# Conditional Linear Planning

**Robert P. Goldman** and **Mark S. Boddy**
Honeywell Technology Center
{ goldman | boddy }@htc.honeywell.com
Honeywell Technology Center, MN65-2200
3660 Technology Drive
Minneapolis, MN 55418

**Abstract**

In this paper we present a sound and complete linear planning algorithm which accomodates conditional actions: actions whose effects cannot be predicted with certainty. Conditional linear planning is significantly simpler than conditional non-linear planning in conception and implementation. Furthermore, the efficiency tradeoffs which favor non-linear planning do not necessarily carry over with the same force to planning with conditional actions. We have applied our conditional linear planner, PLINTH, to the problem of planning image processing actions for NASA's Earth Observing System. We discuss the extension of PLINTH to probabilistic planning.

## Introduction

Classical planning has been criticized for its reliance on a complete model of actions (Brooks 1991). Constructing an elaborate plan to achieve some set of goals makes little sense if the environment is sufficiently unpredictable that the plan is likely to fail at an early stage. There are several approaches to the problem of generating plans for use in a changing and uncertain world. These fall generally into three classes: making plans more robust in the face of changes in the environment (Firby 1987), modifying plans as new information becomes available (Krebsbach, Olawsky, & Gini 1992) and conditional planning (more precisely, planning with conditional actions): planning which takes into account the uncertain outcomes of actions.

Conditional planning provides a limited relaxation of the STRIPS assumption. In this model we do not specify exactly what updates to the world model will occur, given the preconditions of an action are satisfied (if they are not satisfied, the action sequence is not valid). Instead, for each action we specify a set of possible outcomes, each of which is a world model update (as for an unconditional STRIPS operator). Which of these outcomes actually occurs, given the action takes place, is beyond our control. Accordingly, we must plan for all (or at least the most likely) contingencies. We have explored the foundations of conditional action planning in work reported elsewhere (Goldman & Boddy 1994b).

We argue that conditional action planning is suitable for domains in which there is limited uncertainty and in which plans are constructed at a fairly high level of granularity. Planning for organizations is such a domain. We are also experimenting with the planning of image analysis operations for NASA (Boddy, Goldman, & White 1994). The latter application is akin to Etzioni's Softbot application (Etzioni & Segal 1992), a domain in which conditional action planning is being applied (Etzioni *et al.* 1992). Robot planning is probably *not* such an application, unless it can be carried out at a level of abstraction sufficiently high that much of the uncertainty can be ignored.

Warren's WARPLAN-C (Warren 1976) was the first conditional planner. WARPLAN-C was a linear planner using STRIPS rules extended to express uncertain outcomes, designed to support automatic programming. The system was purely forward-planning. Peot and Smith (1992) have developed a non-linear planner for conditional planning. Given the current prevalence and popularity of nonlinear planning, our decision to construct a linear conditional planner may require some explanation. In conventional, "classical" planning applications, non-linear planning is usually an improvement over linear planning because fewer commitments yields a smaller search space, at a relatively minimal added cost to explore each element of that search space (Minton, Bresina, & Drummond 1991). However, it is not clear that this tradeoff operates in the same way for conditional planners. When plans have multiple branches, the savings from considering fewer orderings is likely to be much less and may not repay the cost in the added complexity of individual plan expansion actions. In particular, the domain in which we have applied PLINTH is one in which subgoal interactions are minor, and thus in which a linear planner can be effectively employed. Conditional linear planning is simpler in conception as well as in implementation. In particular, our conditional linear planner can be shown to be sound and complete; we do not yet know of a sound and complete conditional nonlinear planner. Finally, the operation which is needed to properly construct branching non-linear plans — resolving clobberers through conditioning apart — is a very difficult operation to direct.

In this paper, we introduce PLINTH, a linear conditional planner loosely based on McDermott's regression planner PEDESTAL (McDermott 1991). We show that this planner is sound and complete with respect to its action representation. This planner has been implemented in Quintus Prolog, running on Sun SPARC-stations. It has been tested on Peot and Smith's "Ski World" sample problem and on a simplified model of the EOS (Earth Observing System) image processing domain.

## Action representation

We assume a variant of the STRIPS action notation (with action schemas), expanded to include conditional actions and propositions with a third truth value (unknown). This representation is essentially that developed by Peot and Smith (1992), tidied up somewhat by ourselves (Goldman & Boddy 1994b).

A domain is described by a set of atomic propositions. A particular state of this domain is described by partitioning the atomic propositions into three sets: propositions which hold, negated propositions and propositions which are unknown. Note that this representation conflates the state of the world and the planner's knowledge of the state of the world. When possible, this is a convenient way of simplifying the planner's reasoning. We discuss elsewhere conditions under which this simplification is and is not possible (Goldman & Boddy 1994b).

A simple action in this framework is a partial function from world states to world states. This function can be represented by three lists of propositions: those made true by the action, those made false, and those rendered unknown. The conditions under which an action is applicable are represented by another triple of proposition sets, the preconditions.

We enter operators into our planner as predicates of three arguments: operator name, operator preconditions and operator postconditions. The latter two arguments are lists of literals, propositions, negated propositions or unknown propositions (`unk`). As a notational convenience, we record operator schemas rather than operators.

A conditional action is one which has more than one possible outcome. Which outcome occurs cannot be predicted by the planner. This distinguishes conditional actions from actions with context-dependent effects, like those in Pednault's ADL (Pednault 1989). We discuss the relationship between conditional actions and context-dependent actions in more depth elsewhere (Goldman & Boddy 1994b).

Instead of having a single list of postconditions, a conditional action will have a list of ⟨¡outcome-label, postcondition list⟩ pairs. A sample conditional operator is given as Figure 1.

This operator describes the application of a classification algorithm (implemented on a MASPAR) to an image. In order for this operator to be applicable, we must have an image to be classified, that image must be clean (noise has been removed), that image should not already be classified and that image should not be known to be unclassifiable by this algorithm. This last precondition is necessary in order to keep the planner from repeatedly applying the same operator in the futile hope that "this time it will work." The need to encode this kind of precondition is the reason why the third truth value is necessary.

There are two possible outcomes of this operator. Either the operator will work (outcome 1), `Image` will be classified according to the given classification scheme, or the operator will fail and the planner will come to know that this algorithm is not a suitable method for classifying the image (outcome 2).

## Plan representation

Conditional operators complicate the representation of plans. Unlike a conventional linear plan, a conditional linear plan is in the form of a tree. The nodes of the the plan tree are operator instances and edges are marked with outcome labels.[1] Each conditional operator introduces a branch into the tree. From an intuitive standpoint, when the planner introduces a conditional operator, it will be introduced to achieve some effect, which is achieved by some outcome of the operator. Since this effect is not guaranteed to occur, the planner must "repair" its plan by constructing a new plan to the goal from each "unforeseen" outcome. A more detailed and more formal treatment of the plan representation is given elsewhere (Goldman & Boddy 1994b).

Figure 2 gives a conditional plan tree for a plan to classify an input image according to the USGS II classification scheme and to determine a level of confidence in this classification. A confidence level can be determined by taking the results of a classification by a trained classifier and comparing it with the results of a kmeans classifier applied to the same image. The plan in figure 2 contains branches twice, once for each of the operators available to classify images according to USGS II. This is a version of a plan generated by our conditional linear planner for the EOS domain. The has been substantially simplified to fit in this paper; the actual generated plan has 16 nodes (counting three goal nodes and a start node). Many of the additional nodes correspond to operators to change the format of various files in order to meet the preconditions of the classification operators.

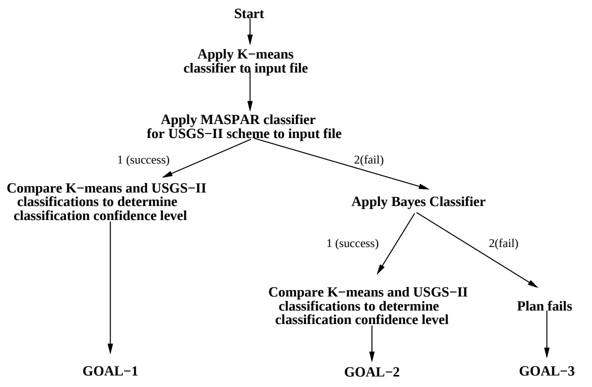
## Planning algorithm

PLINTH's conditional linear planning algorithm is non-

[1] In the interests of mathematical tidiness, think of simple operators as degenerate conditional operators with only a single outcome.

```
cond_operator(nn_maspar_classify(id:Image, class_scheme:SchemeId),
              [file_type(Image, image),
               clean(Image),
               not classified(Image, SchemeId),
               unk unclassifiable(Image, SchemeId, nnMASPAR)],
              [1-[classified(Image,SchemeId),
                  not unclassifiable(Image, SchemeId, nnMASPAR)],
               2-[unclassifiable(Image, SchemeId, nnMASPAR)]]).
```

Figure 1: A conditional operator for image processing.



Start
Apply K−means classifier to input file
Apply MASPAR classifier for USGS−II scheme to input file
1 (success)
2(fail)
Compare K−means and USGS−II classifications to determine classification confidence level
Apply Bayes Classifier
1 (success)
2(fail)
Compare K−means and USGS−II classifications to determine classification confidence level
Plan fails
GOAL−1
GOAL−2
GOAL−3

Figure 2: A conditional plan to classify an image according to the USGS II classification scheme and determine a level of confidence in the classification.

deterministic and regressive. Our development of the algorithm was inspired by McDermott's PEDESTAL and our presentation of the algorithm owes much to McDermott's paper. The algorithm maintains three important data structures: a partial plan, a set of protections and a set of as-yet-unrealized goals. The planner operates by selecting an unrealized goal and nondeterministically choosing an operator to resolve that goal while respecting existing protections. New goals may be introduced when steps are introduced, either to satisfy preconditions or to plan for contingencies introduced by conditional actions. Essentially this algorithm is identical to that of a conventional linear planner. The crucial difference is in the effect of adding a conditional action to the plan.

The three significant data structures are the partial plan tree, the protection set and the set of goals. Partially instantiated plans are represented as trees. Nodes in the tree are operator instances (steps). The root of the tree is a distinguished start node and the leaves of the tree are goal nodes. Each edge in the tree is labeled with an outcome of the operator instance at its tail.

Protections are triples, $< E, P, C >$. $E$ and $C$ are steps of the plan and $P$ is a literal. $E$ is the establisher and $C$ is the consumer. $P$ must hold from the end of $E$ to the beginning of $C$. Goals are pairs: $< P, C >$. $P$ is a literal and $C$, the consumer, is a step of the plan. In our description of the algorithm, we use the term "Plan" to refer to a triple containing the plan tree, the protection set and the goal set.

The initial state of the planner, when given a conjunctive goal to achieve $\wedge_i P_i$ is as follows:

| | |
|---|---|
| Plan Tree | start $\longrightarrow$ end |
| Protections | $\emptyset$ |
| Goals | $\{< P_0, \text{end} >, < P_1, \text{end} >, \ldots\}$ |

The planning algorithm is as follows:

```
plan(Goals, InitConds, Plan) :-
   % construct the initial plan data structure
   initial_plan(Goals, InitConds, Plan1),
   do_plan(Plan1, Plan).
```

Initially we construct a Plan data structure, `Plan1`, from the goals and the initial conditions. Recall that this data structure contains plan tree, protections and goal set. Then we plan until we reach a completed plan, `Plan`.

In the planning process there are two cases. Either there are no more goals, in which case we are done:

```
do_plan(Plan, Plan) :- plan_goals(Plan, ∅).
```

or we (non-deterministically) choose one of the goals, resolve it and continue.

```
do_plan(Plan, NewPlan) :-
  % Plan1 is Plan with Goal removed...
   pop_goal(Goal, Plan, Plan1),
   resolve_goal(Goal, Plan1, Plan2),
   do_plan(Plan2, NewPlan).
```

There are three ways a goal literal can be resolved: it may hold in the initial conditions, it may be established by some step which already exists in the plan, or it may be established by some new step:

```
% NewPlan is a partial plan derived from Plan, in
% which Goal has been achieved.
resolve_goal(Goal, Plan, NewPlan) :-
        use_ics(Goal, Plan, NewPlan).
resolve_goal(Goal, Plan, NewPlan) :-
        use_prev_step(Goal, Plan, NewPlan).
resolve_goal(Goal, Plan, NewPlan) :-
        new_step(Goal, Plan, NewPlan).
```

PLINTH may discharge a goal if that goal proposition holds in the initial conditions (clauses 1 and 2 below). In addition, there must be no step between the start of the plan and the consumer of the goal which clobbers the goal proposition (3 and 4). Finally, in order that the goal literal not be clobbered later, we add to NewPlan a protection stretching from the `start` until the goal literal is consumed (5).

```
use_ics(Goal, Plan, NewPlan) :-
 1 goal_prop(Goal,GoalP),
 2 init_conds(GoalP),
 3 goal_cons(Goal, GoalS),
 4 no_violators(GoalP, start, GoalS, Plan),
 5 add_protect(GoalP,start,GoalS, Plan, NewPlan).
```

Similarly, if there is some step in the plan that already achieves the goal (which is not clobbered by some intervening step), then the goal may be discharged. Again, PLINTH must add a protection which streches from the establishing step to the consumer.

```
use_prev_step(Goal, Plan, NewPlan) :-
   % Cons = Consumer; Est = Establisher
   goal_step(Goal, Cons),
   goal_prop(Goal, Prop),
   previous_achiever(Prop, Cons, Est, Plan),
   add_protect(Prop, Est, Cons, Plan, NewPlan).
```

Finally, PLINTH may discharge a goal through the addition of a new step which achieves that goal. The addition of a new step involves two further choices: of a position in which to insert that step (1) and of an operator, which achieves the goal, an instance of which is to be inserted (2). The step to be added must honor the existing protections (3). Finally, we must add to the Plan data structure a protection of the goal literal (4) and add as goals the preconditions of the newly-added operator (5).

```
new_step(Goal, Plan, NewPlan) :-
   goal_prop(Goal, GoalP),
   goal_step(Goal, Consumer),
 1 insert_point(Consumer, GoalP, Point, Plan),
 2 op_achieves(GoalP, Op),
 3 protections_honored(Op, Point, Plan),
   add_act(Op, Point, Plan, Plan1, Step),
 4 add_protect(GoalP, Step, Consumer, Plan1, Plan2),
 5 add_preconds(Step, Plan2, NewPlan).
```

Thus far, our description is simply that of a linear planner without any conditional actions. The essential difference arises when adding to the plan a conditional action. When adding a conditional action, $A$, there will be some outcome, $O$, such that $A-O$ will establish the goal literal (otherwise $A$ would not have been chosen for insertion). This outcome will establish what one can think of as the "main line" of the plan. However, there will also be some set of alternative outcomes, $\{O_i\}$. In order to derive a plan which is guaranteed to achieve the goal, one must find a set of actions which can be added to the plan such that the goals are achieved after $A-O_i$. for all $i$. The following predicate will be invoked (by `add_act`) when adding a conditional action:

```
add_other_outcome(OutC, Op, Branch, Pl, NewPl) :-
   % Create a new goal step
   new_goal(Branch, Pl, Pl1, NewGoal),
   % connect the Outcome with the new goal step
   add_act1(Op-OutC,NewGoalGoal-Branch,Pl1,NewPl).
```

Note that actions to deal with alternative outcomes may be added either before or after the relevant conditional action. Loosely speaking, we can add to our

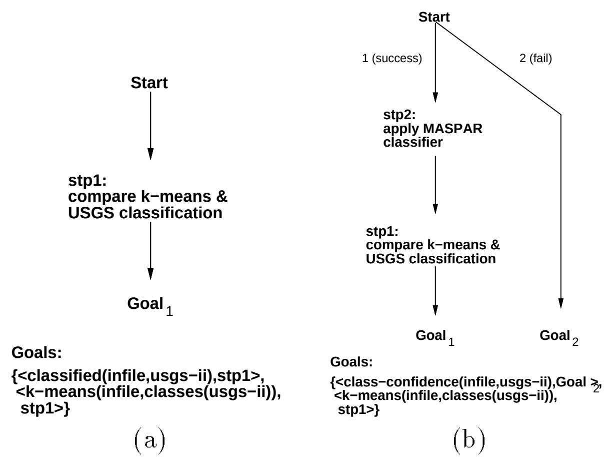


Figure 3: (a) In order to determine a USGS-II classification and find a confidence measure, one compares the results of classifying according to the USGS-II scheme with k-means classifying with the same number of classes. This snapshot of the planner's state shows what the plan looks like before adding the MASPAR classification action. (b) The state of the planner after adding the conditional action of applying the MASPAR classifier.

conditional plans either remedial actions or precautionary actions. For example, if I make a plan to drive a road which may be snowy, I can either bring chains as a precaution or plan to return to my house and get chains if I find the road to be snowy when I get to it.

We establish the requirement for these additional actions by adding a new subtree for every alternative branch. This new subtree initially contains only the conditional step and a new goal node. For each original goal conjunct, we also add a new goal to the goal set with the conjunct as literal and the new goal node as consumer. Figure 3 revisits the example of figure 2 and shows what happens when the conditional action of MASPAR classifying the image `infile` is added to the plan.

**Theorem 1** *The* PLINTH *algorithm is sound.*

**Proof:** The PLINTH algorithm generates plans which satisfy all goals. Assume the contradiction: there exists some plan, constructed according to the PLINTH algorithm in which some goal is not satisfied. Either (a) the goal was introduced but never discharged or (b) the goal was introduced and discharged, but is clobbered. Case (a) does not occur because the algorithm does not halt until all goals are discharged. If case (b) occurs then there exists some step $C$ which consumes some literal $P$ which is established by $E$. Intervening between $E$ and $C$ is some step $S$ which clobbers $P$. Now, $S$ must be introduced either (b1) before the goal is discharged or (b2) after the goal is discharged. (b1) cannot occur: if there existed a step $S$ meeting the restrictions above, then the goal would not be discharged by initial conditions because of the `no_violators` test (step 4 of `user_ics`), would not be discharged by `use_prev_step` because the `previous_achiever` predicate checks for intervening clobberers and $E$ would not be inserted before $S$ by `new_step` because the `insert_point` predicate will not permit regressing $E$ beyond a clobberer. (b2) cannot occur because when the goal is discharged, a protection $< E, P, C >$ will be added to the plan, preventing $S$ from being inserted between $E$ and $C$. □

**Theorem 2** *The* PLINTH *algorithm generates all well-formed plans.*

**Definition 1 (Well-formed plans)** *A plan is well-formed if, for each step $S$ in the plan, there exists some other step $S'$ in the plan subtree rooted at $S$ such that $S'$ has a precondition literal $P$ which is not clobbered by any intervening step $S''$ and which is established by $S$. Top-level goals are treated as "preconditions" of a final step $F$ along each branch of the plan tree.*

**Proof:** Since the algorithm performs nondeterministic search (implemented as exhaustive depth-first search with an increasing depth bound), the program will eventually try all ways of satisfying any undischarged goal. Thus it suffices to show that for any well-formed plan, there is *some* choice of goal orderings and goal resolution methods that will generate that plan. For any plan $P$, the following choices will do the job:

1. Order the steps of $P$, $\{p_1, p_2, \ldots, p_n\}$ such that:
   - For any conditional action $A$ with outcomes $\{O_1, O_2, \ldots O_k\}$, all of the steps in the subtree of $P$ rooted at $A - O_i$ come before those in the subtree of $P$ rooted at $A - O_j$, for all $i < j$, and $A$ is after the steps in the subtree rooted at $A - O_1$ and before the subtrees for all the other outcomes, and
   - along every branch of $P$, steps are ordered in inverse chronological order, with the exception noted for conditional actions.
2. For each step $p_i$ in sorted order:
   If $p_i$ is not a conditional action:
   (a) Choose one of the goals satisfied by $p_i$ in $P$, and resolve it by adding $p_i$. Such a goal will exist, because 1) $P$ is well-formed, and 2) we are adding steps in strict inverse chronological order, so any such goal must already be in the set of goals.
   (b) Resolve all of the other goals satisfied by $p_i$ in $P$, using $p_i$ as an establisher. There will be no clobberers, since there are none in $P$.

   If $p_i$ is a conditional action, the argument above will apply to the effects of the outcome $O_1$ of $p_i$, denoted $p_i - O_1$:
   (a) Choose one of the goals satisfied by $p_i - O_1$ in $P$, and resolve it by adding $p_i$. Such a goal will exist, because 1) $P$ is well-formed, and 2) the subtree rooted at $O_1$ has already been added, so any such goal must already be in the set of goals.
   (b) Resolve all of the other goals satisfied by $p_i - O_1$ in $P$, using $p_i$ as an establisher. There will be no clobberers, since there are none in $P$.
3. Finally, resolve the remaining goals against the other outcomes of the conditional actions in $P$.

The steps in the other outcomes for each conditional action will be added in turn—the goals for those outcomes will have been added when the conditional action was first added. □

## Implementation

The algorithm described here has been implemented in the program PLINTH.[2] PLINTH is written in Quintus Prolog. Using a depth-first iterative deepening search strategy in Prolog permitted us to directly implement the algorithm and retain the properties of soundness and completeness. In fact, the planner is simple enough that the algorithm description above is simply an annotated presentation of the code, with two simplifications: code supporting depth-first search and the handling of schema variables has been removed from the discussion here. PLINTH is being applied to planning image processing operations for NASA's Earth Observing System, in collaboration with Nick Short, Jr. and Jacqueline LeMoigne-Stewart of NASA Goddard.

The automatic generation of plans for image analysis is a challenging problem. Preliminary processing (e.g., removal of sensor artifacts) and analysis (e.g., feature detection) involve a complex set of alternative strategies, depending in some cases on the results of previous processing. For example, detailed location of roads and rivers is only worth doing if there is evidence that those features are present in the image.

We have successfully applied PLINTH to the generation of conditional plans for image analysis in "EOS world" (named by analogy to the "blocks world"), a planning domain based on data analysis problems related to the Earth Observing System's Data and Information System (EOSDIS). This domain is a rich one for planning research. Among the capabilities that will be useful for effective automatic planning for satellite data analysis are conditional actions and information gathering, parallel actions, deadlines and resource limitations, and a distributed environment very reminiscent of Etzioni's SOFTBOT environment.

## Conclusions and future work

In this paper, we describe the conditional linear planner PLINTH and its application to image analysis planning for earth science data. Our eventual goal is an *epsilon-safe* version of PLINTH, in which probabilities

[2] PLINTH is not an acronym; it was suggested by analogy with McDermott's PEDESTAL.

attached to action outcomes are employed to focus planning effort on those eventualities most likely to occur, and to bound plan construction using a probability threshold.[3]. For domains in which many actions are conditional, resulting in a large number of possible courses of action, this kind of reasoning will be absolutely necessary. Constructing a complete conditional plan would be infeasible, but epsilon-safe planning allows us to put a principled limit on how much work the planner does.

Our decision to implement a linear planner was both heuristic (it was easier) and pragmatic (it was sufficient). The resulting planner is simple (we present the essential code in this paper), provably sound and complete, and provides a simple platform for investigating futher extensions. One potential pitfall to AI research in the absence of a particular application—or without at least without having some application(s) in mind—is the temptation to add features that are not relevant to solving some problem. A case in point was our initial assumption that an epsilon-safe planner would be needed for image analysis. As it turns out, the potential users of such a planner are not interested in ranking outcomes by probability—they wanted *all* of the interesting eventualities covered. The plans generated are small enough so that this was feasible, and so the current version of PLINTH suffices.[4]

We are investigating various extensions to the planner described here. The combination of probabilities and information gathering actions requires proper handling of the distinction between what is true and what the planner knows. Treating them as the same results in a situation in which making an observation results in the observed proposition having been known before the observation was made (since it was clearly true before the observation was made). Sloppy handling of the semantics of observation actions may result in a planner that makes repeated observations in the hopes of eventually getting the outcome it wants (as distinct from the entirely reasonable case of repeated observations with a noisy sensor). Draper, et al. describe an approach to integrating probablities and observation actions in (Draper, Hanks, & Weld 1993). We present another in (Goldman & Boddy 1994a).

Another direction we are exploring is the construction of more complicated probabilistic models, allowing the encoding of dependencies among various observations (e.g., hearing a weather broadcast changes the likelihood of a road's being passable). It turns out that simple dependencies of this form, at least, are easy to represent: the probabilities for later actions can easily be made dependent on earlier observation outcomes. We have not addressed the issue of later observations affecting the probability of some previous outcome by providing information about what must have been true at that time.

[3]An identical approach extending the probababilistic planner BURIDAN is discussed in (Draper, Hanks, & Weld 1993)

[4]We are not *quite* so naive as to assume that this will be true in all domains.

## References

Boddy, M. S.; Goldman, R. P.; and White, J. 1994. Planning for image analysis. In *Proceedings of the 1994 Goddard AI Conference.* to appear.

Brooks, R. 1991. Intelligence without representation. *Artificial Intelligence* 47:139–159.

Draper, D.; Hanks, S.; and Weld, D. 1993. Probabilistic planning with information gathering and contingent execution. Technical report, Dept. of Computer Science, University of Washington.

Etzioni, O., and Segal, R. 1992. Softbots as testbeds for machine learning. In *Proceedings of the 1992 AAAI Spring Symposium on knowledge assimilation.*

Etzioni, O.; Hanks, S.; Weld, D. S.; Draper, D.; Lesh, N.; and Williamson, M. 1992. An approach to planning with incomplete information. In Nebel, B.; Rich, C.; and Swartout, W., eds., *Principles of Knowledge Representation and Reasoning:Proceedings of the Third International Conference*, 115–125. Los Altos, CA: Morgan Kaufmann Publishers, Inc.

Firby, R. J. 1987. An investigation in reactive planning in complex domains. In *Proceedings AAAI-87*, 196–201. AAAI.

Goldman, R. P., and Boddy, M. S. 1994a. Epsilon-safe planning. forthcoming.

Goldman, R. P., and Boddy, M. S. 1994b. Representing uncertainty in simple planners. In Doyle, J.; Sandewall, E.; and Torasso, P., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference (KR94).* San Mateo, CA: Morgan Kaufmann Publishers, Inc. To appear.

Krebsbach, K.; Olawsky, D.; and Gini, M. 1992. An empirical study of sensing and defaulting in planning. In Hendler, J., ed., *Artificial Intelligence Planning Systems: Proceedings of the First International Conference*, 136–144. Los Altos, CA: Morgan Kaufmann Publishers, Inc.

McDermott, D. 1991. Regression planning. *International Journal of Intelligent Systems* 6(4):357–416.

Minton, S.; Bresina, J. L.; and Drummond, M. 1991. Commitment strategies in planning: A comparative analysis. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence.* Morgan Kaufmann Publishers, Inc.

Pednault, E. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *First International Conference on Principles of Knowledge Representation and Reasoning.* Morgan Kaufmann Publishers, Inc.

Peot, M. A., and Smith, D. E. 1992. Conditional nonlinear planning. In Hendler, J., ed., *Artificial Intelligence Planning Systems: Proceedings of the First International Conference*, 189–197. Los Altos, CA: Morgan Kaufmann Publishers, Inc.

Warren, D. H. 1976. Generating conditional plans and programs. In *Proceedings of the AISB Summer Conference*, 344–354.