STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving - Richard E. Fikes, Nils J. Nilsson

(from the paper)
The paper describes a problem solver called STRIPS that finds a sequence of operators in a space of world models to transform the initial state model to the into goal state model. STRIPS represents a world model as an arbitrary collection of first-order predicate calculus formulas and is designed to work with models consisting of large number of formulas. It employs a resolution theorem prover to answer questions of particular models and use mean-ends analysis to guide it to desired goal-satisfying model.

The motivation for the system is to have a better mechanism to control the robot. It should be capable of 1) Performing tasks like moving stuff around or navigating around an environment 2) solving problems in real-time efficiently on the very limited hardware of the time.

At the time, this paper made two major contributions to the advance of AI planning: 1) A representation language for planning problems, which have a set of preconditions and side effects. 2) The use of two different search algorithms for the solutions. The first for doing the planning itself which figures out what should be done next on a high-level, and the second for working out what is currently true in the world if the goals have been satisfied.

In addition, in order to make the system more efficient, it uses a strategy known as means-end. This works by looking at what needs to change in the world model, and selecting an appropriate operator to perform the task.

Approximate Planning - Matthew L. Ginsberg

The planning system that we've learned in the class are exact, meaning it is both sound -- all (intermediate states) plans are correct and complete -- the entire state is computed at each generation of the planning graph. This model is reliable in that as long as we have a finite set of preconditions and actions.. However, this model has constraints: 1) we live in a messy world where oftentimes it's impossible to map every single conditions and actions 2) even if it's possible to do that for some domains, there often exists "edge cases (conditions)" to the problem where the chance of running into these case are relatively rare. However mapping all the edge cases in the planning inevitable increases the branching factor for the planning graph exponentially. As a result, edge cases waste a lot of resources and time when searching for a plan.

In this paper, the author argues that plan should be approximately sound and complete, in that *most* plans returned achieve the goal and that *most* such plans are returned. The concept is that 1) when in the process of looking for a plan, acknowledge the existence of edge cases, but not dig into any specific example of them, and 2) search backwards for the solution from the goal.

The regression search ensure that only the edge cases that matter will be examined, which further reduces the unnecessary fan-out.

The search process works like this: it starts with the goal state, then next step (#2) it iterates through all *types* of edge cases that prevent the plan to reach the goal, and uses "?" to *categorize* the exceptions. For example: we have three blocks as illustrated below:

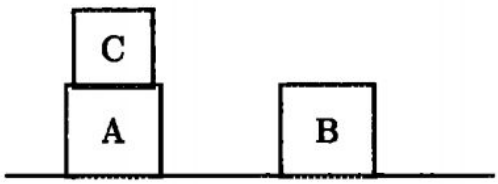

Figure 2: The Sussman anomaly

We want to move C on top of B On(C, B). The corresponding action is move(C, B). There are three edge cases that would block the plan from reaching the goal: 1) Something is moved on top of B [move(?, B) .. move(B, C)]. 2) Something is moved on top of C [move(?, C) .. move(B, C)]. 3) C is moved away after we reach the goal [move(B, C), move(C, ?)]. So you see we effective limit the branch-out to 3 by using "?" without digging into what "something" really is. After identifying all edge cases, we trying to find a union to categorize all three edge cases into one. Next (#3), we list all these edge cases that achieve the goal after all, and so on. The sequence will converges shortly on the set of all plans that achieve of the goal.

In addition, this paper also proved that approximate planning also works with conjunction planning in that two approximate plans g1 and g2 is able to produce a plan for the conjoined goal g1 A g2.

[An Algorithm for Probabilistic Least-Commitment Planning - Nicholas Kushmerick, Steve Hanks, Daniel Weld]

This paper comes up a probabilistic semantics for planning under uncertainty, and presents a fully implemented algorithm that generates plans that succeed with probability no less than a user-supplied probability threshold. With the same motivation as the previous paper: In real world, it's hard to have complete and correct information about the state of the work and about the initial state of its actions. When applying the classical planning algorithm to solve real-world problems, one has to make certainty assumptions, for example, when we say Fly(P1, SFO, JFK) -> At(P1, JFK), we assume that P1 will definitely show up in JFK no matter what. However, it's not always the case, under severe weather conditions, although rare, P1 will have to go back to SFO. When that happens (let's say on average once every year), the plan fails to execute. In this case, should we regard the plan as an invalid plan? No! Because out of 364 days in a year, the plan works as it promised, however on that bad day, it is indeed an invalid plan from the system based on the definition.

This paper recognize the limitation like this and it assigns a probability to outcome (consequence) of each action from the previous state. An action can have a set of consequences {(ta, pa, ea), (tb, pb, eb)} where t is the trigger, p is the probability, and e is the effects. When applying to the example above, Fly(P1, SFO, JFK) will lead to two consequences: {(At(P1, SFO), 0.99, At(P1, JFK)), (At(P1, SFO), 0.01, At(P1, SFO)}, and very likely the second consequence will triggers another Fly action which takes P1 to JFK. Eventually, we say as long as the probability of At(P1, JFK) is >= 0.99, it's a valid plan.