The Intentional Planning System: ItPlanS*

Christopher W. Geib
University of Pennsylvania
Department of Computer and Information Science
200 S. 33rd Street
Philadelphia, PA 19104
Email: geib@linc.cis.upenn.edu

Abstract

This paper describes the Intentional Planning System(ItPlanS) an incremental, hierarchical planner that uses a series of experts to develop plans. This system takes seriously the concept of the context sensitivity of actions while working in a resource bounded framework.

Introduction

Consider the action of an agent opening its hand while holding an object. Suppose that the agent performs this action in two different situations. In the first situation, the object in the agent's hand is supported by some other object. In the second situation, the object is not supported. If the agent opens its hand in the first situation, the object will be stacked on the supporting object. However, if the agent performs the action in the second situation, the object will fall and possibly break.

This example highlights the effect that context can have on action. In many planning systems (McAllester & Rosenblitt 1991; Sacerdoti 1974; Tate 1977), in order to use the hand opening action for both of these effects, the system would need to have two separate actions, one for stacking and one for dropping. But creating more actions, will increase the branching factor at each node of the planner's search space. Many planners will have to consider each of the actions, increasing their runtime. This solution also seems to conflict with our intuitions that there is something the same about the stacking action and the dropping action that is not being captured.

It is therefore surprising, that until recently the building of "different" actions to accomplish different effects has been seen as a legitimate solution to the dilemma of conditional effects. This was a result of the fact that early planning systems did not distinguish between an action and its effects. In fact, any planning system with traditional STRIPS style action representations (Fikes & Nilsson 1971; Nilsson 1980), has this problem.

This problem with the traditional STRIPS style representation of actions is not a new observation. In introducing conditional effects to actions, researchers (Pednault 1987; Schoppers 1987) have acknowledged that context is relevant to determining the effects of an action, and accepted that actions can have wildly differing effects depending on the context of their execution. However their solutions to the problem have been inadequate.

One solution (Pednault 1987) has been to add "secondary" preconditions to capture the "conditional" effects that actions have. Unfortunately this solution is little more than a theoretically sound version of creating separate actions for different effects. Thus, this kind of frame work gives a formal language to talk about conditional effects of actions but does not provide a practical solution for how to plan using them. It also rejects the fundamental realization that all of an actions effects are context dependent.

Other researchers (Agre 1988; Chapman 1987; Schoppers 1987) have solved the problems of the context sensitivity of action effects by considering all possible world states and deciding before hand the "correct action." While these systems do work, once built they lack flexibility since the "correct action" cannot be changed at runtime. They have traded off-line time and flexibility for context sensitivity and reaction time. For some applications this exchange is unacceptable.

This paper will discuss the intentional planning system (ItPlanS), a planner that takes the context sensitivity of the effects of action seriously in the light of bounded rational agency without the failings of either of the two previous mentioned solutions. Three issues are central to the design of ItPlanS: (1) building default plans by using preference information given by intentions, (2) using situated reasoning about the results of actions to predict the effects of action, and (3) using experts to patch and optimize plans. This paper will discuss how each of these ideas plays a role in ItPlanS, and then discuss how ItPlanS provides a solution to the problems of context dependent effects of actions.

*This research is supported by ARO grant no. DAAL03.89.C.0031 PRIME

Examples used in this paper come from the first domain that ItPlanS has been used in: a "two-handed" blocks world.

ItPlanS Overview

ItPlanS is an incremental, hierarchical planner similar to McDermott's (McDermott 1978) NASL system and Georgeff's (Georgeff & Lansky 1987) PRS. ItPlanS produces its plans by performing an incremental, left-to-right, depth-first expansion of a set of intentions. This process is incremental in that the system only expands its intentions to the point where it knows the next action that should be performed. ItPlanS then evaluates the initial plan to verify that in fact it will achieve the agent's intentions and patches any problems that it finds. In the final phase of planning experts are called to suggest improvements to the existing partial plan. When this optimization phase is complete, the action is performed and ItPlanS resumes the expansion process to determine the next action. This incremental expansion-verification-patching-optimization-execution loop is continued until all of the system's intentions are achieved. The following four sections of the paper will examine in detail each of these processes in detail.

It is reasonable to ask at this point, why not build complete plans? Simply put, the answer is speed and resource use. ItPlanS was designed to work in environments where an agent may have to act on partial knowledge, where it does not have sufficient knowledge of the world or the effects of its actions on the world to build a complete plan. In these situations an agent must plan forward from the current world state, initially expanding those actions that it plans on taking first.

Intention Expansion

ItPlanS distinguishes between two kinds of intentions: positive and negative. Positive intentions represent those actions that the agent is committed to performing or those states that the agent is committed to performing actions to bring about. Negative intentions, in contrast, represent those actions or states that the agent has a commitment not to bring about. For example, the system might have a negative intention toward breaking objects. In the rest of this paper "intentions" will be used to refer to positive intentions except were explicitly stated otherwise or obvious from context.

In ItPlanS, intentions are more than just a representation of the system's commitments to act, they also order the actions that the system considers in fulfilling its commitments. For example, if an agent has an intention to stack a block on top of another block, the agent shouldn't start the process of building a plan to achieve this by considering the action of jumping up and down. In general, there are a small number of actions that under "normal" conditions will result in the stacking of one block on top of another. Resource bounded systems must give preference to these actions in their planning deliberations.

ItPlanS uses this kind of preference information to build an initial plan. Each positive intention in the system provides a preference ordering for the actions that can be used to achieve it. Notice that, this kind of preference ordering should be sensitive to conditions that change in the environment. However, since the determination of the conditions that would cause such changes are beyond the scope of this research, the orderings provided by intentions in ItPlanS are fixed.

ItPlanS initially expands an intention by using the first action in the preference ordering. Having decided on an action to achieve the given intention the system adds this action and expansion to the plan and continues the expansion process by considering the leftmost sub-intention of the expansion. This process continues until a "basic" or primitive action is reached. Thus, the system rapidly develops a default plan of action rather than examining all of the possible actions to determine an initial plan that is guaranteed to work.

For example, a left-handed agent would in general prefer to pick up objects with its left hand over its right. Thus the default plan for picking up objects calls for the use of the left hand. Obviously this plan will be suboptimal if the left hand is in use. However, ItPlanS builds this initial plan and leaves the correction of the hand choice to later processing, preferring instead to develop an initial plan as rapidly as possible.

Verification of the Initial Plan

Having selected default action expansions, the system has no guarantee that the plan developed will result in the achievement of the system's intentions. To verify this, the system performs a *limited, situated simulation* of the actions at each level of decomposition in the plan. This simulation determines (1) if the action will satisfy the positive intention given the current world state and (2) if there are problems created by the action choices that have been made.

There are two important features of this simulation. First, since the planning process is incremental, this simulation takes place while the system is confronted with the actual world state in which the action will be carried out. Thus, the system doesn't need to maintain a complex world model. It has immediate access to the state in which the action will be performed, and its simulation can therefore be specific to the situation at hand. This allows the simulation to predict even those effects that are the result of the context.

Second, since this simulation is happening for each action in the decomposition the simulation can take place at different levels of abstraction. For example, consider the case of simulating the action **move(x, y)**. The simulation of the action can yield information that block **x** is over **y** without providing exact coordinates. Thus, simulation can be limited to only the "relevant" information.

This simulation provides more information than simply confirming that an action will satisfy a given intention. The simulation will also reveal conflicts with the negative intentions. For example, suppose the system has a negative intention toward actions that result in things being broken. Further, suppose the action to be simulated is **move(x,y)** and there is another block on **x**. The simulation would show that by moving **x** the block on top of it may fall thus conflicting with the negative intention. These conflicts are flagged and stored for repair by a later expert. See the next section.

The simulation also allows the system to notice when the performance of an action will invalidate a previously achieved goal. Again this condition is noted and stored for repair by an expert. Thus, the simulation phase answers three questions. First, will the action satisfy the intention? Second, will the action conflict with existing negative intentions? Finally, will the action clobber an already achieved intention?

Initial Plan Correction Experts

Having simulated the actions and stored the possible problems, ItPlanS enters the plan correction phase of its algorithm. In this phase experts are called to correct the problems that were noticed in the simulation. Each of these experts is expected to provide complete changes to the plan to solve the problem.

By changing a node in the plan, the expansion of that node becomes irrelevant to the goal. Therefore an expert is required to provide not only the changes to a single node but a new partial plan from that node down to the level of primitive actions. As a result any change suggested by an expert that would alter the previously decided upon next action to be taken, will provide a new "next action." This guarantees that the agent always has some action to take if it is called on to act.

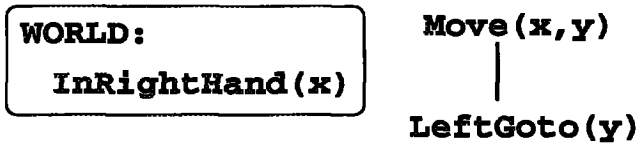

Figure 1: Simulation Failure: before

ItPlanS currently has two such "problem correcting" experts: an expert for resolving negative intention conflicts, and an expert for altering a plan when the action is found not to satisfy the given intention. The results of these experts can best be appreciated through an example. Consider Figure 1 in which the system has the intention to **move(x,y)**. The system's default plan for achieving this goal is simply moving its left hand to **y**. This would be a perfectly reasonable plan, assuming that **x** were in its left hand. However this is not the case.

Since the block is in the right hand, the simulation phase will notice this problem and the simulation failure expert will be called. This expert is called any time an action in the plan is simulated and found to not satisfy its intention. This expert first looks for other possible methods of achieving the intention by simply searching along the preference ordering list that was used to select the action. In this case, the next action the system has in its preference ordering is moving the right hand. Since this does satisfy the intention to **move(x,y)** the expert can replace the intention to execute **LeftGoto(y)** by an intention to perform a **RightGoto(y)** (Figure 2) and remove the problem from the list of simulation failures so that the next expert can be called.

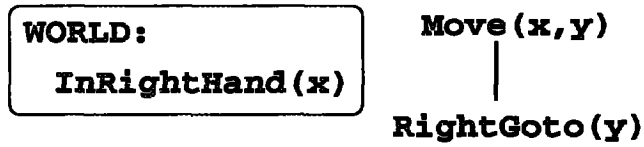

Figure 2: Simulation Failure: after

Note that, if no action were found in the preference ordering that would have satisfied the intention, then the expert would appeal to a difference minimization strategy to identify what was preventing the original action from satisfying the intention. Having identified the needed conditions, the expert would have added the achievement of these conditions to the intention structure.

The negative intention conflict resolution expert works in a similar manner, preferring to find another way to achieve a given intention over changing the world to allow the action to succeed. All of the processing that has been described so far has been designed to produce plans that achieve a desired intention. The next section discusses optimizing these plans so that the intentions are achieved efficiently.

Optimizing the Expansion

Consider an agent with two intentions, to pick up an object and to throw it into the garbage. The agent is capable of picking up the object with either hand. Suppose it chooses its left. Suppose further, that the object is directly in front of the agent but the garbage can is to the agent's right. Clearly, it would be easier for the agent to throw the object into the garbage can if it were picked up with the right hand. However, there is no way for a planning system to make this hand choice determination without "looking-ahead" to the next intention.

It should be clear that the plan expansion and patching described so far only take into account the intention to be expanded or satisfied and the state of the world. Thus, the initial plan produced by the system does not consider possible optimizations that could be made on

the basis of future intentions. These kinds of changes are the prerogative of the optimizing experts.

Optimizing Experts

This work has identified and implemented three such optimizing experts. The first is an expert in hand selection, while the other two are forms of what Pollack (Pollack 1991) has termed *intentional overloading*. However, before discussing specific optimizing experts, a more detailed description of the operation of these optimizers will be given.

When an optimizing expert is called, it is given access to the system's positive intentions, negative intentions, the world state, and the memory of previously taken actions. Thus, each expert has access to all of the information that the initial planning algorithm had. Notice that the changes that an optimizer suggests may make commitments about the future. For some optimizations to have their desired effect, not only must the current action be performed in a particular manner, but *the system must also commit to performing other actions in the future*. For an example of this, see the section on implemented optimizing experts. After each expert finishes, the changes it suggests are made. The expert may then be called again to suggest further changes or a different expert may be called.

Bounding Look-Ahead in Optimizers

The optimizers implemented in ItPlanS only look "one step ahead." That is, while the expert is provided with all of the system's positive intentions, at each level of an "intention tree" they only consider the intention being expanded and that intention's next right-sibling in making its decisions. This limitation is not as confining as it might appear. Since intentions are hierarchical, the "one-step look-ahead" encompasses more and more of the system's intentions as one moves up through the intention structure.

Still, one might argue that experts could easily be designed with a two, three or more step horizon. However, building experts with an unbound look-ahead would allow ItPlanS to develop a complete plan for the achievement of its intentions before engaging in action in the world. This is unacceptable. If ItPlanS is to be incremental, the look-ahead of experts must be bounded. This limit has been set at one, not because the only productive optimizations will be found within this bound, but rather because it is a simple place to start looking for optimization techniques.

Implemented Optimizing Experts

The simplest of the optimizers implemented so far uses no look-ahead; it only looks at the current plan and previously achieved goals. This expert makes changes to decisions about hand choice made in the initial plan. For example, in Figure 3, the system has an intention to get block x (**Get(x)**) which it has already satisfied, and an intention to get block y (**Get(y)**) which it is in the process of satisfying. As before, the default plan

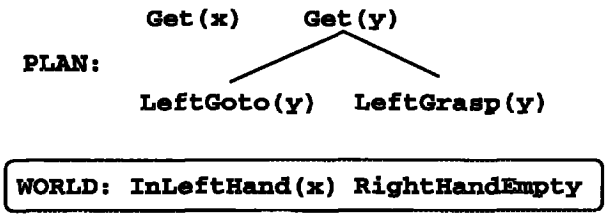

Figure 3: Hand Choice Optimization: before

for getting **y** involves the use of the left hand.

When simulating **Get(y)**, ItPlanS recognizes that the action will have to violate **Get(x)**, and flags this. If the violation is not fixed, ItPlanS would build a plan to empty the left hand in order to get **y**. However, since the hand choice optimizer is called before this happens, it recognizes this as a situation in which the use of the right hand will simplify the overall plan. Therefore, it suggests the change to the plan shown in Figure 4.

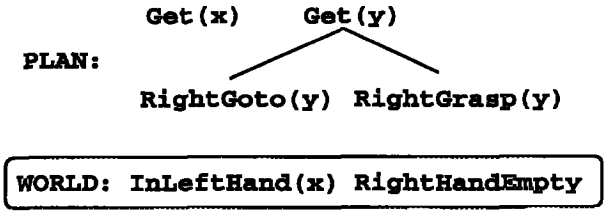

Figure 4: Hand Choice Optimization: after

Pollack uses the term *intentional overloading* to describe the use of a single intention to fulfill multiple parts of a plan. In ItPlanS, this concept is broken into

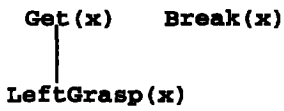

Figure 5: Overloading Example: initial

two functional cases. First, the same single intention can occur in multiple places in a plan such that the achievement of both occurrences can be accomplished by a single action. For example, in Figure 5 the system is given the intentions to get an object and then to break it. The intention of getting an object can be achieved by two different actions: grasping the object in the right hand or grasping the object in the left hand. Suppose initially the system decides on the left hand. Now suppose that the action of breaking an object involves placing the object in the agent's right hand and crushing it. By calling the intention overloading expert, the system can recognize that grasping the object is the only step in "getting" it and the first step in breaking it. Thus, the two intentions to grasp

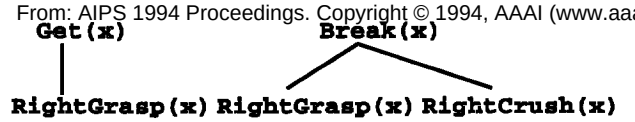

Figure 6: Overloading Example: optimized

the given object can be collapsed into a single intention if the system initially picks up the object with the right hand (Figure 6). Notice that the change suggested by the expert does expand the intention to break the object.

The second type of overloading is where the action that is chosen to fulfill a given intention has the effect of satisfying another intention in the plan. For example the action of stacking a block on another block also satisfies the intention of emptying the hand. In the first case the same intention exists in two places in the agent's intention structure in such a way that they can both be achieved at the same time. In the second case, the effects of satisfying one intention also satisfy a second intention. In one case ItPlanS looks for multiple occurrences of the same intention and in the other it compares the effects of various actions with other future intentions.

The Anytime Implementation

ItPlanS is currently implemented in LISP as an anytime algorithm and has worked in two problem domains: a two handed blocks world and in the SodaJack project (Geib, Levison, & Moore 1994) as a high-level planner for an animated Soda-Jerk.

This algorithm has been implemented in an anytime format in the following manner. First the default plan is created and the simulation/verification process is run once, experts are then run in an anytime format with each one contributing an improvement to the plan. The plan verification process is run between the experts to guarantee that the plan will work and that any problems with the suggested changes are noticed. Thus, while the experts are improving the plan a base plan has been established which the experts can improve. However, anytime after the default plan is created the system can be halted and will have an action that can be performed.

The existence of the list of problems created by the simulation phase of the algorithm suggests a "final expert" that might be added to the algorithm. When the algorithm is terminated, a final expert could be invoked to check the list of known problems with the current plan of action. If one or more of these problems were significant, the "final expert" would have the option of executing the action called for by the plan or doing nothing and waiting for the next opportunity to act. Such an expert has not yet been added to the system.

Discussion

In some sense, ItPlanS finesses the entire issue of context sensitive effects of action. Since the initial intention expansion choices are not made by back chaining on the preconditions of actions or by minimizing differences between states, the problem of using an action's conditional effects does not come up. Since the preference ordering simply lists the actions in order, it is irrelevant to the algorithm if the effect that is to be achieved is a "primary" or "secondary" effect of the action.

Secondly, since the simulations that are performed to verify the plan are done while the system is faced with the actual world, the simulation can predict all of the effects of the action. This allows the examination of the possible interactions between the action and previously achieved goals and negative intentions to include the conditional effects of the action as well as the "primary" effects.

Related Work

In this section I will try to point out some of the important places where I have borrowed others ideas and where this system diverges from those ideas. Clearly ItPlanS takes a number of ideas with a number of systems. For example, It shares the concept of planning one step ahead with McDermott's NASL (McDermott 1978), experts with Sussman (Sussman 1975), and the absence of a continuously maintained world model is derived from Agre's (Agre 1988) conception of "leaning on the world." Its hierarchical nature is similar to Georgeff's PRS (Georgeff & Lansky 1987) and Wilkins' SIPE (Wilkins 1988) and can be contrasted with the conception of hierarchical plans presented by Sacerdoti (Sacerdoti 1974) and McAllester and Rosenblitt (McAllester & Rosenblitt 1991)

These similarities noted, ItPlanS is probably most similar in design philosophy to Simmons and Davis' GTD system (Simmons & Davis 1987). They describe a system which plans by means of a generate, test and debug algorithm which sounds remarkably like the method I have described here. There are however a number of differences between ItPlanS and GTD. First and probably most significantly ItPlanS is hierarchical and incremental in its expansion of goals. Thus, while GTD is required to simulate the performance of a linear series of actions, ItPlanS only simulates the next action at each level of decomposition. Thus GTD is required to model sequences of actions in the environment and maintain an extended world model. ItPlans, on the other hand, can examine the actual world state at each time point and answer the question, will the action that is to be performed next actually achieve its goal given the state of the world. Thus, the incremental and hierarchical expansion method of ItPlanS minimizes the cost of simulation.

Also, unlike GTD, the anytime formulation of ItPlanS gives it the flexibility to run optimization ex-

perts when enough time is present. This makes GTD more vulnerable to the order in which its plans are generated. For example, if GTD were to discover and verify a suboptimial plan, the system would be left with this plan. ItPlanS on the other hand, if given more time will be able to run optimizing experts which can improve the plan.

Pednault (Pednault 1987) also gives treatment of planning with actions that have secondary effects. While Pednault's presentation is remarkably clear and complete, the runtime of the solution he proposes makes it inappropriate for resource bounded agents (Bratman, Israel, & Pollack 1988; Dean & Boddy 1988; Pollack 1991). This raises the obvious question of the runtime of ItPlanS' algorithm. Unfortunately since there is no exhaustive list of experts and their computational complexity, it is impossible to give a runtime for the entire algorithm. However if we assume that the cost of the simulation step does not vary with the level of the action in the hierarchy (an assumption born out in this implementation), then the initial expansion process and simulation for a single cycle through the planner are both of order $log_k(n)$ where k is the average branching factor for an action and n is the number of steps in the complete expansion of the goal. This means that in the best case the algorithm requires $nlog_k(n)$. As for a worst case analysis, this would require an analysis of the experts used. Notice however, since each iteration of the default expansion phase of the algorithm introduces at most $log_k(n)$ new actions to the plan and each expert should only need to examine the new parts of the plan, it is not unreasonable to predict that experts can be designed to fall within this computational bound.

Conclusion

This paper has presented an implemented, hierarchical, incremental planning system designed to seriously consider, plan for, and use the context sensitive effects of actions. It does this by eliminating preconditions in favor of building default plans and using limited simulation to identify problems with the plan that are then corrected by experts.

Acknowledgments

The author would like to thank Bonnie Webber for supporting this work, and Norm Badler, Mike Georgeff, Martha Pollack, and Marcel Schoppers for their thoughtful comments on various drafts of this paper.

References

Agre, P. 1988. The dynamic structure of everyday life. Technical Report 1085, MIT Artificial Intelligence Laboratory.

Bratman, M.; Israel, D.; and Pollack, M. 1988. Plans and resource-bounded practical reasoning. *Computational Intelligence* 4(4):349 355.

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

Dean, T., and Boddy, M. 1988. An analysis fo time-dependent planning. In *Proceedings of AAAI.*

Fikes, R., and Nilsson, N. 1971. A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Geib, C.; Levison, L.; and Moore, M. B. 1994. Sodajack: an architecture for agents that search for and manipulate objects. Technical Report MS-CIS-94-13/LINC LAB 265, Department of Computer and Information Science, University of Pennsylvania.

Georgeff, M., and Lansky, A. 1987. Reactive reasoning and planning. In *Proceedings of AAAI.*

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proceedings of AAAI.*

McDermott, D. 1978. Planning and acting. *Cognitive Science* 2:71–109.

Nilsson, N. J. 1980. *Principles of Artificial Intelligence.* Tioga Publishing Co.

Pednault, E. P. 1987. Extending conventional planning techniques to handle actions with context-dependent effects. In *Proceedings of AAAI.*

Pollack, M. 1991. Overloading intentions for efficient practical reasoning. *Nous* 25:513–536.

Sacerdoti, E. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* 5:115–135.

Schoppers, M. 1987. Universal plans of reactive robots in unpredictable environments. In *Proceedings of IJCAI.*

Simmons, R., and Davis, R. 1987. Generate, test and debug: combining associational rules and causal models. In *Proceedings of IJCAI.*

Sussman, G. J. 1975. *A Computer Model of Skill Acquisition.* Elsevier.

Tate, A. 1977. Generating project networks. In *Proceedings of IJCAI.*

Wilkins, D. E. 1988. *Practical Planning.* Morgan Kaufmann.