Incremental, Approximate Planning

Charles Elkan
Department of Computer Science
University of Toronto*

ABSTRACT: This paper shows how using a nonmonotonic logic to describe the effects of actions enables plausible plans to be discovered quickly, and then refined if time permits. Candidate plans are found by allowing them to depend on unproved assumptions. The nonmonotonic logic makes explicit which antecedents of rules have the status of default conditions, and they are the only ones that may be left unproved, so only plausible candidate plans are produced. These are refined incrementally by trying to justify the assumptions on which they depend. The new planning strategy has been implemented, with good experimental results.

1 Introduction

Because of uncertainty and because of the need to respond rapidly to events, the traditional view of planning (deriving from STRIPS [Fikes *et al.*, 1972] and culminating in TWEAK [Chapman, 1987]) must be revised drastically. That much is conventional wisdom nowadays. One point of view is that planning should be replaced by some form of improvisation [Brooks, 1987]. However an improvising agent is doomed to choose actions whose optimality is only local. In many domains, goals can only be achieved by forecasting the consequences of actions, and choosing ones whose role in achieving a goal is indirect. Thus traditional planners must be improved, not discarded.

This paper addresses the issue of how to design a planner that is incremental and approximate. An approximate planner is one that can find a plausible candidate plan quickly. An incremental planner is one that can revise its preliminary plan if necessary, when allowed more time.

It is not clear how existing planning strategies can be made approximate and incremental. We therefore first outline a strategy for finding guaranteed plans using a new formalism for specifying planning problems, and then show how to extend this guaranteed strategy to make it approximate and incremental.

Our approach draws inspiration from work on abductive reasoning. A plan is an explanation of how a goal is achievable: a sequence of actions along with a proof that the sequence achieves the goal. An explanation is abductive (as opposed to purely deductive) if it depends on assumptions that are not known to be justified. We find approximate plans by allowing their proofs of correctness to depend on unproved assumptions. Our planner is incremental because, given more time, it refines and if necessary changes a candidate plan by trying to justify the assumptions on which the plan depends.

The critical issue in abductive reasoning is to find plausible explanations. Our planning calculus uses a nonmonotonic logic that makes explicit when an antecedent of a rule has the epistemological status of a default condition. The distinguishing property of a default condition is that it may plausibly be assumed. These antecedents are those that are allowed to be left unjustified in an approximate plan. Concretely, every default condition in the planning calculus expresses either a claim that an achieved property of the world persists, or that an unwanted property is not achieved. Thus the approximate planning strategy only proposes reasonable candidate plans.

Sections 2 and 3 below present the formalism for specifying planning problems and the strategy for finding guaranteed plans. In Section 4 the strategy is extended to become approximate and incremental. Section 5 contains experimental results, and finally Section 6 discusses related and future work.

2 The planning formalism

Different formal frameworks for stating planning problems vary widely in the complexity of the problems they can express. Using modal logics or reification, one can reason about multiple agents, about the temporal prop-

*For correspondence: Department of Computer Science, University of Toronto, Toronto M5S 1A4, Canada, (416) 978-7797, cpe@ai.toronto.edu.

erties of actions, and about what agents know [Moore, 1985; Konolige, 1986; Cohen and Levesque, 1990]. The simplest planning problems can be solved by augmented finite state machines [Brooks *et al.*, 1988], whose behaviour can be specified in a propositional logic. The planning problems considered here are intermediate in complexity. They cannot be solved by an agent reacting immediately to its environment, because they require maintaining an internal theory of the world, in order to project the indirect consequences of actions. On the other hand, they involve a single agent, and they do not require reasoning about knowledge or time.

Our nonmonotonic logic for specifying this type of planning problem is called the PERFLOG calculus.[1] Technically, the calculus is the language of locally stratified definite clauses with the minimal model semantics of [Przymusiński, 1987] and certain "laws of nature" presented below. The PERFLOG calculus is distinctive in that it has a well-defined first-order semantics *and* it is practically usable for planning. Other proposed planning formalisms with a well-defined semantics either do not have first-order expressiveness (for example the TWEAK calculus [Chapman, 1987]), or else they use logics for which simple proof procedures capable of inventing plans are not known (for example the circumscriptive calculus of [Lifschitz and Rabinov, 1989]).

The Yale shooting problem [Hanks and McDermott, 1986] is at the simple end of the spectrum of planning problems for which the PERFLOG calculus is appropriate. It serves here to introduce the calculus by example. We start with the laws of nature mentioned above. In the following rules, think of s as denoting a state of the world, of a as denoting an action, and of $do(s, a)$ as denoting the state resulting from performing the action a in the initial state s. Finally, think of p as denoting a contingent property that holds in certain states of the world: a fluent.

$$\forall a, s, p \;\; causes(a, s, p) \rightarrow holds(p, do(s, a)) \qquad (1)$$

$$\forall a, s, p \;\; holds(p, s) \wedge \neg cancels(a, s, p) \rightarrow holds(p, do(s, a)). \qquad (2)$$

The rules (1) and (2) are frame axioms. Rule (1) captures the commonsense notion of causation, and rule (2) expresses the commonsense "law of inertia": a fluent p holds after an action a if it holds before the action, and the action does not cancel the fluent. Note that since in addition to a, one argument of *causes* and of *cancels* is s, the results of an action (that is, the fluents it causes and cancels) may depend on the state in which the action is performed, and not just on which action it is.

[1] PERFLOG is an abbreviation for "performance-oriented perfect model logic."

Given rules (1) and (2), a particular planning domain is specified by writing axioms that mention the actions and fluents of the domain, and say which actions cause or cancel which fluents. In the world of the Yale shooting problem, there are three fluents, *loaded*, *alive*, and *dead*, and three actions, *load*, *wait*, and *shoot*. The relationships of these fluents and actions are specified by the following axioms:

$$\forall s \;\; causes(load, s, loaded) \qquad (3)$$

$$\forall s \;\; holds(loaded, s) \rightarrow causes(shoot, s, dead) \qquad (4)$$

$$\forall s \;\; holds(loaded, s) \rightarrow cancels(shoot, s, alive) \qquad (5)$$

$$\forall s \;\; holds(loaded, s) \rightarrow cancels(shoot, s, loaded). \qquad (6)$$

The initial state of the world s_0 is specified by saying which fluents are true in it:

$$holds(alive, s_0). \qquad (7)$$

According to the nonmonotonic semantics of PERFLOG collections of rules,

$$holds(dead, do(do(do(s_0, load), wait), shoot))$$

is entailed by rules (1)–(7). The Yale shooting problem is thus solved.

The advantages and disadvantages of the PERFLOG calculus will be discussed in a forthcoming paper. It can be extended to match the expressiveness of competing proposed nonmonotonic logics for reasoning about action. For the purposes of this paper, what is most important is that the calculus is usable for inventing plans, not just for specifying when a plan is correct. Given clauses (1)–(7) and the query $\exists p \; holds(dead, p)$?, the planning strategy of the next section quickly produces the answer substitution $p = do(do(x, load), shoot)$. (The variable x in the generated plan indicates that it works whatever the initial situation.)

3 Finding guaranteed plans

The previous section showed how to state the relationships between the actions and fluents of a planning domain as a PERFLOG set of axioms. This section describes a strategy for inventing plans using such a set of axioms; the next section extends the strategy to be approximate and incremental.

A PERFLOG set of axioms is general logic program, and our planning strategy is a four-point extension of the standard PROLOG procedure for answering queries against a logic program.

Iterative deepening. The standard PROLOG strategy can be viewed as depth-first exploration of an and/or tree representing the space of potential proofs

of the query posed by the user. Each or-node corresponds to a subgoal[2] that must be unified with the head of some clause, and each and-node corresponds to the body of a clause. The root of the tree, always an or-node, is the user's query. Depth-first exploration can be implemented many times more efficiently than other exploration patterns, but it is liable to get lost on infinite paths. These paths can be cut off by imposing a depth bound. The idea of iterative deepening is to repeatedly explore a search space depth-first, each time with an increased depth bound [Stickel and Tyson, 1985].

Conspiracy numbers. Iterative deepening algorithms differ in how the depth of a node is defined. The conspiracy idea underlies the best known way of defining depth in and/or trees. A conspiracy for a partially explored and/or tree is a minimal (with respect to subsets) set of subgoal leafs such that if all the subgoals in the conspiracy have compatible answer substitutions, then an answer substitution exists for the root goal of the tree. Suppose that for each member of a conspiracy, whether it has an answer substitution is a statistically independent event. Even so, as the conspiracy gets larger, heuristically the chance that all the members have compatible answer substitutions decreases as if these events were negatively correlated. Thus a good definition of the depth of a leaf node is the size of the smallest conspiracy to which it belongs. Conspiracy sizes can be computed efficiently [Elkan, 1989].

Negation-as-failure. The strategy described so far applies to positive subgoals only. Given a negated goal, the negation-as-failure idea is to attempt to prove the un-negated version of the goal. If this attempt succeeds, the negated goal is taken as false. If no proof exists for the un-negated goal, then the negated goal is taken as true. Negation-as-failure is combined with iterative deepening by using the conspiracy depth measure to limit searches for proofs of un-negated notional subgoals corresponding to negated actual subgoals. If the and/or tree representing the space of possible proofs of a notional subgoal is completely explored, without finding a proof, then the corresponding actual negated subgoal is taken as true. If a proof of the notional subgoal is found, then the actual negated subgoal is taken as false. If exploration of the possible proofs of the notional subgoal is cut off by the current depth bound, it remains unknown whether or not the notional subgoal is provable, so for soundness the actual negated subgoal must be taken as false.

Freezing and constructive negation. Negation-as-failure only works on ground negated subgoals. Suppose the unit clauses $p(a)$ and $q(b)$ are given, and consider the query $\exists x\ \neg p(x) \wedge q(x)$?. This query should have one answer, $x = b$, but the strategy described so far produces no answer: naive negation-as-failure attempts to prove $p(x)$, succeeds, deems $\neg p(x)$ to be false, and fails on the whole query. The solution to this problem is to apply negation-as-failure to ground negated subgoals only. When a negated subgoal is encountered, it is postponed until it becomes ground. Concretely, in the example above $\neg p(x)$ is delayed, and $q(x)$ is solved, obtaining the substitution $x = b$. Now $\neg p(x)[x \mapsto b]$ is revived, and proved. This process is called freezing [Naish, 1986]. If postponement is not sufficient to ground a negated subgoal, then an auxiliary subgoal is introduced to generate potential answers. This process is called constructive negation [Foo *et al.*, 1988].

The performance of the planning strategy just described could be improved significantly, notably by caching subgoals once they are proved or disproved [Elkan, 1989]. Nevertheless it is already quite usable. More important as a basis for further work, it is sound and complete.

Lemma: The guaranteed planning strategy is sound.
Proof: Negation-as-failure is sound under the completion semantics for general logic programs [Clark, 1978]. The perfect model semantics allows a subclass of the class of models allowed by the completion semantics. Therefore given a query of the form $\exists p\ holds(\sigma, p)$?, if the strategy above returns with the answer substitution $p = \pi$, then $holds(\sigma, \pi)$ is true, and π is a correct plan. ∎

Completeness is a more delicate issue. In general, perfect models may be non-recursively enumerable [Apt and Blair, 1988], and all sufficiently expressive nonmonotonic logics have non-computable entailment relations. However PERFLOG theories all have a similar structure, using the same three fundamental predicates, so their completion and perfect model semantics essentially coincide, and the strategy above is complete.

4 Finding plausible plans

This section describes modifications to the strategy of the previous section that make it approximate and incremental. In the same way that the guaranteed planning strategy is in fact a general query-answering procedure, the incremental planning strategy is really a general procedure for forming and revising plausible explanations using a default theory.

Any planning strategy that produces plans relying on unproved assumptions is *ipso facto* unsound, but by its incremental nature the strategy below tends to soundness: with more time, candidate plans are either proved to be valid, or changed.

Approximation. The idea behind finding approximate plans is simple: an explanation is approximate

[2] Here and in similar contexts, 'goal' refers to a literal for which an answer substitution is wanted.

if it depends on unproved assumptions. Strategies for forming approximate explanations can be distinguished according to the class of approximate explanations that each may generate. One way to define a class of approximate explanations is to fix a certain class of subgoals as the only ones that may be taken as assumptions. Looking at the PERFLOG formalism, there is an obvious choice of what subgoals to allow to be assumptions. Negated subgoals have the epistemological status of default conditions: the nonmonotonic semantics makes them true unless they are forced to be false. It is reasonable to assume that a default condition is true unless it it is provably false.

There is a second, procedural, reason to allow negated subgoals to be assumed, but not positive subgoals. Without constructive negation, negated subgoals can only be answered true or false. Negation-as-failure never provides an answer substitution for a negated subgoal. Therefore unproved negated subgoals in an explanation never leave "holes" in the answer substitution induced by the explanation. Concretely, a plan whose correctness proof depends on unproved default conditions will never change because those defaults are proved to hold.

In more detail, the guaranteed planning strategy is modified as follows. When a negated subgoal becomes ground, the proof of its notional positive counterpart is attempted. If this attempt succeeds or fails within the current depth bound, the negated subgoal is taken as false or true, respectively, as before. However, if the depth bound is reached during the attempted proof, then the negated subgoal is given the status of an assumption.

Incrementality. An approximate explanation can be refined by trying to prove the assumptions it depends on. If an assumption is proved, the explanation thereby becomes "less approximate". As just mentioned, proving an assumption never causes a plan to change. On the other hand, if an assumption is disproved, the approximate plan is thereby revealed to be invalid, and it is necessary to search for a different plan.

Precisely, any negated subgoal is allowed to be assumed initially. Each iteration of iterative deepening takes place with an increased depth bound. For each particular (solvable) planning problem, there is a certain minimum depth bound at which one or more approximate plans can first be found. Each of these first approximate plans depends on a certain set of assumptions. In later iterations, only subsets of these sets are allowed to be assumed. This restriction has the effect of concentrating attention on either refining the already discovered approximate plans, or finding new approximate plans that depend on fewer assumptions.

```
% rules for how the world evolves

holds(P,do(S,A)) :-
        causes(A,S,P).
holds(P,do(S,A)) :-
        holds(P,S), not(cancels(A,S,P)).

% the effects of actions

causes(pounce(lion,X),S,eats(lion,X)) :-
        can(pounce(lion,X),S).
can(pounce(X,Y),S) :-
        holds(in(X,L),S), holds(in(Y,L),S),
        not(call(X = Y)),
        not(Z,holds(eats(X,Z),S)).
causes(jump(X),S,in(X,arena)) :-
        can(jump(X),S), holds(in(X,cage),S).
can(jump(lion),S) :-
        holds(eats(lion,centurion),S).
cancels(drop(X,Y),S,eats(X,Y)) :-
        can(drop(X,Y),S).
can(drop(X,Y),S) :-
        holds(eats(X,Y),S).
holds(in(X,H),S) :-
        holds(eats(lion,X),S),
holds(in(lion,H),S).

% the initial state of the world

holds(in(christian,arena),s0).
holds(in(lion,cage),s0).
holds(in(centurion,cage),s0).
```

Figure 1: The theory of a lion and a Christian.

5 Experimental results

Implementing the planning strategies described above is straightforward, because the PERFLOG calculus is based on directed clauses. In general, it is insufficiently realized how efficiently logics with this basis, both monotonic and nonmonotonic, can be automated. The state of the art in PROLOG implementation is about nine RISC cycles per logical inference [Mills, 1989]. Any PERFLOG theory could be compiled into a specialized incremental planner running at a comparable speed.

The experiment reported here uses a classical planning domain: a lion and a Christian in a stadium. The goal is for the lion to eat the Christian. Initially the lion is in its cage with a centurion, and the Christian is in the arena. The lion can jump from the cage into the arena only if it has eaten the centurion. The lion

eats a person by pouncing, but it cannot pounce while it is already eating. The PERFLOG theory in Figure 1 describes this domain formally.

Using the guaranteed planning strategy of Section 3, the query `holds(eats(lion,christian),P)?` is first solved with conspiracy depth bound 19, in 4.75 seconds.[3] The plan found is

```
P = do(do(do(do(s0,pounce(lion,centurion)),
                   jump(lion)),
                   drop(lion,centurion)),
                   pounce(lion,christian)).
```

Using the approximate planning strategy of Section 4, the same query is solvable in 0.17 seconds, with conspiracy depth bound 17. The candidate plan found is

```
P = do(do(do(s0,pounce(lion,centurion)),
                jump(lion)),
                pounce(lion,christian)).
```

This plan depends on the assumption that no Z exists such that

```
holds(eats(lion,Z),do(do(s0,pounce(lion,centurion)),
                                jump(lion))).
```

Although the assumption is false and the plan is not correct, it is plausible. Note also that the first two actions it prescribes are the same as those of the correct plan: the approximate plan is an excellent guide to immediate action.

6 Discussion

The strategy for incremental, approximate planning uses simplifying assumptions in a principled way: first the planner searches for a plan assuming that default conditions hold, then it attempts to prove that they do hold. The idea of relying on assumptions that are left unproven appears in [Feldman and Rich, 1986] and elsewhere. This paper shows how a formal nonmonotonic logic determines reasonable potential assumptions, and how iterative deepening can be used to modulate the effort expended on checking these assumptions. The point that default theories suggest how to focus inference is made independently in [Ginsberg, 1990]. To accommodate arbitrary sources of knowledge about plausible assumptions, our implementation allows the user to say explicitly what must always be proved, and what may sometimes be left unproved, as in [Chien, 1989].

From a knowledge-level point of view, approximate planning is a type of hierarchical planning. Each maximum conspiracy size defines a different abstraction space in which to search for plans. In each space the available actions and their effects are the same. However, the lower the maximum conspiracy size, the more each action is stripped of its difficult-to-check preconditions. Abstraction spaces defined in this way have the advantage that the execution of any plan invented using them can be initiated immediately if it is necessary to act instantly. Other hierarchical planners typically construct plans using abstract actions that must be elaborated before they can be executed [Knoblock, 1989].

Selecting good simplifying assumptions is a type of abductive inference. Abduction mechanisms have been investigated a great deal for the task of plan recognition, not so much for the task of inventing plans, and not at all for the task of inventing plausible plans. These three different tasks lead to different choices of what facts may be assumed. In the work of [Shanahan, 1989] for example, properties of the initial state of the world may be assumed. In our work, the facts that may be assumed say either that an established property of the world persists, or that an unestablished property does not hold.

An incremental approximate planner is an "anytime algorithm" for planning in the sense of [Dean and Boddy, 1988]. Anytime planning algorithms have been proposed before, but not for problems of the traditional type treated in this paper. For example, the real-time route planner of [Korf, 1987] is a heuristic graph search algorithm, and the route improvement algorithm of [Boddy and Dean, 1989] relies on an initial plan that is guaranteed to be correct.

For future work, one important direction is to quantify how an approximate plan is improved by allowing more time for its refinement. Another problem is to find a planning strategy that is focused as well as approximate and incremental. A focused strategy would be one that concentrated preferentially on finding the first step in a plan–what to do *next*.

[3] All times are for an implementation in CProlog, running on a Silicon Graphics machine rated at 20 MIPS.

References

[Apt and Blair, 1988] Krzysztof R. Apt and Howard A. Blair. Arithmetic classification of perfect models of stratified programs. In Kenneth Bowen and Robert Kowalski, editors, *Fifth International Conference and Symposium on Logic Programming*, volume 2, pages 765–779, Seattle, Washington, August 1988. MIT Press.

[Boddy and Dean, 1989] Mark Boddy and Thomas Dean. Solving time-dependent planning problems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 979–984, August 1989.

[Brooks *et al.*, 1988] Rodney A. Brooks, Jonathan H. Connell, and Peter Ning. Herbert: A second generation mobile robot. MIT AI Memo 1016, January 1988.

[Brooks, 1987] Rodney A. Brooks. Planning is just a way of avoiding figuring out what to do next. Technical Report 303, Artificial Intelligence Laboratory, MIT, September 1987.

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

[Chien, 1989] Steve A. Chien. Using and refining simplifications: Explanation-based learning of plans in intractable domains. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 590–595, 1989.

[Clark, 1978] Kenneth. L. Clark. Negation as failure. In Hervé Gallaire and Jack Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, New York, 1978.

[Cohen and Levesque, 1990] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(2–3):213–261, 1990.

[Dean and Boddy, 1988] Thomas Dean and Mark Boddy. An analysis of time-dependent planning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 49–54, August 1988.

[Elkan, 1989] Charles Elkan. Conspiracy numbers and caching for searching and/or trees and theorem-proving. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 341–346, August 1989.

[Feldman and Rich, 1986] Yishai A. Feldman and Charles Rich. Reasoning with simplifying assumptions: A methodology and example. In *Proceedings of the National Conference on Artificial Intelligence*, pages 2–7, August 1986.

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

[Foo *et al.*, 1988] Norman Y. Foo, Anand S. Rao, Andrew Taylor, and Adrian Walker. Deduced relevant types and constructive negation. In Kenneth Bowen and Robert Kowalski, editors, *Fifth International Conference and Symposium on Logic Programming*, volume 1, pages 126–139, Seattle, Washington, August 1988. MIT Press.

[Ginsberg, 1990] Matthew L. Ginsberg. Defaults and hierarchical problem solving. In *Preprints of the Third International Workshop on Nonmonotonic Reasoning*, Lake Tahoe, May/June 1990.

[Hanks and McDermott, 1986] Steve Hanks and Drew McDermott. Default reasoning, nonmonotonic logics, and the frame problem. In *Proceedings of the National Conference on Artificial Intelligence*, pages 328–333, August 1986.

[Knoblock, 1989] Craig A. Knoblock. Learning hierarchies of abstraction spaces. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 241–245. Morgan Kaufmann Publishers, Inc., 1989.

[Konolige, 1986] Kurt Konolige. *A Deduction Model of Belief.* Pitman, 1986.

[Korf, 1987] Richard E. Korf. Real-time path planning. In *Proceedings of the DARPA Knowledge-Based Planning Workshop*, 1987.

[Lifschitz and Rabinov, 1989] Vladimir Lifschitz and Arkady Rabinov. Things that change by themselves. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 864–867, 1989.

[Mills, 1989] Jonathan W. Mills. A pipelined architecture for logic programming with a complex but single-cycle instruction set. In *Proceedings of the IEEE First International Tools for AI Workshop*, September 1989.

[Moore, 1985] Robert C. Moore. A formal theory of knowledge and action. In *Formal Theories of the Commonsense World.* Ablex, 1985.

[Naish, 1986] Lee Naish. *Negation and Control in* PROLOG. Number 238 in Lecture Notes in Computer Science. Springer Verlag, 1986.

[Przymusiński, 1987] Teodor C. Przymusiński. On the declarative semantics of stratified deductive databases and logic programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216, Los Altos, California, 1987. Morgan Kaufmann Publishers, Inc.

[Shanahan, 1989] Murray Shanahan. Prediction is deduction but explanation is abduction. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1055–1060, 1989.

[Stickel and Tyson, 1985] Mark E. Stickel and W. M. Tyson. An analysis of consecutively bounded depth-first search with applications in automated deduction. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 1073–1075, August 1985.