Appears in *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, College Park, Maryland, 1992.

Conditional Nonlinear Planning

Mark A. Peot
Stanford University
Department of Engineering-Economic Systems
Stanford, CA 94305
peot@rpal.rockwell.com
(415) 325-7143

David E. Smith
Rockwell International
444 High St.
Palo Alto, CA 94301
de2smith@rpal.rockwell.com
(415) 325-7162

Abstract

Work-in-progress on the design of a conditional nonlinear planner is described. CNLP is a nonlinear planner that develops plans that account for foreseen uncertainties. CNLP represents an extension of the conditional planning technique of Warren [75] to the domain of nonlinear planning.

1 Problem

Most classical AI planners develop unconditional sequences of actions. Observations of the environment during plan execution have no effect on the sequence of actions executed during the course of that plan. In many planning domains, it is desirable to develop a plan that takes advantage of observations made during plan execution to select the actions that are executed. This is a conditional plan. The actions in the plan are conditioned on observations made prior to their execution.

A conditional plan is necessary when uncertainties in the environment or in the results of actions preclude the selection of a single course of action to accomplish a goal. A conditional plan tests the environment to determine whether a planned sequence of actions is appropriate or not. Rather than replanning at runtime, the conditional planner develops a set of plans for every projected contingency. Note that this is not quite the same as reactive planning. A conditional planner develops plans that are reactive only to a few predicted sources of uncertainty. Reactive planners improvise solutions at run time as uncertainties, predicted or unpredicted, arise. A conditional plan does not exhibit the 'persistent goal-seeking behavior' of a reactive plan [Schoppers 89]. Still, a conditional planner may develop good plans for those predicted sources of uncertainty because it considers the effects of actions on several possible futures simultaneously. For example:

–A conditional planner can prepare for contingencies by ensuring that items needed for the execution of more than one possible plan branch are collected when it is convenient to do so. (Example: "I better take something to do in case the coffee house is closed.")

–A conditional planner can avoid "painting itself into a corner" by considering the impact of actions on the feasibility of contingency plans. (Example: "I'm glad that I remembered to bring an extra key!")

–A contingent planner can consider explicitly the benefit of seeking out and observing information. In conjunction with a decision-theoretic plan evaluator, it can identify the optimal information collection policy by implicitly calculating the value of information [Howard 66].

2 WARPLAN-C

Warplan-C [Warren 76] uses a simple approach for developing conditional *linear* plans. Certain of the actions that comprise a plan are tagged as conditional. Conditional actions have two possible outcomes P or $\neg$ P. During the first planning pass, the planner assumes that all of the conditional actions are unconditional with a single outcome P. Warplan-C attempts to develop a plan using one branch of each conditional and then reinvokes the planner to plan for each dangling 'else' branch. This is illustrated below.

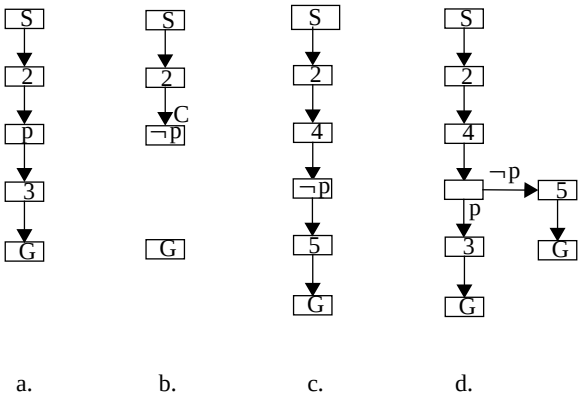


Figure 1: Operation of Warplan-C

The planner develops an unconditional plan assuming that the conditional action has outcome P (Fig 1.a.). After such a plan is found, the planner is reinvoked to plan for the other branch of the conditional. The initial segment used for this plan is the initial segment of the previous plan except that the conditional action is replaced by an action whose preconditions are the protected conditions of the initial segment that were used by action 3 or the goal. The outcome of this new action is ¬ P (shown in Fig 1.b). A new plan is found for this branch of the conditional (Fig 1.c) and the result is combined with the original plan to form the final conditional plan (Fig 1.d). Note that actions can be added before the conditional action in order to satisfy the 'else' part of the conditional (for example, step 4 in Fig 1.c, 1.d). Also note that Warplan-C makes no attempt to fuse the branches of the conditional plan.

3 Conditional Nonlinear Plans

The planner that we are developing is a conditional version of the Systematic Nonlinear Planner (SNLP) [McAllester and Rosenblitt, 91; Soderland and Weld, 91]. In the following discussion, we will describe enough of the terminology to understand an extended example. The complete description of the planner follows the example.

An *operator* is a STRIPS operator [Fikes+Nilsson, 71]. Operators are used to describe actions. An operator consists of a set of preconditions and a set of postconditions. We do not use an add and a delete list. Instead we use three truth values for P. P may be true, false or unknown denoted by P, ¬ P, and Unk(P) respectively.

Actions may have several different, mutually exclusive sets of outcomes for postconditions. The operator `Observe` (described below) has two possible outcomes: either the road from x to y is clear or it isn't. The postconditions associated with a single greek letter are mutually exclusive and are collectively exhaustive. Exactly one of the sets of postconditions denoted by $\alpha_1 ... \alpha_n$ are observed. We will call this type of action a *conditional action.*

```
(Observe (Road ?x ?y))
        Pre:    Unk(Clear ?x ?y)
                (at ?x)
        +α1:    (Clear ?x ?y)
        +α2:   ¬(Clear ?x ?y)
```

A plan *step* is what we call an operator when it appears in a plan. A step is identical to an operator except that it is tagged with labels representing the action's *reason* and the *context*. Reason contains the list of goals that the plan step contributes to. Context captures the set of events in the environment that the action is conditioned on. An action may be executed in a plan if the observations made during execution thus far are consistent with the context of that action. Reason and context will be described more fully later in the paper.

A *causal link* is a triple <E, P, C> where P is a proposition, E is a step that has P in its postcondition, and C is a step that has P as a precondition. We will refer to E as the *establisher* of the condition P and refer to C as the *consumer* of that condition. A causal link represents a commitment that the P precondition of step C will be satisfied by a postcondition of step E and that no other action should interfere in the time between the execution of C and E. Causal links are sometimes referred to as *protection intervals*.

A *conditioning link* is a triple {A, α, B} where A and B are actions and α is a condition. Action B may not be executed unless the outcome of action A is α.

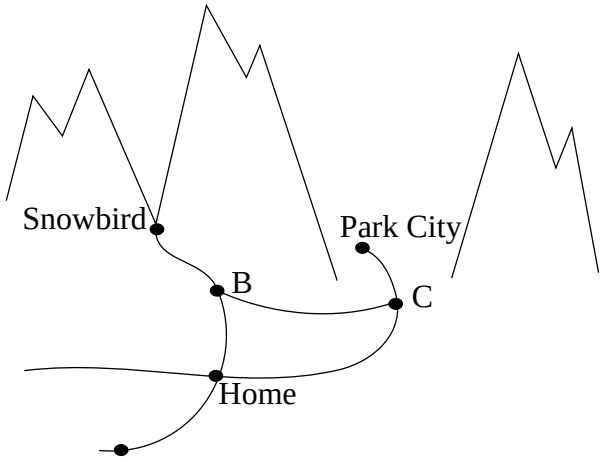
An *ordering constraint* is a constraint on the order of two steps in a plan. (McAllester calls this a *safety condition.*) A > B means that A must be executed at some point after B is executed. There are ordering constraints implicit in causal links and conditioning links. C > E in the causal link <E, P, C> or the conditioning link {E, α, C}.

A *conditional plan* consists of a set of steps, reason and context labels for those steps, a set of ordering constraints, a set of bindings for variables in the plan, a set of causal links and a set of conditioning links.

4 Overview of CNLP

The following simple example demonstrates the operation of our nonlinear planner, CNLP.

Let's say that our objective is to ski. In order to ski, we need to get our skis and drive to a place that has a ski resort. In the simple example diagrammed below, there are only two ski resorts, Snowbird and Park City (abbreviated S and P). It is possible that the road from B to S or the road from C to P will be covered with snow. All of the other roads are clear. If the road is snow covered then it is impassable. If the road from B to S or the road from C to P is snowed in, then we can observe this fact from B or C respectively. There are three operators in this domain: the Observe operator described above, (Drive ?x ?y) and (Get Skis). In order to drive from x to y, we must be at x when the action is executed and the road from x to y must be clear. In order to get our skis, we must be at home. Our goal is to be at a resort with our skis: (At ?x) and (Resort ?x) and (Have Skis). Note that it may not always be possible to achieve the goal.



The planner starts with two dummy operators. The first operator, **IC**, has the initial conditions as its postconditions. The second operator has preconditions corresponding to the conjunctive clauses of the goal. We call our first attempt to instantiate the goal G_1. Initially, the context for G_1 is true. This means that we expect to derive a plan that results in the goal being satisfied in every context. The planner works by adding causal links from actions to *open conditions*, resolving conflicts whenever they arise. An open condition is a precondition that has not been established through a causal link. There are two ways of adding a causal link to a plan. The first, **add-link**[1]**,** adds a causal link between an existing action and an open precondition. **Add-step** adds a new action to a plan. The other planning operators resolve conflicts between pieces of the plan.

The figure on the next page illustrates a conditional nonlinear plan that solves the example problem. IC and the links from IC to other nodes have been omitted for clarity. Solid lines, dashed lines and thick lines represent causal links, ordering constraints and conditioning links, respectively.

CNLP is a nondeterministic planner much like SNLP [McAllester+Rosenblitt 91] or TWEAK [Chapman 87]. This type of planner nondeterministically selects a completions for incomplete plans. We will not attempt to describe how a nondeterministic planning algorithm works and assume that the reader is familiar with [McAllester+Rosenblitt 91, or Soderland+Weld 91].

The plan consists of three different conditional branches corresponding to the case when the road from **b** to **s** is clear, the case when **b** to **s** is not clear and the road from **c** to **p** is clear and the case when neither road is clear. These branches are labelled with the contexts α_1, $\alpha_2\beta_1$, and $\alpha_2\beta_2$ respectively. The actions that comprise a given conditional branch are labelled with the context of that conditional branch. There is a separate goal attempt for every conditional branch. No attempt is made to merge branches after they split.

[1] We have borrowed the names for most of our planning operations from [Soderland+Weld, 91].

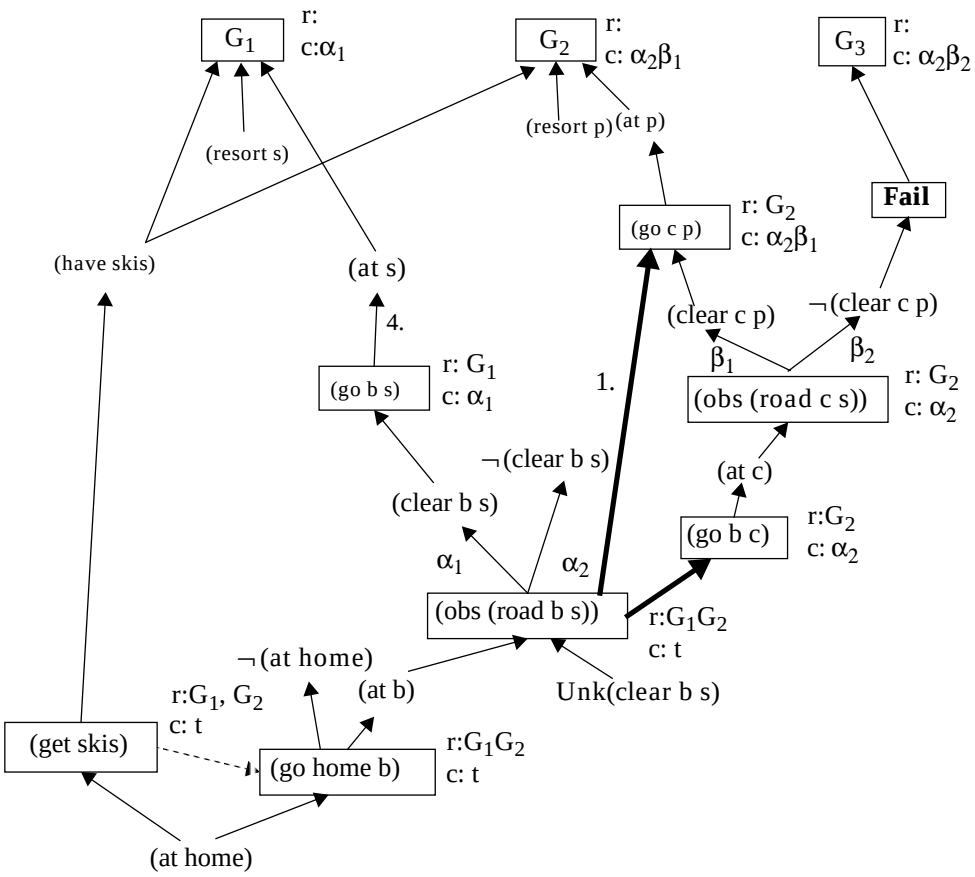


Figure 2 -- A conditional nonlinear plan for skiing.

Before discussing the planner itself, we'll discuss some of the unique features of a conditional nonlinear plan. The first is that actions are tagged with reason and context labels. Consider the action **(get skis)**. This action has a context of 'true' and a reason G_1G_2. A context of 'true' in this situation means that the action can be executed regardless of the result of any observations made during plan execution. Reason G_1G_2 means that the action is used in the first and second attempts to achieve the goal. There is no need to execute an action if the goal actions in its reason become impossible to achieve. When this happens, the action is no longer necessary for any of the goal attempts that are still feasible.

An action like **(Observe (road b s))** has multiple possible outcomes. The outcomes are each labelled with a unique identifier called an *observation label.* Subsequent actions that depend on this observation are tagged with a context that contains an observation label from this action. For example, **(go b s)** has a context containing α_1. This means that **(go b s)** should only be executed when the observation denoted by α_1, **(clear b s)**, has been observed.

The contexts are also used in determining whether a postcondition of an action can clobber a causal link. There is no need to resolve a clobberer if the clobberer and the causal link that it clobbers never occur together in the execution of a plan. The consumer of the causal link and the potential clobberer must have *compatible* contexts in order for a conflict to occur. A new method for resolving clobberers suggests itself. A conflict may be resolved by restricting the contexts of the clobberer and the consumer of the causal link so that these two actions can never occur together in a valid completion of the plan.

The planner starts by attempting to a plan for the initial goal attempt G_1**.** During planning for G_1, its context becomes something other than true. This means that the goal attempt G_1 cannot be achieved in every completion of the plan. When this happens the planner attempts to

achieve the goal in another way. It adds another dummy goal operator, G_2, and attempts to achieve G_2 using only new actions and actions that are consistent with some unexplored context in the plan. This context becomes the *global context* for the planning process. Adding new goal operators rather than fusing all of the branches into the same goal node simplifies the planner, particularly when the planner uses variables. In the skiing example, the actual variable bindings are different in G_1 and G_2. G_1 is the goal for skiing at Snowbird and G_2 represents the goal for skiing at Park City. The global context is used to control the process of adding links to previously planned steps in the plan. A link may only be made to a plan step that has a context that is compatible with the global context. For example, if we are planning for the case where α_2 is true, then we should not be able to link to a plan step that is conditional on α_1 being true since α_1 and α_2 can never become true at the same time.

Finally, there is no possible plan that accomplishes the goal when both roads are blocked. We cannot count on the planner terminating in this situation since planning is undecidable [Chapman, 87]. We suggest that some upper bound be placed on the cost of the steps that can be added to a plan in order to accomplish a particular goal. When this bound is exceeded, the plan branch is considered impossible to achieve. We annotate the plan with a *Fail* operator that has as its precondition the observation(s) that could not be tied into a plan that accomplishes the goal. *Fail* satisfies all of the preconditions of the goal that it corresponds to. In this case, *Fail* is linked to all of the preconditions of G_3, the goal that is impossible to achieve. The context for the Fail operator is set to the current global context. When the plan is executed, the Fail operator signals that it is no longer possible to pursue the goal.

5 CNLP

In this section, we make the definitions suggested in the example more formal and define the planner. For simplicity, we will ignore variable bindings. The planner may be easily extended to include variables using the technique outlined in [McAllester+Rosenblitt 91].

Definition 1: An *observation label* is a unique identifier associated with one of the outcomes of a plan step that has multiple mutually-exclusive outcomes. An observation label is *compatible* with another observation label when the two labels are identical or when they come from different sets of outcomes. We have been denoting observation labels with small, subscripted greek letters (α_3, β_2, γ_1, etc.). For example, if the greek letter denotes an observation, then the subscript would denote the specific result of that observation. α_x refers to the situation where x is the specific outcome of the outcome set denoted by α. Note that α_x is compatible with α_y only when x = y. α_x is always compatible with β_y when $\alpha \neq \beta$ since these labels refer to different observations.

Definition 2: A *context* is a set of observation labels.

A context summarizes a set of observations. When the observations that have been made thus far match the observations denoted by the context of a plan step then that plan step may be executed. For example, if a plan step contains a context $\alpha_2\beta_1$ then that plan step should only be executed if observation event α was observed to have outcome 2 and observation event β was observed to have outcome 1. In our example, the operator (go c p) is contingent on observing $\neg$ (clear b s) and (clear c p), denoted by α_2 and β_1 respectively. The idea of a context seems similar in spirit to the idea of a *chronoset* [McDermott 82]. It is used to identify the possible future that the plan step is a part of.

Observation labels are propagated from the contexts of a plan step to the contexts of following plan steps using the following rules. The context of a plan step, A, includes the observation labels of a plan step, B, whenever there exists a causal link of the form <B, X, A> or a conditioning link of the form {B, α, A}. If B is a conditional plan step, then A also includes the observation label corresponding to the post condition X of step B. Additional observation labels may be added to a context as long as they are compatible with the labels already in the context. The *context of a condition* is the context derived by merging the context of the plan step that establishes the condition with the observation label (if any) associated with the condition itself. This is simpler than it sounds. We are concerned with labelling future branches of a ‘chronicle tree’-like

structure with events that occurred in the past [McDermott 82]. See Figure 3.

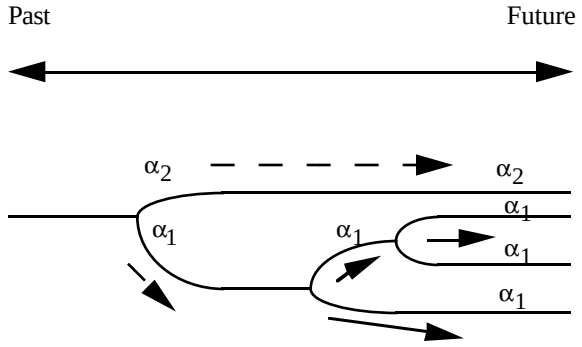


Figure 3: Label propagation in the plan network.

Definition 3: A *descendent* of a step is a plan step that can be reached by following causal links or conditioning links forward in time. An *ancestor* of a step is a plan step that can be reached by following causal links or conditioning links backwards in time.

Definition 4: A *topological sort* of a plan graph is a linear ordering of the steps in the graph that respects the temporal constraints denoted by the graph's causal links, conditioning links, and the ordering constraints.

Definition 5: Plan step A is *possibly after* a plan step B when A is after B in at least one topological sort of the plan graph. A is *possibly before* B when A is before B in at least one topological sort of the plan graph. A is *possibly between* plan steps B and C, if A is possibly after B and A is possibly before C.

Definition 6: The *kernel* of a proposition is the proposition left over after all of the 'negation' and 'unknown' symbols are stripped from it. |p| denotes the kernel of p. For any p, $|\neg p| = |Unk(p)| = |p|$.

Definition 7: A plan step C *clobbers* a causal link <A, p, B> if

1. C is possibly between A and B.
2. The kernel of a postcondition of C, |q|, equals |p| and $p \neq q$.
3. The context of C and B are compatible.

The three parts of this definition are worth some discussion. A clobberer indicates that there is a condition that possibly conflicts with a condition needed for the execution of an action. In order to clobber a causal link, the clobbering action must be able to occur between the time that the establishing action of a causal link occurs and the time that the consuming action occurs. Two conditions are in conflict only when their kernels unify but their truth values differ. SNLP [McAllester+Rosenblitt, 91] declares that two conditions are in conflict whenever the kernels unify even if the clobber condition is identical to the condition that it clobbers. This is done in order to guarantee systematic generation of partial plans. When we tested our version of SNLP we discovered that the time spent in planning dropped dramatically when we modified the planner so that two conditions with identical kernels and truth values are never in conflict. The overhead spent exploring duplicate plans seems much smaller than the overhead spent in declobbering 'nonconflicting' conditions.

Definition 8: An *open condition* is a pair <P, X> where P is a precondition of a plan step X that is not established by a causal link. That is, there is no causal link of the form <A, R, X> where R unifies with P under the bindings of the plan.

The planner works by eliminating conflicts in the plan (eliminating clobberers) and by proving that the preconditions of each plan step are true when that plan step is executed (attaching causal links to open conditions).

Definition 9: A conditional nonlinear plan is called *complete* if the following conditions hold:

1) There are no clobberers, that is, no action clobbers a causal link.

2) There are no open conditions. Every precondition of every step in the plan is linked to an action that establishes it.

3) There exists a topological sort of the plan graph.

4) The contexts for the goals in the plan form a tautology. The context of every postcondition in the plan must be compatible with at least one of the goal operators.

6 Plan Construction Operations

The planning algorithm defined below attempts to make a plan complete by incrementally and nondeterministically repairing all of the sources of incompleteness. A plan can be incomplete whenever it possesses a clobberer, an open condition or there is an observation context that is not compatible with the context of any of the goal operators.

6.1 Resolving Open Conditions.

In order to resolve an open condition, a causal link must be established from a new plan step to the open condition or from an existing plan step. The planner operations that do this are called **add-step** and **add-link**. Both of these operators take an open condition <P, S>, a partial plan, and a cost bound as arguments. Add-link requires the global context, g-context, as an argument so that it may decide which operators may be used to satisfy a subgoal of the current goal. The global context prevents the planner from linking the goal to conditional plan branches that have already been expanded.

Add-Step(<P, S>, plan)

1) Nondeterministically select an operator, O, that possesses a postcondition that matches the unsatisfied precondition, P. This is the new plan step, N.

2) Let new-plan be plan + N + <N, P, S> . That is, bind new-plan to the partial plan constructed by adding step N, and causal link <N, P, S> to the old plan. Update the context of each of the descendents of N to include the context of the condition P. If the new causal link touches a goal node, add the goal node to the reason of N. Return new-plan.

Add-Link(<P, S>, plan, g-context)

1) Nondeterminstically select a step O from plan that is possibly before S, has a context that is compatible with g-context and possesses a postcondition that unifies the precondition P.

2) Let new-plan be plan + <O, P, S> . Update the context of each of the descendents of S as in **Add-Step.** Update the reasons of O and each of the ancestors of O so that they include the reasons of S. Return new-plan.

6.2 Resolving Clobberers

A clobberer can be resolved by attacking any one of the necessary conditions for the clobberer to exist. These are: restricting the clobberer to occur before or after the causal link it clobberers, or restricting the contexts of the clobberer and the link consumer so that they are incompatible. In the following routines, X is the clobbering action, <E, P, C> is the causal link that is clobbered, and Q is the postcondition of X that is doing the clobbering.

Promote(X, <E, P, C>, plan)

1) If C is not a goal and X is possibly after C, let new-plan be plan + (X > C). Return new-plan.

Demote(X, <E, P, C>, plan)

1) If E is not IC, the plan step that contains the initial conditions, and X is possibly before E, let new-plan be plan + (X < E). Return new-plan.

Condition(X, <E, P, C>, plan)

1) Nondeterministically select a conditional plan step A such that A is possibly before both X and C.

2) Nondeterministically select two of the observation labels, α_i and α_j of A such that $i \neq j$.

3) Let the context of A be C_A. If $C_A \cup \alpha_i$ is compatible with the context of X and $C_A \cup \alpha_j$ is compatible with the context of C, then bind new-plan to plan + {A, α_i, X} + {A, α_j, C}. Update the context of X and the contexts of the descendents of X to include $C_A \cup \alpha_i$ and update the context of Y and the contexts of the descendents of Y to include $C_A \cup \alpha_j$. Return new-plan.

6.3 The Top Level.

This is an auxiliary function to edit the plan when a goal cannot be established with a plan that satisfies the current cost bound.

Fail(Plan, Global-Context, Current-Goal)

Fail conditions a *Fail* action on the global-context and links the *Fail* to the current goal. All actions that have Current-Goal as their only reason are pruned from the plan. The current-goal's context is set to the global-context and the resulting plan is returned.

The procedure Find-Conditional-Completion (F-C-C) attempts to complete the plan by fixing flaws in it nondeterministically. G-C is the global context. Goal is the current goal attempt, and Bound is the cost bound. Note that an attempt to plan for a goal is failed if the actions required to accomplish that goal exceed the cost bound.

F-C-C(Plan, G-C, Goal, Bound)

1. If the cost of the portions of Plan that are consistent with the global context exceeds the current cost bound, then return

   F-C-C(Fail(Plan, G-C, Goal), G-C, Goal, Bound).

2. If the plan is complete then exit returning the plan.

3. If the postcondition Q of plan-step X in Plan clobbers a causal link <E, P, C> then bind new-plan nondeterministically to one of the following:

a. Promote(X, <E, P, C>, Plan)

b. Demote(X, <E, P, C>, Plan)

c. Condition(X, <E, P, C>, Plan)

Return F-C-C(new-plan, G-C, Goal, Bound).

4. If there is an open condition <P, S> in the plan then bind new-plan nondeterministically to one of the following:

a. Add-Step(<P, S>, Plan)

b. Add-Link(<P, S>, G-C, Plan)

Return F-C-C(new-plan, G-C, Goal, Bound).

5. Else, there is a post condition with a context, C_{new}, that is not compatible with the context of any goal. Add a new goal step, G_{new}, to the plan and

return F-C-C(Plan, C_{new}, G_{new}, Bound).

7 Discussion

We have described an algorithm to develop conditional nonlinear plans. We are currently implementing the algorithm to confirm that it does in fact work. We also still need to assemble the proofs that show that the algorithm is correct and complete.

There are several novel features of the planning approach proposed. The approach proposed uses a labelling technique similar to that used by an ATMS [DeKleer, 86] in order to keep track of when actions are visible to other actions. We have developed a notion of completeness for conditional planning that does not require that the planner be able to accomplish a goal in all possible circumstances. Finally, we have introduced a new declobbering method that is unique to conditional planning, the **condition** operator.

We believe that the CNLP algorithm will rarely be useful on its own. The size and complexity of the plans generated by CNLP increase exponentially with the number of observation actions in the plan. The amount of computation may be reduced by attaching a relative likelihood measure to the various contexts in the plan. The planner may elect to skip contexts that are sufficiently unlikely, reducing the number of extensions that must be explored by a significant amount. There are other strategies that may be applicable for reducing the time spent in planning such as the use of abstraction or dynamic programming [Einav 91]. We view the development of CNLP as the first step in the development of a decision-theoretic nonlinear planner.

Acknowledgments

The authors would like to thank Jack Breese and Ross Shachter for their contributions to our work. We also thank the anonymous reviewers for their comments and criticism. This work is partially funded by DARPA contract F30602-91-C-0031 and by the National Science Foundation through a Graduate Fellowship.

References

[Chapman, 87] Chapman, D. Planning for Conjunctive Goals, *Artificial Intelligence* **32**, pp 333-377.

[de Kleer, 86] de Kleer, J. An Assumption-based TMS, *Artificial Intelligence* **28**, pp 127-162.

[Einav, 91] Einav, D., Reasoning with Uncertainty and Resource Constraints, PhD Dissertation, Department of Engineering-Economic Systems, Stanford University, 1991.

[Fikes+Nilsson, 71] Fikes, R. E. and Nilsson, N. J. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving, *Artificial Intelligence* **2**(3/4), pp 189-208.

[Howard, 66] Howard, R. E., Information value theory, *IEEE Transactions on Systems Science and Cybernetics*, vol. SSC-2, pp. 22-26, 1966.

[McAllester+Rosenblitt, 91] McAllister, D. and Rosenblitt, D. Systematic Nonlinear Planning, *Proceedings of AAAI-91*, Anaheim.

[Schoppers, 89] Schoppers, M. E., *Representation and Automatic Synthesis of Reaction Plans*, Report No.89-1546, Department of Computer Science, University of Illinois at Urbana-Campaign, 1989.

[Soderland+Weld, 91] Soderland, S. and Weld, D. S., Evaluating Nonlinear Planning, Technical Report 91-02-03, Department of Computer Science and Engineering, University of Washington.

[Waldinger, 77] Waldinger, R. J., Achieving Several Goals Simultaneously, *Machine Intelligence 8,* Chichester: Ellis Norwood Limited.

[Warren, 76] Warren, D. H. D., Generating Conditional Plans and Programs, in *Proceedings of the Summer Conference on AI and Simulation of Behavior*, Edinburgh, 1976.

[Warren, 74] Warren, D. H. D., Warplan: A System for Generating Plans, in Allen, J., Hendler, J, and Tate, A. eds, *Readings in Planning*, San Mateo, California: Morgan Kaufmann, 1990.