APS, a Prolog-Based Anytime Planning System

Helmut Prendinger* Mitsuru Ishizuka
Department of Information and Communication Engineering
School of Engineering, University of Tokyo
7-3-1, Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
E-mail: {helmut,ishizuka}@miv.t.u-tokyo.ac.jp

Abstract

We introduce APS (Anytime Planning System), a planner that finds *plausible* (or approximate) plans quickly and can be interrupted at (almost) any time [5]. The generated plans may depend on unproved default assumptions that can be completely proved as time permits [6]. APS includes an efficient mechanism to determine (in)consistency of new observations with an already constructed (successful) plan, a simple mechanism for plan repair, and control modules.

1 Introduction and Motivation

In this paper we challenge the view that the 'planning as theorem proving' paradigm using Prolog technology is inappropriate for dynamically changing environments. The main argument is that a theorem proving process cannot be interrupted, and therefore exogenous events that call for immediate action are ignored by such a planner [1]. In dynamic environments however, a planner has to respond rapidly to observations that possibly outdate an already constructed plan (referred to as *reactive control* [1, 15, 2]), while at the same time keep track of the achievement or maintenance of a high-level goal (called *classical* planning [7, 16]).

A software agent assisting a user performing routine tasks such as downloading and printing a document from the Web is suggestive for the type of agent under discussion. On the one hand, the agent has to generate a sequence of actions, for instance: *click_link*, then *download*, then *print*; on the other, the agent needs to react to the state of the network, for instance, if a virus is detected, some 'emergency' action should be carried out immediately.

There are only few attempts to combine both planning and reactive control in a logic programming framework. Kowalski and Sadri [10] propose a proof procedure that utilizes goal-reduction rules for the planning task and encodes (reactive) condition-action rules as integrity constraints. A simple version of their observation-thought-action cycle works as follows:

to *cycle* at time t;
(i) observe any input at time t;
(ii) 'assimilate' the input at time $t+1$;
(iii) reduce current goals to subgoals for n time steps;
(iv) perform action(s) at time $t+n+2$;
(v) cycle at time $t+n+3$;

The parameter n in (iii) is indicative of the resources spent on planning. However, it is not clear in their approach what the agent outputs in those cases where neither a (reactive) condition-action rule has to be triggered (since there are resources left), nor a complete plan can be invented (since there are *not* that much resources left). To this end, we will introduce the notion of an 'approximate plan'. It seems to us that their framework additional resources are only needed to *choose* an atomic action or precompiled plan among alternatives.

We will introduce an *anytime* classical planner that is interruptible and hence exhibits a form of reactive behavior. When interrupted during plan generation, the system outputs at least an approximate plan. In short, the planner utilizes three modules:

- **Approximate Plan Generation Module.** The planner constructs a plan that is approximate in the sense that it may depend on unproved (default) assumptions, expressed by the 'negation-as-failure' (NAF) operator **not** [4]. Hence the planner is unsound; yet, under 'favorable' conditions, the anytime planner can generate a plan that achieves the goal as desired. In this case we are given a workable plan together with constraint conditions (associated with action preconditions) that have not yet been shown to hold. We assume that plans gen-

*This research was funded by a fellowship of the Japan Society for the Promotion of Science (JSPS).

erated by an approximate planning processs can indeed be obtained quickly.

- **Constraint Condition Checking Module.** If the (approximate) plan is to be applied again, we have to check whether observations defeat some default precondition of the plan. This part is handled by a special mechanism (CONSTRUCT-AND-REMEMBER mechanism) that can detect undercutted default preconditions efficiently.

- **Response Module.** Depending on the way observations affect (or not affect) constraint conditions, an appropriate response is suggested: either a REPAIR module or an EMERGENCY module is employed.

Of course, the full power of the proposed planning system is only in effect if a certain goal has to be achieved multiple times at subsequent time points. Yet, many applications related to information technology have this feature: sending or retrieving files, building up an e-mail or telnet connection, and even more advanced tasks that could be handled by an intelligent software agent. It is assumed that the web application domain does not require extensive planning such as spacecraft assembly, nor do fluents change so frequently as in video games like Pengo [1].

The rest of the paper is organized as follows. In Section 2, situation calculus is introduced as a planning formalism for *classical* planning. Section 3 briefly discusses the concept of *reactive* planning (control). The core part of this paper is Section 4. There it is explained (i) what we mean by an approximate plan, (ii) how new evidence can have an impact on an already construced (approximate) plan, and (iii) how to respond to different kinds of observations. Section 5 discusses the paper and related work, and Section 6 gives a conclusion.

2 Classical Planning

A classical planning agent performs the following steps: (i) any observations about the initial situation are made; (ii) a plan that achieves the goal is constructed; (iii) the plan is executed. Note that the world must be 'frozen' while planning, else classical planning is not possible. Hence classical planning is actually concerned with *preplanning* where a plan is completed prior to execution. If some fluent relevant to the plan under consideration changes its truth-value spontaneously, it is possible that some precondition crucial to the plan is not satisfied any more. In this case, classical planners have to repair the plan, or even construct a new plan from scratch. Within classical planning, this task is called *replanning* [16].

Classical planning problems will be expressed in the situation calculus (SC) [12], a well-known method to model changing worlds in first-order logic. SC assumes that all changes to the world are brought about by named *actions* that transform the situation where the action is initiated to a situation where the action terminates. (Later on, we will allow that unnamed actions occur that are only visible by their results, namely observations.) For instance, $download(f, comp)$ can denote the action of downloading file f to computer $comp$. Let the constant $s0$ denote the *initial* situation, that is, the situation where no action has occurred. Then

do(lpr(f),do(download(f,comp),s0))

is the situation resulting from first downloading f to $comp$ in situation $s0$, and then printing out (the file denoted by) f. The first-order term $do(\alpha, S)$ denotes the situation attained by executing action α in situation S, where $do/2$ is a distinguished binary function symbol. Notice that plans in SC are simply first-order terms denoting a situation, where the plan sequence can be read off from right to left. Moreover, SC has (relational) *fluents* such as $loc(f, d)$ ("the location of file f is directory d"), where $loc/2$ is a predicate symbol. We will use the $holds(P, S)$ predicate to express that some relation P holds in a situation S. Here we employ the method of *reification* that treats atomic formulas, for instance, $loc(myfile, d)$, as terms.

A *(classical) planning theory* T consists of the following parts: (i) domain-independent theory; (ii) domain theory; (iii) problem description and goal.

Domain-independent Theory. The domain-independent theory is captured by means of the definition of the $holds/2$ predicate.

(DI-1) holds(P,do(A,S)) :- causes(A,S,P).
(DI-2) holds(P,do(A,S)) :- holds(P,S), **not** cancels(A,S,P).

(DI-1) says that a fluent P holds in the situation after an action A is performed in a situation S, if the action 'causes' the fluent to become true. (DI-2) encodes the commonsense 'law of inertia': a fluent P holds in the situation resulting from the performance of an action A, if P was true before the execution of the action and action A cannot be proved to 'cancel' the fluent (reverse its truth-value) in situation S. Recall that **not** F means "it is not provable that F" (F a ground atomic formula).

Domain Theory. The domain theory describes how fluents are affected by actions.

(d1) causes(click_link(X),S,selected(X)) :-
    poss(click_link(X),S).
(d2) causes(download(X,Y),S,on_disk(X,Y)) :-
    holds(selected(X),S), holds(comp(Y),S),
    holds(has_capacity_for(Y,X),S),

poss(download(X,Y),S).
(d3) causes(lpr(X),S,printed(X)) :-
holds(on_disk(X),S), poss(lpr(X),S).
(d4) causes(enter_pw(X,Y),S,connected(X,Y)) :-
poss(enter_pw(X,Y),S).
(d5) poss(click_link(X),S) :- holds(link(X),S).
(d6) poss(download(X,Y),S) :-
holds(comp(Y),S), holds(provider(Z),S),
not holds(network_down(Y,Z),S),
not holds(permission_denied(Z,Y),S).
(d7) poss(lpr(X),S) :- true.
(d8) poss(enter_pw(X,Y),S) :-
holds(comp(X),S), holds(provider(Y),S),
holds(know_pw(X),S).
(d9) holds(network_down(X,Y),S) :-
holds(server_down(X,Y),S).
(d10) holds(network_down(X,Y),S) :-
holds(client_down(X,Y),S).
(d11) holds(network_down(X,Y),S) :-
holds(connection_defect(X,Y),S).
(d12) holds(server_down(X,Y),S) :-
holds(maintenance(X,Y),S),
not holds(proxy(X,Y),S).
(d13) holds(client_down(X,Y),S) :-
holds(hit_disconnect(X,Y),S).
(d14) holds(connection_defect(X,Y),S) :-
holds(broken_line(X,Y),S).

The *causes*/3 predicate defines the effects of actions, for instance, clicking a hypertext link (referring to a document) 'causes' the document to be selected. Clauses (d9–11) describe three possibilities for a broken network. The possibilities are given a more detailed explanation in clauses (d12–14). Note that clauses (d9–14), introduced to explain *network_down*/2, are 'static' since they do not introduce new actions, and are thus not subject to the law of inertia. As usual we distinguish between two types of preconditions [14].

Fluent preconditions. They state under what conditions a fluent becomes true in the situation resulting from performing an action. Downloading a file to the hard-disk succeeds if the file is selected and the hard-disk of the computer has enough capacity to store the file. Fluent preconditions are positive literals expressed by means of the *holds*/2 predicate.

Action preconditions. These are the conditions that enable an action to be carried out. Action preconditions are collected in the body of clauses that have the *poss*/2 predicate in their head. Action preconditions can be positive or negative literals. A document can be downloaded given, among other conditions, it is not possible to show that the network (connecting client and server) is down.

Problem Description and Goal. The problem description contains the description of the objects of the domain, together with the description of the initial situation $s0$.

(pr1) holds(link(file),s0).
(pr2) holds(has_capacity_for(myc,file),s0).
(pr3) holds(comp(myc),s0).
(pr4) holds(provider(myp),s0).
(pr5) holds(knows_pw(myc),s0).
(g) ?- holds(printed(file),Plan).

For instance, clause (pr2) says that *myc* has enough capacity to store file $file$ in situation $s0$. The query (g) is the planning goal.

What is a Plan? The calculus outlined above can be used to *generate* linear plans (sequences of actions) as a side-effect of the theorem-proving task to answer the query

?- holds(printed(file),Plan).

If $\pi = \{Plan/do(lpr(file),\ do(download(file, myc),\ do(click\_clink)))\}$ is an answer substitution for the query $?-holds(printed(file), Plan)$, then π is a correct plan to achieve $printed(file)$, corresponding to the action sequence

[click_link(file);download(file,myc);lpr(file)]

SC planning is sound and complete [6], and semi-decidable, that is, given a conjunction of goal fluents, if there is a situation (a plan) that simultaneously satisfies them, it can be found.

3 Reactive Planning

The idea of 'reactive' planning tries to overcome restrictions of classical planners in that it accounts for unpredictable developments in a domain. Reactive plans are based on the concept of *universal* plans [15]: informally, all situations an agent may encounter are generated, and for each of these situations, an atomic action is prescribed. Procedurally, universal plans are sets of condition-action rules of the form "**if** c **then** α", read as "if condition c holds, then execute action α".

By way of example, consider the following *control module* that is supposed to maintain the goal $\neg virus$ (no virus is present).

if $malevolent\_virus$	**then**	$shutdown\_comp$
if $known\_virus$	**then**	$activate\_scan\_disc$
if $\neg virus$	**then**	$do\_nothing$

Typically one never uses 'exact' universal plans, but 'approximate' versions that suggest actions only for situations the agent is likely to encounter.

4 Anytime Planning System

In this section, we introduce APS, an anytime planner for dynamically changing environments.

4.1 What is Approximated?

Since we propose to generate plans that depend on unproved assumptions (approximate plans), it is important to decide *which* assumptions are concerned. Fortunately, the (planning) domain theory makes explicit which conditions in the formulation of action axioms have the status of *strict* (pre-)conditions and which have the status of *default* (pre-)conditions (expressing exceptional conditions). Naturally, default conditions of the form **not** F may be plausibly assumed [6, 8]. APS will temporarily ignore conditions that usually hold and will actively try to achieve strict preconditions. It is important to note that unproved negated ground conditions **not** $p(\bar{t})$ never leave 'holes' in the answer substitution produced by the planner (the plan), since goals of the form **not** $p(\bar{t})$ can only be true or false. Currently, we use the iterative deepening strategy to cut off the search tree expanding from negated conditions. In each iteration a 'less approximate' plan is invented (one that depends on fewer assumptions).

Besides their appearance in the domain theory, a condition of the form **not** F also occurs in the formulation of the law of inertia. However, **not** $cancels(A, S, P)$ in the domain-independent theory serves a totally different task than default conditions in the domain theory. It asserts that it is not provable that the planner inserts an action to the plan that reverses the truth-value of an already established fluent. The condition corresponds to the famous *modal truth criterion* (MTC) of Chapman's TWEAK planner [3]. In general, we argue that approximations should be justified with the (temporary) assumption of a 'normal' world where default conditions can be assumed to hold. The modal truth criterion (expressed by means of **not** here), on the other hand, concerns the internal workings of how the planner generates plans. However, if plan generation is expensive, the MTC can be left unproved as well. However, we have no clear idea about the usefulness of the outcome in this case.

4.2 The Impact of Observations

Starting point. The planner has produced an approximate plan that achieves its goal. APS records the conditions under which the reasoning process happened to succeed, that is, constraint conditions associated with default conditions in the rule bodies of clauses with head $poss/2$. Constraint conditions are essentially complex data structures that are are computed by the CONSTRUCT-AND-REMEMBER mechanism [13], and can be efficiently checked for (in)consistency with incoming information.

Observations that are made *after* an approximate plan is generated can be classified according to their impact on the plan. We distinguish the following possibilities: either observations are consistent with constraint conditions, or they are inconsistent with constraint conditions, such that (i) the system aborts, or (ii) the REPAIR module generates a new plan, or (iii) the EMERGENCY module is activated.

Assume APS has generated the approximate plan

[click_link(file);download(file,mycomp);lpr(file)]

and, at depth 0, a constraint condition Γ_α for the download action (α for short),

$$\Gamma_\alpha = \{\langle \mathbf{not} holds(network\_down(myc, myp), s1), \\ \mathbf{not} holds(permission\_denied(myc, myp), s1)\rangle\}$$

where $s1 = do(click\_link(file), s0)$. Suppose the plan is correct and we are about to download another file denoted by $newfile$. The new query is

?- holds(printed(newfile),NewPlan).

In order to take advantage of the former reasoning process, we have to make sure that the constraint conditions (there is only one for the download action) are still satisfied.

Observations consistent with constraints

Case 1. If new observations are consistent with the constraint conditions, the approximate plan has to be reinvented nevertheless since the CONSTRUCT-AND-REMEMBER mechanism only detects inconsistencies as far as constraints regarding default conditions in action preconditions are concerned (but recall our assumption that approximate plans can be generated quickly).

Assume we make the new observation

(o1) holds(new_e_mail(myc),s3)

The new problem description describes the situation at $s3$ where $s3 = [click\_link(file); download(file, myc); lpr(file)]s0$. It includes the facts (pr1–5) that are derived via the law of inertia, together with the effects of plan execution ($selected(file)$, $on\_disc(file, myc)$, and $printed(file)$). The question is whether the new observation is consistent with constraint Γ_α. Notice that Γ_α refers to situation $s1$ whereas the observations refer to situation $s3$. The satisfaction of the constraint can be safely propagated to successor situations $S > s1$, given that new observations including effects described by the action axioms, do not violate the constraint. It is easily seen that (o1) does not violate the constraint.

Observations inconsistent with constraints

Case 2. Take the same set of observations as above plus the evidence that the server is under maintenance (hence temporarily not accessible), represented as

(o2) holds(maintenance(myc,mypr),s3).

As seen in the figure below, there is no provision to satisfy the constraint; in particular, $holds(proxy(myc,myp),s3)$ fails and the domain theory does not contain any rescue actions (for instance, trying to download the paper from a different server). Observations such as (o1) are called *destructive* because there is no way to repair the plan. The planner aborts execution in this case.

holds(network_down(myc,myp),s3)
SUCCESS

holds(server_down(myc,myp),s3)

holds(maintenance(myc,myp),s3)
SUCCESS

notholds(proxy(myc,myp),s3)

holds(proxy(myc,myp),s3)
FAILURE

Case 3. This case is more interesting since a new query (possibly) leading to a working plan is automatically triggered. Defeating observations that lead to a new plan are called *constructive*.

The same observations as in Case 1 are made, but with an additional observation formalized as

(o3) holds(hit_disconnect(myc,myp),s3).

Intuitively, the user (accidentally) hit the "cancel connection" key (breaking the connection between client and server), thereby violating the constraint condition Γ_α.

holds(network_down(myc,myp),s3)
SUCCESS

holds(client_down(myc,myp),s3)

holds(hit_disconnect(myc,myp),s3)
SUCCESS

Unlike the situation where the server was down the domain theory provides a 'rescue' action for this situation. To find out about available 'rescue' actions the planner has to look up a library that stores information about effects that (i) can be achieved by the planner and (ii) have an effect *opposite* to the condition that is responsible for the defeated constraint condition. For instance, this library may contain the item

opposite(client_down(X,Y),connected(X,Y)).

among other *opposite*/2 facts. Procedurally, the entry triggers the new query

?- holds(printed(newfile),NewPlan),
holds(connected(myc,myp),NewPlan).

The new plan proposes the 'enter password' action to restore consistency and reads

[enter_pw(myc,myp);click_link(newfile);
download(newfile,myc);lpr(newfile)]

The process to restore consistency is not completely general since only fluents that are opposite to conditions mentioned in certain positions in the constraint table are able to trigger rescue actions.

Observations call for immediate action

In certain circumstances it is not advisable to proceed by trying to execute a (repaired) plan, and some emergency action has to be taken, for instance, when a 'killer' virus is detected. For those situations the EMERGENCY module is employed.

4.3 The APS Mechanism

APS can be summarized as follows.
Input a classical planning theory T with goal G.

1. For G generate an approximate plan π such that (i) $\pi = [\alpha_1; ...; \alpha_n]$ is correct, and (ii) the deadline (maximal depth d) of constructing constraint conditions Γ_{α_i} ($1 \leq i \leq n$) is not met;
2. Record approximate plan π and Γ_{α_i}'s;
3. Record constraint conditions for each α_i in π;
4. Cycle at time t,

(i) Observe input $O(t)$ at t;
(ii) Update fact base of T with $O(t)$ at $t+1$;
(iii) Check inputs for consistency with Γ_π at $t+2$,
(iii-1) **if** inputs consistent with Γ_π,
then recompute an approximate plan π',
(iii-2) **else** employ the REPAIR module or EMERGENCY module (depending on $O(t)$);
(iv) Expand $\Gamma_{\pi'}$ by n depth steps;
(v) $d = d + n$; $t = t + n + 3$;
(vi) Cycle at time t;

The REPAIR module constructs a new (approximate) plan if this is possible. A detailed description of this module can be found in [13].

5 Discussion and Related Work

APS is reminiscent of the observation-thought-action cycle of Kowalski and Sergot [10] that was proposed as a framework unifying rational and reactive behavior. Similar to their approach, we try to interleave planning, sensing, execution. However, in APS the conceptual tie between the planning theory and the reactive component is rather loose: the EMERGENCY module is employed upon certain observations, but the module is distinct from the planning theory whereas in [10] reactivity is wired into the theory.

Baral and Son [2] attempt to establish a *formal* connection between theories of reasoning about action (as planning formalisms) and reactive control modules.

Lespérance *et al.* [11] have recently added a reactive component to CONGOLOG (a logic-based agent programming language). Their framework contains 'interrupts' that may suspend or abandon current plan *execution* whereas we provide a notion of interrupted plan *generation* that leads to approximate plans.

The notion of an 'approximate' plan is reminiscient of Elkan's use of approximate plans in his PERFLOG calculus. However, he follows a 'one-shot' approach where information gained during an interrupted reasoning process cannot be reused.

Ginsberg and Holbrook [9] also make the distinction between strict preconditions and default preconditions (in action precondition axioms). They show that the distinction lends itself to a natural assignment of priorities to goals which is important for *hierarchical* planning; strict conditions have high priority while default conditions are assigned lower priority.

6 Conclusion

In the spirit of anytime reasoning, we attempt to make classical planning (with situation calculus) more 'reactive' by allowing the planning process to be interrupted. The output of such an interrupted plan generation process is meaningful since it gives an approximate plan that may depend on unproved assumptions. The domain theory indicates which assumptions concern exceptional situations and can be temporarily ignored. The resulting Anytime Reasoning System is most appropriate for domains where goals have to be established multiple times and which require a modest form of reactivity, such as software domains.

References

[1] Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings AAAI-87*, pages 268–272, Seattle, WA, 1987.

[2] Chitta Baral and Tran Cao Son. Relating theories of action and reactive control, 1998. Submitted for publication.

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

[4] Keith L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[5] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings AAAI-88*, pages 49–54, 1988.

[6] Charles Elkan. Incremental, approximate planning. In *Proceedings AAAI-90*, pages 145–150, 1990.

[7] R. Fikes and N. J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[8] Matthew L. Ginsberg. The computational value of non-monotonic reasoning. In *Proceedings KR-91*, pages 262–268, Cambridge, Massachusetts, 1991.

[9] Matthew L. Ginsberg and Hugh W. Holbrook. What defaults can do that hierarchies can't. In *Proceedings of the 1992 Nonmonotonic Resoning Workshop*, 1992.

[10] Robert A. Kowalski and Fariba Sadri. Towards a unified agent architecture that combines rationality with reactivity. In *Proceedings International Workshop on Logic in Databases*, LNCS 1154, 1996.

[11] Yves Lespérance, Kenneth Tam, and Michael Jenkin. Reactivity in a logic-based robot programming framework. In *AAAI Fall Symposium in Cognitive Robotics*, 1998.

[12] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of Artificial Intelligence. In B. L. Webber and N.J. Nilsson, editors, *Readings in Artificial Intelligence*, pages 431–450. Morgan Kaufmann Publishers, Inc., 1981, Los Altos, California, 1969.

[13] Helmut Prendinger. *Towards an Agent Architecture for Dynamically Changing Environments.* PhD thesis, Department of Computer Science/Logic and Philosophy of Science, University of Salzburg, 1998.

[14] Raymond Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation. Papers in Honor of John McCarthy*, pages 359–380. Academic Press, Inc., Boston, New York, 1991.

[15] M. J. Schoppers. Universal plans for reactive robots in unpredictable domains. In *Proceedings IJCAI-87*, pages 1039–1046, 1987.

[16] David E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann Publishers Inc., San Mateo, CA, 1988.