From AAAI Technical Report SS-96-04. Compilation copyright © 1996, AAAI (www.aaai.org). All rights reserved.

What is planning in the presence of sensing? (position paper version)

Hector J. Levesque
Department of Computer Science
University of Toronto
Toronto, ON, M5S 3H5 Canada
hector@cs.toronto.edu

This paper is a very abridged version of one submitted to the 1996 National Conference of the AAAI [8].

Much of high-level symbolic AI research has been concerned with planning: specifying the behaviour of intelligent agents by providing goals to be achieved or maintained. In the simplest case, the output of a planner is a sequence of actions to be performed by the agent. However, a number of researchers are investigating the topic of *conditional planning* where the output, for one reason or another, is not expected to be a fixed sequence of actions, but a more general specification involving conditionals and iteration. Surprisingly, despite the existence of conditional planners, there has yet to emerge a clear and general specification of what it is that these planners are looking for: what is a plan in this setting, and when is it correct?

There are a number of ways of making the planning task precise, but perhaps the most appealing is to put aside all algorithmic concerns, and come up with a specification in terms of a general theory of action. In the absence of sensing actions, one candidate language for formulating such a theory is the situation calculus [9]. We will not go over the language here except to note the following components: there is a special constant S_0 used to denote the *initial situation*, namely that situation in which no actions have yet occurred; there is a distinguished binary function symbol *do* where $do(a, s)$ denotes the successor situation to s resulting from performing the action a; relations whose truth values vary from situation to situation, are called (relational) *fluents*, and are denoted by predicate symbols taking a situation term as their last argument; finally, there is a special predicate $Poss(a, s)$ used to state that action a is executable in situation s.

Within this language, we can formulate domain theories which describe how the world changes as the result of the available actions. One possibility is a theory of the following form [11]:

- Axioms describing the initial situation, S_0.
- Action precondition axioms, one for each primitive action a, characterizing $Poss(a, s)$.
- Successor state axioms, one for each fluent F, stating under what conditions $F(\vec{x}, do(a, s))$ holds as function of what holds in situation s. These take the place of so-called effect axioms, but provide a solution to the frame problem [11].
- Unique names axioms for the primitive actions.
- Some foundational, domain independent axioms.

For any domain theory of this sort, we have a simple but general specification of the planning task (in the absence of sensing actions), which dates back to the work of Green [3]:

Classical Planning: Given a domain theory *Axioms* and a goal formula $\varphi(s)$ with a single free-variable s, the planning task is to find a sequence of actions[1] $\vec{a}$ such that

$$Axioms \models Legal(\vec{a}, S_0) \wedge \phi(do(\vec{a}, S_0))$$

where $do([a_1, \ldots, a_n], s)$ is an abbreviation for

$$do(a_n, do(a_{n-1}, \ldots, do(a_1, s) \ldots)),$$

and where $Legal([a_1, \ldots, a_n], s)$ stands for

$$Poss(a_1, s) \wedge \ldots \wedge Poss(a_n, do([a_1, \ldots, a_{n-1}], s)).$$

In other words, the task is to find a sequence of actions that is executable (each action is executed in a context where its precondition is satisfied) and that achieves the goal (the goal formula φ holds in the final state that results from performing the actions in sequence).[2] A planner is *sound* if any sequence of actions it returns satisfies the constraints; it is *complete* if it is able to find such a sequence when one exists.

Of course in real applications, for efficiency reasons, we may need to move away from the full generality of this specification. In some circumstances, we may settle for a sound but incomplete planner. We may also impose constraints on what what sorts of domain theories or goals are allowed. For example, we might insist that S_0 be described by just a finite set of atomic formulas and a closed world assumption, or that the effect of executable actions not depend on the context of execution, as in most STRIPS-like systems.

[1] To be precise, what we need here (and similarly below for robot programs) are not actions, but ground terms of the action sort that contain no terms of the situation sort.

[2] This definition is easily augmented to accommodate maintenance goals, conditions that must remain true throughout the execution.

However, it is clearly useful to understand these moves in terms of a specification that is unrelated to the limitations of any algorithm or data structure. Note, in particular, that the above account assumes nothing about the form of the preconditions or effects of actions, uses none of the terminology of STRIPS (add or delete lists *etc.*), and none of the terminology of "partial order planning" (threats, protecting links *etc.*). It is neutral but perfectly compatible with a wide range of planners. Indeed the STRIPS representation can be viewed as an *implementation strategy* for a class of planning tasks of this form [4].

This then raises an interesting question. In classical planning, it is assumed that what conditions hold or do not hold at any point in the plan is logically determined by the background domain theory. This workshop, however, is concerned with domains where, because of incomplete knowledge of the initial situation, or the presence of exogenous events, or uncertainty about the effects of actions, it becomes necessary to use *sensing actions* to determine at runtime whether certain fluents hold. Is there a specification of the planning task in such domains which, once again, is neutral with respect to the choice of algorithm and data structure?

The closest candidate that I could find is that of Etzioni *et al.* [2]. Unfortunately, as a specification, their account has a number of drawbacks. For one thing, it is formulated as a rather complex refinement of the STRIPS account. It deals only with atomic conditions or their negations, assumes that we will be able to "match" the effects of actions with goals to be achieved, and so on. There are also other representational limitations: it does not allow preconditions on sensing actions, and does not handle iteration. While limitations like these may be perfectly reasonable and even necessary when it comes to formulating efficient planning procedures, they tend to obscure the logic behind the procedure. Why must postconditions for ordinary actions not contain runtime variables? What would break if postconditions for sensing actions had more than one runtime variable? Just why is it that we need a value to be a constant at one point, while it is fine for it to be a variable at another? and so on.

Instead of building on a STRIPS-like definition of planning, we might again try to formulate a specification of the planning task in terms of a general theory of action, but this time including sensing actions and the effect they have on the knowledge of the agent or robot executing them.

As it turns out a theory of this sort already exists. Building on the work of Moore [10], Scherl and Levesque have provided a theory of sensing actions in the situation calculus [12]. Briefly, what they propose is a new fluent K, whose first argument is also a situation: informally, $K(s', s)$ holds when the agent in situation s, unsure of what situation it is in, thinks it could very well be in situation s'. Since different fluents hold in different situations, the agent is also implicitly thinking about what could be true. Knowledge for the agent, then, is what *must* be true because it holds in all of these so-called accessible situations: **Know**(ϕ, s) is an abbreviation for the formula $\forall s'[K(s', s) \supset \phi(s')]$. Beyond this encoding of traditional modal logic into the situation calculus, Scherl and Levesque provide a successor state axiom for K, that is an axiom which describes for any action (ordinary or sensing) the knowledge of the agent after the action as a function of its knowledge and other conditions before the action. Thus, Reiter's solution to the frame problem is extended to the K fluent.

While the above theory accounts for the relationship between knowledge and action, it does not allow us to use the classical definition of a plan. This is because, in general, there is no *sequence of actions* that can be shown to achieve a desired goal; typically, what actions are required depends on the runtime results of earlier sensing actions.

It is tempting to amend the classical definition of planning to say that the task is now to find a *program* (which may contain conditionals or loops) that achieves the goal, a sequence of actions being simply a special case. But saying we need a program is not enough. We need a program that does not contain conditions whose truth value is unknown to the agent at the required time: that is, the agent needs to *know how* to execute the program. One possibility is to develop an account of what it means to know how to execute an arbitrary program, for example, as we did in [5], and as done by Davis in [1]. While this approach is certainly workable, it does lead to some complications. There may be programs that the agent knows how to execute in this sense but that we do not want to consider as plans.[3] Here, we make a much simpler proposal: invent a programming language $\mathcal{R}$ whose programs include both ordinary and sensing actions, and which are all so clearly executable that an agent will trivially know how to do so.

We will not present such a programming language here (see [8, 7]). But assume that we have done so, and that for any such program r, that we have a formula $Rdo(r, s, s')$ which means that r terminates legally when started in s, and s' is the final situation.[4] Then we propose a definition of the planning task as follows:

> **Revised Planning:** Given a domain theory *Axioms* and a goal formula $\phi(s)$ with a single free-variable s, the planning task is to find a robot program r in the language $\mathcal{R}$ such that:
>
> $$Axioms \models \forall s.K(s, S_0) \supset \exists s'[Rdo(r, s, s') \wedge \phi(s')]$$
>
> where *Axioms* can be similar to what it was, but now covering sensing actions and the K fluent.

To paraphrase: we are looking for a robot program r such that it is known in the initial situation that the program will terminate in a goal state.[5] Note that we are requiring that

[3] Consider, for example, the program that says (the equivalent of) "find a plan and then execute it." While this program is easy enough to generate, figuring out how to execute it sensibly is as hard as the original planning problem.

[4] Although programs in $\mathcal{R}$ ought to be deterministic and trivial to execute, some may loop forever.

[5] We are requiring the agent to *know how* to achieve the goal, in that the desired r must be known initially to achieve ϕ. A variant would require an r that achieved ϕ starting in S_0, but perhaps un-

the program lead to a goal state starting in any s such that $K(s, S_0)$; in different s, doing r may require very different sequences of actions.

In a sense, this revised specification of the planning problem also provides us with an initial (highly inefficient) implementation:

Planning Procedure (ϕ)
  repeat with $r \in \mathcal{R}$
    if $Axioms \models \forall s.K(s, S_0) \supset \exists s'[Rdo(r, s, s') \wedge \phi(s')]$
      then return r
    end if
  end repeat
end Procedure

We can also think of the r as being returned by answer extraction [3] from an attempt to prove the following:

$$Axioms \models \exists r\ \forall s.K(s, S_0) \supset \exists s'[Rdo(r, s, s') \wedge \phi(s')]$$

Either way, the procedure would be problematic: we are searching blindly through the space of all possible robot programs, and for each one, the constraint to check involves using the K fluent explicitly as well as the (likely second-order!) *Rdo* formula.

However, we do not want to suggest that a specification of the planning task ought to be used this way to generate plans. Indeed, our criticism of earlier accounts was precisely that they were too concerned with specific planning procedures. In fact, in our own work, we have decided for a number of reasons to forego planning entirely in favor of high-level program execution. Given that these high-level programs can involve considerable nondeterminism, discovering how to execute them shares many features with planning. However, in many cases, the high-level programs can be handled much more efficiently as they provide very strong hints about their execution; in extreme cases, the high-level programs are fully deterministic, and no searching at all is required. The interested reader should consult [6, 7] for details.

References

[1] Ernest Davis. Knowledge preconditions for plans. Technical Report 637, Computer Science Department, New York University, 1993.

[2] Oren Etzioni, Steve Hanks, Daniel Weld, Denise Draper, Neal Lesh, and Mike Williamson. An approach to planning with incomplete information. In Bernhard Nebel, Charles Rich, and William Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 115–125, Cambridge, MA, 1992. Morgan Kaufmann Publishing.

[3] Cordell C. Green. Theorem proving by resolution as a basis for question-answering systems. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 183–205. American Elsevier, New York, 1969.

[4] Fangzhen Lin and Raymond Reiter. How to progress a database II: The STRIPS connection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 2001–2007, Montreal, Aug. 20-25, 1995.

[5] Yves Lespérance, Hector J. Levesque, Fangzhen Lin, and Richard B. Scherl. Ability and knowing how in the situation calculus. In preparation, 1995.

[6] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. submitted to the *Journal of Logic Programming*, 1995.

[7] Hector J. Levesque. The execution of high-level robot programs with sensing actions: theory and implementation. In preparation, 1995.

[8] Hector J. Levesque. What is planning in the presence of sensing? Submitted to AAAI 96.

[9] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[10] R. C. Moore. A formal theory of knowledge and action. In J. R. Hobbs and R. C. Moore, editors, *Formal theories of the common sense world*, pages 319–358. Ablex Publishing, Norwood, NJ, 1985.

[11] 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, San Diego, CA, 1991.

[12] Richard B. Scherl and Hector J. Levesque. The frame problem and knowledge-producing actions. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 689–695, Washington, DC, July 1993. AAAI Press/The MIT Press.

beknownst to the agent. A third variant might require not merely ϕ, but that the agent know that ϕ. Space precludes discussing these variants here.