w)$ becomes derivable.

As argued in [Winslett, 1988], the remedy for this problem is to calculate successor state descriptions not in terms of sets of arbitrary formulae but in terms of ground instances of literals occurring in the theory. (These can be understood as partial Herbrand models of the theory, hence the term possible *models.*) The idea is that instead of the description S of the recent world state you take the set of possible models of S, which may be unary or not, and apply the action in each of these possible models. Moreover, action effects must be represented as subsets of possible models (partial possible models, for short), not as arbitrary formulae. This is a difference to Winslett's approach, which is necessary to handle alternative action effects properly. Again, see [Brewka and Hertzberg, 1991] for examples and a discussion of this point.

An ambiguous outcome of an action is captured as a set of alternative outcomes, formulated as different partial possible models. The idea of calculating successor states by considering maximally consistent sets of formulae containing the just obtained action effects is imported from Ginsberg and Smith, but is applied only to possible models, not to arbitrary formulae.

In the example of closing the door, the description of the initial situation yields three possible models, involving essentially: either the door is open, or the window is open, or both of them are open. After closing the door, we have two possible worlds, involving essentially: either the door is closed and the window is open, or the door is closed and the window is closed.

We will now describe the basics of the approach formally.

2.2 FORMAL DEFINITION

We assume a first order logical language $\mathcal{L}$ of closed formulae. A subset of $\mathcal{L}$ is the finite set $L = \{s_1, \ldots, s_n\}$ of ground atomic sentences. L must be sufficient to describe all relevant aspects of different states of the world as a conjunction of ground literals l_i built from these atomic sentences, i.e., $l_i = s_i$ or $l_i = \neg s_i$. Each n-elementary conjunction $l_1 \wedge \cdots \wedge l_n$ of literals from all elements of L is called an *L-model.*[2] If this causes no confusion, we will switch between the conjunctive notation for L-models or partial L-models as defined and set notation, e.g., $\{l_1, \ldots, l_n\}$, which is to be interpreted as the conjunction of its elements.

We assume a set $K \subseteq \mathcal{L}$ representing the domain *background knowledge,* i.e., knowledge about what is true in all situations. Assuming furthermore that a situation is described as a formula of $\mathcal{L}$, we now define

Def. 2.1 (Possible model) *Let $S \in \mathcal{L}$ be a formula and K the background knowledge. The possible models in S with respect to K are*
$Poss_K(S) = \{pm | pm$ *is an L-model and* $K \cup \{S\} \cup pm$ *is satisfiable.*$\}$

We use $Poss(K)$ as a shorthand for the possible models of the conjuction of formulae in K.

Consider again the door and window world. Let $L_0 = \{open(d), open(w)\}$; the background knowledge $K_0 = \{open(d) \oplus closed(d), open(w) \oplus closed(w)\}$;[3] $\mathcal{L}_0$ is defined appropriately. Let now $S_0 = open(d)$, i.e., all you know is that the door is open. Then

$$\begin{array}{rcl} Poss_{K_0}(S_0) & = & \{\ \{open(d), open(w)\}, \\ & & \ \ \{open(d), \neg open(w)\}\ \} \end{array}$$

The respective complete situation descriptions are obtained using K_0, enforcing, e.g., the conclusion $\neg closed(d)$ in both possible models.

[2]This definition clearly overloads the term model which keeps its usual logical definition. The excuse for this overloading is that the concept is very close to the usual notion of Herbrand model and that [Winslett, 1988] does the same. Moreover, there will be no confusion of our models with the usual notion of model as we will primarily use *possible* models, which will be defined next.

[3]$\oplus$ is the exclusive disjunction.

Our actions are generalized variants of STRIPS operators with pre and postconditions, without distinguishing between added and deleted postconditions. We allow for alternative outcomes of actions applied in the same situation; an example is the action of tossing a coin. Independently from that, an action may produce different effects when applied in different situations; consider as an example the action of toggling a lightswitch: the light is on if it was off before, and vice versa. The general format of actions, then, is:

$$\begin{array}{lcll} [\ \ Pre_1 & | & Post_{1,1}, \ldots, Post_{1,l(1)}; & \\ \quad\vdots & & & \\ \ \ Pre_m & | & Post_{m,1}, \ldots, Post_{m,l(m)}; & \\ \ \ true & | & Post_{\text{true}} &] \end{array}$$

The preconditions Pre_i are arbitrary formulae from $\mathcal{L}$; each postcondition element $Post_{i,j}$ is a set (conjunction) of literals of ground atoms from L. The preconditions are tested in the order they appear. The first precondition that holds determines the possible effects of the action. By default, $Post_{\text{true}}$ is the empty set. This captures one of the potential intuitions about what may result from—well—applying an action in a situation where it is not applicable: The action changes nothing. Different intuitions are possible, but this is not an issue here. As a shorthand, the *true* case is skipped, i.e., $[Pre|Post]$ stands for $[Pre|Post; true|\{\}]$. In this action format, nothing is said about the relative probabilities with which the different postconditions in one branch will occur. This might be done along the lines of [Hanks, 1990], but we will not do so here.

As an example, consider the action *nocirculation* of preventing circulation in the door and window world: If the door and window are both open, one of them will be arbitrarily shut, otherwise nothing is done.

$$\begin{array}{l} nocirculation = \\ [\ open(d) \wedge open(w) \ \ | \ \ \{\neg open(d)\}, \{\neg open(w)\}\ ] \end{array}$$

Note that there is no need to specify that something is closed when it is not open any more: this will follow from the background knowledge. Thus, we provide a solution to both frame and ramification problems. See [Brewka and Hertzberg, 1991] for details about when this solution is sound.

We will now define the result of an action; before doing so, however, we need the intermediate concept of p-conformity for a set of literals p made from L. This is needed to capture the intuition that a possible model, that results from applying an action making p true, contains p but differs as little as possible from the possible model representing the original situation.

Def. 2.2 (Maximal p-conformity)
Let pm and pm' be elements of $Poss(K)$, and p a partial model. pm' is maximally p-conform with pm iff $pm \cap pm'$ is (set inclusion) maximal, subject to the condition that $p \subseteq pm'$.

For example, the possible model $\{open(d), \neg open(w)\}$ is maximally $\{\neg open(w)\}$-conform with the possible model $\{open(d), open(w)\}$, whereas, though being an element of $Poss(K)$, $\{\neg open(d), \neg open(w)\}$ is not.

Def. 2.3 (Result of an action)
Let S be a formula, and let a be an action such that

$$a = [\; Pre_1 \mid Post_{1,1}, \ldots, Post_{1,l(1)}; \; \vdots \; Pre_m \mid Post_{m,1}, \ldots, Post_{m,l(m)} \;]$$

The result of applying a in S given K, denoted $r_K(a, S)$, is the set:
$\{pm' \in Poss(K) | \exists pm \in Poss_K(S)$ such that pm' is maximally $Post_{i,j}$-conform with pm, where i is the smallest integer such that $K \cup pm \models Pre_i$ and $j \in \{1, \ldots, l(i)\}\}$.

As an example, let us apply the action *nocirculation* to the incomplete situation S_0 and the K_0 given previously. We have:

$$r_{K_0}(nocirculation, S_0) = \{ \{open(d), \neg open(w)\}, \{\neg open(d), open(w)\} \}$$

In the first possible model of $Poss_{K_0}(S_0)$, the door and window are both open. Thus, the action can produce one of the two possible models, depending on whether the door or else the window is closed. For the second possible model of $Poss_{K_0}(S_0)$, there is already no circulation. Here, the action changes nothing, and the model equals the one resulting from closing the window.

Naturally enough, the result of a linear plan, i.e., a total ordering of actions, is the set of possible models resulting from the successive application of the actions in the plan. The result of a nonlinear plan, i.e., a partial ordering of actions, then, is the union of possible worlds resulting from all linearizations.

This suffices as a description of the possible models action formalization; consult [Brewka and Hertzberg, 1991] for further information. We showed that this formalization copes with certain forms of uncertainty at planning time: lacking information (disjunctions in or incomplete specification of the start situation), ambiguity (alternative operator effects), and context dependency of operator effects. That is quite powerful; but it is not enough. The possible models give no help for dealing with uncertainty at execution time that would require to react flexibly to failure, sabotage, or serendipity which are impossible to think of at planning time by definition. Developing a framework additionally facilitating this is what we will do now.

3 REPRESENTING ALTERNATIVE PLANS AS FINITE AUTOMATA

We start with the question how to *represent* plans, which are in accordance with the possible models formalization, such that they can be reused flexibly and guide their execution monitoring. We will then briefly describe how to *generate* these plans.

As for plan representation, the idea is to define an analogue to STRIPS triangle tables [Fikes *et al.*, 1972] that supports handling alternative action effects as defined in the possible models framework. The key observation concerning the representation is that possible models are finite by definition. So the task is to represent a structure of finitely many actions, applicable in a subset of finitely many possible models, where each action application produces another subset of the possible models; there is a set of start and a set of finish models.

It is easy to see that such a structure can be understood as a finite automaton (FA): actions correspond to states of the automaton, the possible worlds to the finite alphabet, a dummy start action resulting in the initial possible models corresponds to the initial state, another dummy action with the goal description as precondition to the final state of the automaton, and the results of the actions define the transition function. The resulting FA is even deterministic.

The plan representation we actually use is a variant of a FA, where it is mostly technical to show the correspondence, see [Thiébaux, 1991]. The idea is first to define a graph type, called *T-M* graph, mirroring the interplay between tasks, i.e., occurrences of actions in a plan, and possible models; and then to represent individual planning problems given by a formula S defining the start situation, the background knowledge K and a set of actions A, as a special *T-M* graph.

Def. 3.1 (*T-M* Graph)
A T-M graph is a bipartite directed graph consisting of T- and M-nodes with a unique root, where:

- *The root is a T-node; all leaves are M-nodes.*
- *T-nodes have only M-nodes as successors.*
- *Non-leaf M-nodes have exactly one T-node as successor; all non-root T-nodes have exactly one predecessor.*

Def. 3.2 ((S, K, A) Graph)
Given two formulae S and K, a set of actions A, and Start, a task built from the action that is defined by:

$$start = [\; true \mid pm_1, \ldots, pm_n \text{ such that } \{pm_1, \ldots, pm_n\} = Poss_K(S) \;]$$

An (S, K, A) graph is a T-M graph consisting of exactly the following nodes:

- *M-nodes are possible models from $Poss(K)$.*
- *T-nodes are Start and tasks built from actions in A. Start occurs exactly once in the graph.*
- *Start is the root; all $Poss_K(S)$ elements are its sucessors, and only these.*
- *Let m be a non-leaf M-node and t its successor T-node. The successors of t are the M-nodes corresponding to all the elements of $r_K(t, m)$ (extending the r_K function to T- and M-nodes in the obvious way).*

The last item is the really interesting one, guaranteeing that all and only the results of tasks are included in the graph. To use an (S, K, A) graph as a plan, you must make sure that its leaves correspond to goal states and that there is at least one path to a leaf from each of its nodes. More formally:

Def. 3.3 (g-Achievingness, Validity, Plan)
Given a formula g, an (S, K, A) graph achieves g if and only if for all of its leaves m: K together with the possible model corresponding to m entail g.
An (S, K, A) graph is valid if and only if from each of its nodes there is at least one path to a leaf.
Given a planning problem specified by a formula S representing the initial state, a background knowledge K, a set of actions A, and a formula g representing the goal, a plan solving this problem is a valid g-achieving (S, K, A) graph.

As an informal lemma, note that a valid graph cannot include a T-node corresponding to a task that is not applicable under the possible model corresponding to its predecessor M-node, i.e., it changes nothing. The reason is that the loop formed by these two T and M-nodes would be a blind alley in the graph.

Consider the following example. Starting a lecture in a room in a building on a noisy street requires that the window be closed. Sometimes, a janitor would open it between lectures, but not always, so nothing is known initially about the window being open or closed. Moreover, you must turn on the light, which the janitor would switch off between lectures; the light is operated by two toggle switches *sw1* and *sw2* both of which can be in positions $swi, \neg swi$. The light is *on* if and only if *sw1* and *sw2* are in equal positions. Both switches are old and unreliable: immediately after flipping them, they may flip back into their former position, i.e., toggling a switch may or may not have an effect.

The language L_1 for describing possible models consists of *open*, *sw1*, and *sw2*; $\neg open$ is, outside of possible models, called *closed*, and $\neg on$ is called *off*. The available action set A_1 consists of *toggle1* for toggling *sw1* and *close-window* for closing the window if open.

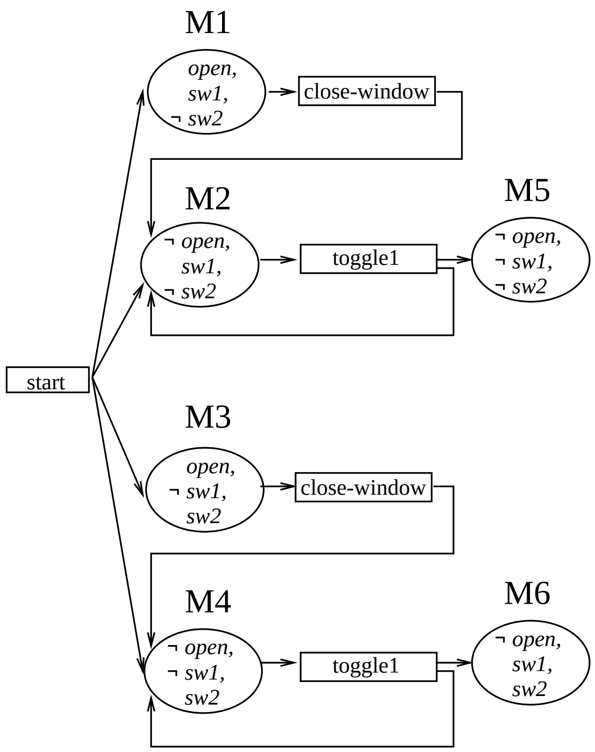

Figure 1: An (S, K, A) graph representing an alternative plan for the lecture example.

In summary, we have

$$\begin{array}{rclcl} \textit{close-window} & = & [\ \textit{open} & \mid & \{\neg \textit{open}\} \quad] \\ \textit{toggle1} & = & [\ \textit{sw1} & \mid & \{\neg \textit{sw1}\}, \{\}; \\ & & \neg \textit{sw1} & \mid & \{\textit{sw1}\}, \{\} \quad] \end{array}$$

$$\begin{array}{l} K_1 = \{\textit{open} \oplus \textit{closed},\ \textit{on} \leftrightarrow (\textit{sw1} \leftrightarrow \textit{sw2}),\ \textit{on} \oplus \textit{off}\}, \\ S_1 = \{\textit{off}\}, \\ g_1 = \textit{closed} \wedge \textit{on} \end{array}$$

A plan for solving the problem is shown in figure 1. This plan is to be interpreted as follows: Finding yourself in one of the possible start situations where the window is closed, toggle *sw1* until it works; else close the window first and continue as before. So, this plan, as one would intuitively expect from something *planned,* does *not* describe all possible courses of action but follows a rationale: close the window first and switch on the light then. Of course, it would have been possible to find other plans.

Note that M nodes occur only once in the graph, even if generated by different T nodes. This keeps the graph finite, because there are only finitely many possible models. Note further that the graph looks a bit like the representation of a part of the state space, where not all applicable tasks are included. However, it is different from a state space representation in that the representation of a state is restricted to the vocabulary L for describing the possible models. In fact, the graph is a finite part of the state space where states are restricted to *essential* [Lifschitz, 1987] ground atoms or to *frame fluents* [Lifschitz, 1990].

Our (S, K, A) graphs are comparable to Kabanza's [1990] branching time temporal logics based *world automata* in that we use actions (events) with alternative effects. (S, K, A) graphs are different in that they have a bipartite node set. The reason behind this difference is a difference in ontology: what we would view as alternative outcomes of an action applied in a situation, would be viewed as a single action (event) followed by different external events by Kabanza. An advantage of his approach is that world automata describing "purely external" courses of events and world automata involving agent induced events, i.e., plans, have an identical structure and can thus be handled identically. The advantages of our graphs will become apparent when we talk in section 4 about replanning and plan reuse, which seem to be a little cumbersome in Kabanza's approach.

The requirement that non-leaf M-nodes have exactly one successor T-node in T-M graphs makes the execution of a plan deterministic. That has its advantages. On the other hand, it enforces that only the analogs of linear plans can be represented here; e.g., if either of the two execution sequences of two commutatively executable actions would lead to the goal, then the very format of our plans forces the planner to commit itself to one of these sequences. Experimenting with ways to introduce non-linearity into the plans and investigating the implications is part of our future research.

How can these plans be generated? As we have not yet defined an appropriate version of regression [Waldinger, 1977] for our action format, we presently generate the plans in forward fashion in PASCALE. Moreover, we haven't worried about clever control heuristics, so one should not have high expectations about PASCALE's perfomance. In fact, PASCALE performs A^* search through the space of partial valid (S, K, A) graphs. Such a search always terminates as the search space is finite, which is inherited from the finiteness of the state space. Moreover, restricting the search to *valid* graphs reduces the search space considerably.

The approach allows for an *anytime* planning behavior [Dean and Boddy, 1988], if you introduce a possibility ordering of action postconditions, and plan for the most probable outcomes first. This is in fact done in [Drummond and Bresina, 1990]. Introducing probability into (S, K, A) graphs is part of our future research.

Finally, it is unnecessary to work on the whole space of possible models at once. An agreeable feature of g-achieving (S, K, A) graphs is that they can be generated incrementally and reused later, which is a topic for the next section.

4 BUILDING A SEMI-REACTIVE SYSTEM

It is commonplace that not all possible events can be accounted for at planning time. This remains true when using (S, K, A) graphs: you can plan for anticipated execution failures such as sticking toggles, or for lacking information such as the light's initial state. But the plans don't deal with every conceivable course of events. This is no bug, but a matter of efficiency.

Hence, the point here is *not* that executing (S, K, A) graphs doesn't require the ability to recover from execution failures and to replan. The point *is* that these plans are already close to what is needed for execution monitoring and are a good basis both for monitoring and for replanning in the case of execution failures. Thus, facilitated by the possible models approach, planning, execution, and plan reuse are intimately intertwined, motivating the term of *semi-reactive* planning. All this shall be made plausible now, starting with execution monitoring and turning to replanning and plan reuse afterwards. All this will be described informally; consult [Thiébaux, 1991] for a more formal treatment.

4.1 EXECUTION MONITORING

To see what the problems are, let us modify our lecture preliminaries example. Consider that while fumbling with *sw1*, the window gets reopened (by some obstructive student). Let us first assume that *sw1* is in its original position and that the light is still off. Hence, we find ourselves in another state which may or may not (depending on whether the window was initially open or not) equal the one which executing the plan started at.

Obviously, an execution monitoring component should resume the plan execution from the M-node corresponding to the most recent world state, i.e., execute (or reexecute) the action *close-window*. In this respect, the functionality of our system is as powerful as STRIPS's [Fikes *et al.*, 1972] facilities based on triangle tables.

Thus, monitoring the plan execution is focused on monitoring possible models. Note that this feature of our approach also constrains the choice of L: Basing a planner plus execution monitor on the possible models framework will work only if the atomic sentences set L consists of all relevant observables, where relevant means that these observables allow to conclude that the plan execution is still on the right track if they have the expected truth values.

Another form of focusing concerns the part of the (S, K, A) graph, that can be kept in the focus of the execution monitor. At planning time, many features of the actual start situation for plan execution may be unknown, resulting in alternative paths through the graph. At the start of the execution, many of these unknown features may become known; in our example, the values of *open*, *sw1*, and *sw2* (and other derived features) can be observed, resulting in a unique possible start model. If the plan execution can be assumed to be sufficiently reliable, then the plan executor can strip off everything from the plan which is not on the possible paths from the current M-node to goal M-nodes. If, e.g., the start situation in the lecture scenario corresponds to the M-node M2 in figure 1, all non-successors of this node can be stripped off.

The straightforward algorithm for extracting the relevant part of a graph for a particular situation is given in [Thiébaux, 1991]. Note that extraction can proceed as execution proceeds and more and more information becomes available, narrowing down the plan and describing the expected course of things less and less ambiguously. Should, as by obstructive students, a world state occur that does not correspond to an M-node in the current focus, then you have to download the original plan and try to locate the now current M-node.

4.2 REPLANNING

However, you may fail to locate such node in the original plan, which leads to replanning. In our lecture example consider the case of the aforementioned obstructive student opening the window right after you managed to turn on the light. The resulting world state includes both *on* and *open*, and there is no corresponding M-node in the complete plan. To get out of this situation, you have to replan.

As usual, replanning is similar to planning, and the only interesting question is to what extent you can inherit effort from the previous planning. Concerning this, note that an (S, K, A) graph can be extended incrementally to include more M-nodes; in order to ascertain g-achievingness of the new graph, you only must make sure that there is a path from every new M-node to an old M-node or to a newly introduced goal node. Again, the finiteness of the set of M-nodes provides this process with an upper limit. In PASCALE, we use the strategy to stop replanning as soon as the resulting graph is g-achieving. This may be achieved by inserting new M-nodes corresponding to unexpected world states as successors of the *Start* task and connecting these new M-nodes to old ones. As there might have been a shorter path from the new M-node to a *new* goal M-node, the new plan may be suboptimal. This strategy for reusing the old plan is convenient, but it is not enforced by the use of possible models, and other strategies are conceivable.

In our example, assuming that the initial state of the world corresponded to node M2 in figure 1, the unexpected situation with the light on and the window open would lead to introducing another M-node $\{open, \neg sw1, \neg sw2\}$, which would be connected via a new *close-window* T-node to the old goal node M5.

4.3 REUSING OLD PLANS

In fact, replanning as described here is just a special case of the more general technique of reusing a plan from an existing library of plans. Let us start with describing a simple case of plan reuse.

Given two goals g and h, if $g \vdash h$, then every g-achieving graph is also h-achieving, i.e., you can reuse a plan achieving g for obtaining h. To make a silly lecture scenario example, you can reuse the graph in figure 1 to achieve $sw1 \leftrightarrow sw2$. But note two things: First, the resulting plan need not be optimal (there may be simpler ways to generate h); and second, reusing here doesn't mean that the plan is really finished (there may be possible initial worlds not contained in the old M-nodes, for which you must replan to achieve h).

In general, you will not simply find solutions to instances of your current problem in your plan library. In this case, it is still possible to find "useful" parts of old plans, extract them using the extraction mechanism sketched above, and extend the resulting plan embryos to become valid h-achieving. The crucial question is the definition of usefulness; in [Thiébaux, 1991], a numerical measure is used to state to what extent leaves of old plans and possible models for the recent goal overlap. A detailed discussion, however, lies out of scope here.

5 FURTHER RELATED WORK

Let us conclude with some further remarks comparing our concepts with related ones. For a summary, please reread the abstract.

Situation-action rules. Another approach to encoding reactivity are situation-action rules, e.g., [Drummond, 1989; Drummond and Bresina, 1990; Downs and Reichgelt, 1991]. An (S, K, A) graph can obviously be translated into a set of situation-action rules (both for achieving the same goal) by interpreting the possible models corresponding to the M-nodes as IF parts of a rule, and the T-node task as the THEN part. A graph has just more structure than a set of rules—no more and no less. The particular struture of T-M graphs prevents what would be conflicts between two applicable rules. Thus, the graph focuses the execution monitor, allowing to choose the next execution step deterministically. Moreover, the technique of extraction is obvious in the graphs, but far less obvious in rule sets. Finally, the (S, K, A) graphs are conceptually closer to classical plans than are rule sets, allowing direct replanning.

Universal Plans. To provide the extreme case of plan reuse for a goal g, you can generate (S, K, A) graphs containing M-nodes corresponding to the whole of $Poss(K)$ in advance. This could be done by giving no information about the start situation, i.e., the *Start* node of the (S, K, A) graph to be generated has successor M nodes corresponding to all $Poss(K)$ elements. Thus, you generate a kind of universal plan [Schoppers, 1989b] for the domain for achieving g. However, this remark is not intended to actually recommend generating these universal plans, but just to state that our theory contains this as an extreme case.

Acknowledgements

This work is partially funded by the German Federal Ministry for Research and Technology (BMFT) in the joint project TASSO under grant ITW8900A7. TASSO is also part of the GMD Assisting Computer (AC) *Leitvorhaben.*

Thanks to the following persons for commenting on drafts: Marie-Odile Cordier, Tom Gordon, Hans Werner Güsgen, Robert Morris, Eric Rutten, Josef Schneeberger.

References

[Agre, 1988] P.E. Agre. *The Dynamic Structure of Everyday Life.* PhD thesis, MIT Artificial Intelligence Laboratory, 1988.

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

[Brewka and Hertzberg, 1991] G. Brewka and J. Hertzberg. How to do things with worlds: On formalizing actions and plans. Submitted for publication, 1991.

[Chapman, 1987] D. Chapman. Planning for conjunctive goals. *J. Artificial Intelligence*, 32:333–377, 1987.

[Dean and Boddy, 1988] T.L. Dean and M. Boddy. An analysis of time-dependent planning. In *Proc. AAAI-88*, pages 49–54, 1988.

[Downs and Reichgelt, 1991] J. Downs and H. Reichgelt. Integrating classical and reactive planning within an architecture for autonomous agents. In J. Hertzberg, editor, *European Workshop on Planning. EWSP '91, Sankt Augustin, FRG, March 1991, Proceedings*, pages 13–26. Springer Verlag (Lecture Notes in AI vol. 522), 1991.

[Drummond and Bresina, 1990] M. Drummond and J. Bresina. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proc. AAAI-90*, pages 138–144, 1990.

[Drummond, 1989] M. Drummond. Situated control rules. In *Proc. 1st Intl. Conference on Principles of Knowledge Representation and Reasoning (KR-89)*, pages 103–113, 1989.

[Fikes and Nilsson, 1971] R.E. Fikes and N.J. Nilsson. STRIPS: A new approach to theorem proving in problem solving. *J. Artificial Intelligence*, 2:189–208, 1971.

[Fikes *et al.*, 1972] R.E. Fikes, P.E. Hart, and N.J. Nilsson. Learning and executing generalized robot plans. *J. Artificial Intelligence*, 3:251–288, 1972.

[Georgeff and Lansky, 1987] M.P. Georgeff and A.L. Lansky. Reactive reasoning and planning. In *Proc. AAAI-87*, pages 677–682, 1987.

[Ginsberg and Smith, 1988] M.L. Ginsberg and D.E. Smith. Reasoning about action I: A possible worlds approach. *J. Artificial Intelligence*, 35:165–195, 1988.

[Hanks, 1990] S. Hanks. Practical temporal projection. In *Proc. AAAI-90*, pages 158–163, 1990.

[Kabanza, 1990] F. Kabanza. Synthesis of reactive plans for multi-path environments. In *Proc. AAAI-90*, pages 164–169, 1990.

[Lifschitz, 1987] V. Lifschitz. On the semantics of STRIPS. In M.P. Georgeff and A.L. Lansky, editors, *Proc. 1986 Workshop Reasoning about Actions and Plans, Timberline, OR*, pages 1–9, Los Altos, 1987. Morgan Kaufmann.

[Lifschitz, 1990] V. Lifschitz. Frames in the space of situations. *J. Artificial Intelligence*, 46:365–376, 1990.

[Reichgelt and Shadbolt, 1990] H. Reichgelt and N. Shadbolt. A specification tool for planning systems. In *Proc. ECAI-90*, pages 541–546, 1990.

[Sandewall, 1991] E. Sandewall. Features and fluents. Technical Report LiTH-IDA-R-91-29, Dept. of Computer and Information Sciences, Linköping University, 1991.

[Schoppers, 1989a] M.J. Schoppers. In defense of reaction plans as caches. *AI Magazine*, 10(4):51–60, 1989.

[Schoppers, 1989b] M.J. Schoppers. *Representation and Automatic Synthesis of Reaction Plans.* PhD thesis, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, 1989. also as Tech. Rep. UIUCDCS-R-89-1546.

[Thiébaux, 1991] S. Thiébaux. PASCALE: A system for planning under uncertainty. TASSO-Report 28, GMD, 1991.

[Waldinger, 1977] R. Waldinger. Achieving several goals simultaneously. *Machine Intelligence*, 8:94–136, 1977.

[Winslett, 1988] M. Winslett. Reasoning about action using a possible models approach. In *Proc. AAAI-88*, pages 89–93, 1988.