Probabilistic Planning with Information Gathering and Contingent Execution*

Denise Draper Steve Hanks Daniel Weld
Department of Computer Science and Engineering, FR-35
University of Washington
Seattle, WA 98195
206-543-1695
{ddraper, hanks, weld}@cs.washington.edu

Abstract

One way of coping with uncertainty in the world is to build plans that include actions that will produce information about the world when executed, and constrain the execution of subsequent steps in the plan to depend on that information. Literature on decision making discusses the concept of information-producing actions (also called sensory actions, diagnostics, or tests), the value of information, and plans contingent on information learned from tests, but these concepts are missing from most AI representations and algorithms for plan generation.

This paper presents a planning representation and algorithm that models information-producing actions and constructs plans that exploit the information produced by those actions. We extend the BURIDAN [Kushmerick *et al.*, 1993] probabilistic planning algorithm, adapting the action representation to model the behavior of imperfect sensors, and combine it with a framework for contingent action that extends the CNLP algorithm [Peot and Smith, 1992] for conditional execution. The result, C-BURIDAN, is an implemented planner that builds plans with probabilistic information-producing actions and contingent execution.

1 Introduction

One way of coping with uncertainty in the world is to build plans that include both information-producing actions and other actions whose execution depends on that information. For example if we wished to acquire a car, we might plan to ask a mechanic to examine a particular car and purchase it only if the report indicates the car is in good working order. Information-producing actions and contingent plans are complementary: it makes no sense to improve one's information about the world if that information can't be exploited later. Likewise, building a contingent plan is useless unless the agent can learn more at execution time than it knows while planning.

This paper presents an implemented algorithm for probabilistic planning with information-producing actions and contingent execution. We extend the BURIDAN [Kushmerick *et al.*, 1993] probabilistic action representation to allow actions with both informational and causal effects, and combine it with a framework for building contingent plans that builds on the CNLP algorithm [Peot and Smith, 1992]. C-BURIDAN takes as input a probability distribution over initial world states, a goal expression, a set of action descriptions, and a probability threshold, and produces a contingent plan that makes the goal expression true with a probability no less than the threshold.

1.1 Example

Suppose that a manufacturing robot is given the goal of having a widget painted (PA), processed (PR), and then notifying (NO) the supervisor that it is done. Processing the widget is accomplished by rejecting (reject) parts that are flawed (FL) or shipping (ship) parts that are not flawed ($\overline{\text{FL}}$). The robot also has an action paint that usually makes PA true, and an action notify that makes NO true. Initially all flawed widgets are also blemished (BL), and vice versa.

Although the robot cannot directly tell if the widget is flawed, the action inspect can be used to determine whether or not it is blemished: executing inspect is supposed to produce a report of **ok** if the widget is unblemished and a report of **bad** if a blemish is detected. The inspect action can be used to decide whether or not the widget is flawed because the two are initially per-

*This paper has been submitted to AI Planning Systems (1994). This research was funded in part by NASA GSRP Fellowship NGT-50822, National Science Foundation Grants IRI-9206733 and IRI-8957302, and Office of Naval Research Grant 90-J-1904.

fectly correlated. The use of inspect is complicated by two things, however: (1) inspect is sometimes wrong: if the widget is blemished then 90% of the time it will report **bad**, but 10% of the time it will erroneously report **ok**. If the widget is *not* blemished, however, inspect will always report **ok**. (2) Painting the widget removes a blemish but not a flaw, so executing inspect after the widget has been painted no longer conveys information about whether it is flawed.

Assume that initially there is a 0.3 chance that the widget is both flawed and blemished and a 0.7 chance that it is neither. A planner that cannot use information-producing actions or contingencies can at best build a plan with success probability 0.7: it assumes the widget will not be flawed, and generates a plan to paint and ship the widget, then notify the supervisor. A planner that can exploit sensor actions and contingencies can generate a plan that works with probability .97 (Figure 1): first inspect the widget, then paint it. Then if the inspection reported **ok**, ship the widget, otherwise reject it. Either way, notify the supervisor of completion. This plan, which C-BURIDAN generates, fails only in the case that the widget was initially flawed but the sensor erroneously reports **ok**. It has success probability $(0.3)(0.1) = 0.03$.

1.2 Contributions

C-BURIDAN is an implemented contingent planner, extending existing planning technology in several ways:

- **Informational effects:** C-BURIDAN can distinguish between an action that *observes* whether an object is blemished (inspect) and one that *changes* whether an object is blemished (paint). This distinction is crucial for effective planning in realistic domains [Etzioni *et al.*, 1992].
- **Branching plans that rejoin:** C-BURIDAN generates contingent plans in which different actions are executed depending on prior observations. C-BURIDAN builds plans whose execution paths can diverge then rejoin, unlike previous planners [Warren, 1976, Peot and Smith, 1992] that support diverging plan branches but do not allow them converge later in the plan.
- **Noisy sensors:** C-BURIDAN's probabilistic action model can represent perfect, noisy, or biased sensors. The accuracy of a sensor can depend on the prevailing world state.
- **Informational dependencies:** C-BURIDAN can make use of correlated information, such as planning to sense BL when it needs information about FL.

2 Actions & Contexts

Our representation and semantics is based on the BURIDAN planner [Kushmerick *et al.*, 1993]; here we provide a brief summary, and refer the reader to [Draper *et al.*, 1993] for more detail. A *state* is a complete description of the world at a point in time. Uncertainty about the world is represented using a random variable over states. An *expression* is a set (conjunction) of literals. In our example, the world is initially in one of two possible states: $s_1 = \{\mathsf{FL}, \mathsf{BL}, \overline{\mathsf{PR}}, \overline{\mathsf{PA}}, \overline{\mathsf{NO}}\}$ and $s_2 = \{\overline{\mathsf{FL}}, \overline{\mathsf{BL}}, \overline{\mathsf{PR}}, \overline{\mathsf{PA}}, \overline{\mathsf{NO}}\}$, and the distribution $\tilde{s}_I$ over these states is $P[\tilde{s}_I = s_1] = 0.3$, $P[\tilde{s}_I = s_2] = 0.7$. In other words, the both states agree that the widget is not PAinted, or PRocessed and that the supervisor has not been NOtified. The most probable state, s_2, has the widget not FLawed and not BLemished.

2.1 Actions

Our action representation distinguishes between changes an action makes to the state of the world and changes it makes to the agent's state of knowledge about the world. The paint action shown in Figure 2 changes the state of the world: if the widget has not yet been PRocessed, with probability 0.95 it will become PAinted and all BLemishes removed, otherwise the action will not change the state of the world at all. The leaves in the figure are called *consequences*; they represent the effect of the action under different conditions in the world.

The inspect action, in contrast, doesn't change whether BL is true or not, but it does provide the agent with information about BL's state. To model the information conveyed by executing an action, we associate a set of *observation labels* with each action—when an action is executed, it will report exactly one of its observation labels to the agent. We identify the conditions that produce an observation label by partitioning the action's consequences into sets called *discernible equivalence classes*, or DECs (indicated in the figures by heavy double ovals), and assign a label to each one. The inspect action has two observation labels, **ok** and **bad**, and two corresponding DECs. If an agent executes inspect and receives the report **bad**, it is certain that BL was true when inspect was executed. A report of **ok** would tend to indicate that BL was false, though the agent could not be certain. The information conveyed by inspect is characterized by the conditional probabilities $P[\mathbf{bad} \mid \mathsf{BL}] = 1$, $P[\mathbf{bad} \mid \overline{\mathsf{BL}}] = 0$, $P[\mathbf{ok} \mid \overline{\mathsf{BL}}] = 0.9$, and $P[\mathbf{ok} \mid \mathsf{BL}] = 0.1$, which is a standard probabilistic representation for an evidence source. The agent's state of belief about BL after receiving a report—$P[\mathsf{BL} \mid \mathbf{ok}]$ or $P[\mathsf{BL} \mid \mathbf{bad}]$—can be computed using Bayes' rule, and depends both on these conditional probabilities and also on the prior probability that BL is true when inspect is executed.

Formally, an action is a set of *consequences*, a set of *observation labels*, and their corresponding *discernible equivalence classes*. Each consequence is a tuple of the form $\langle \mathcal{T}_\iota, \rho_\iota, \mathcal{E}_\iota \rangle$, where $\mathcal{T}_\iota$ is a conjunction of literals known as the consequence's *trigger*, ρ_ι is the conditional probability of this consequence given its trigger, and $\mathcal{E}_\iota$ is the set of *effects* associated with the consequence. Each DEC is a subset of the action's consequences, and together they form a partition of the consequences. Many actions, such as paint will have a single DEC, in which case executing the action provides no information to the agent about which of its consequences actually occurred (and in this case we do not indicate the DEC in the pictorial representation of the action). An action is *information-producing* if it has more than one DEC,

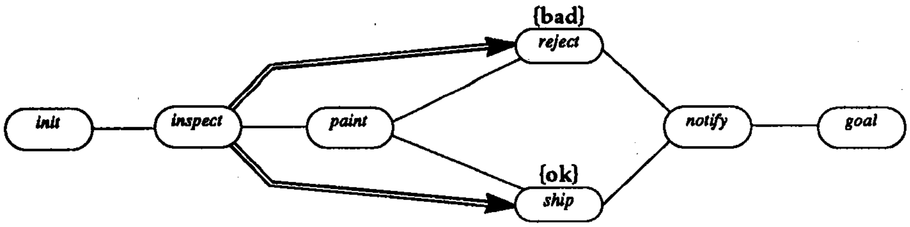

Figure 1: A contingent plan with rejoining branches.

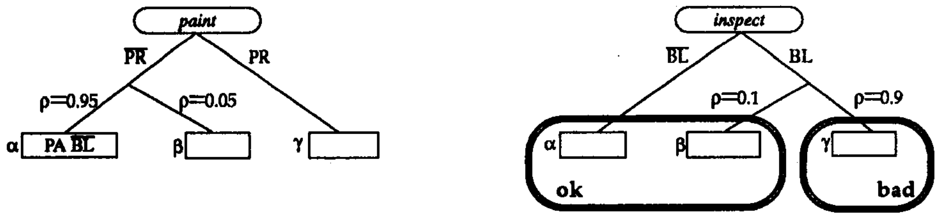

Figure 2: A causal and an information-producing action.

and *causal* if it has nonempty effect sets. Actions can be both information-producing and causal. For example, we might model a **pickup** action that both potentially changes the state of the world—whether the block was being held—and contains observation labels indicating whether or not the action was successful. Likewise a **test-blood** action might detect a disease, but also affects the state of the patient.

2.2 Contexts

We represent contingent execution in a manner nearly identical to CNLP [Peot and Smith, 1992]. Each action in the plan is annotated with a *context*, dictating the circumstances under which the action should be executed. A context is a set (conjunction) of observation labels from previous steps in the plan, denoted context(A_i). We say two contexts are *compatible* if they do not disagree on any action's label.[1] During execution, a step will only be executed when its context is compatible with the actual observations produced by executing previous steps (called the *execution context*).

For example, consider this sequence of annotated actions: ⟨inspect{}, ship{ok}, reject{bad}⟩. An agent would always execute the first step, inspect, since the empty context is always acceptable. Suppose that inspect returned the report **bad**, which would be included in the execution context. The agent would then consider, but decline, to execute ship, since its context is not compatible with the execution context. The agent would finally execute reject, since its context *is* compatible with the execution context.

3 An Overview of the C-BURIDAN Algorithm

C-BURIDAN takes as input a probability distribution $\tilde{s}_I$ over initial states, a set of actions $\{A_i\}$, a goal expression $\mathcal{G}$, and a probability threshold τ. For the problem described in this paper, $\tilde{s}_I$ is defined in Section 2, the set of actions is {inspect, paint, ship, reject, notify}, the goal is {PR, PA, NO}, and we will set $\tau = 0.8$. As output, C-BURIDAN returns a sequence of annotated actions such that their execution achieves $\mathcal{G}$ with probability at least τ.

C-BURIDAN searches a space of *plans*. Each plan consists of a set of *actions* $\{A_i\}$, *contexts* for each A_i, a partial *temporal ordering* relation over $\{A_i\}$, a set of *causal links*, and a set of *subgoals*. A causal link caches C-BURIDAN's commitment that a particular consequence of a particular action should help make a literal true later in the plan. For example, the presence of the link $\text{paint}_\alpha \xrightarrow{PA} \text{goal}$ indicates that the planner has decided that the α consequence of paint is supposed to make PA true for use by goal. Our causal links are similar to the causal links or protection intervals used by many planners, but there are important differences which we will explain below. A subgoal is a pair of the form $\langle d, A_i \rangle$, and represents the planner's intent to make literal d true when action A_i is executed. *Threats* play the same role as in other causal-link planners, but an additional provision is made for contexts: A_t threatens link $A_{p\alpha} \xrightarrow{d} A_c$ if some consequence of A_t asserts $\overline{d}$, if A_t can occur between A_p and A_c, and if context(A_t) is compatible with both context(A_p) and context(A_c).

Like BURIDAN, C-BURIDAN begins searching from an initial *null plan* (Figure 3), which contains the two dummy actions A_0 and A_G (encoding the initial state distribution and the goal expression respectively), and the ordering constraint $A_0 < A_G$. The initial action A_0 has one consequence for each state in the initial probability distribution with non-zero probability. The **goal** action A_G has a single SUCCESS consequence triggered by the goal expression. The null plan's subgoals are all pairs of the form ⟨g, goal⟩, where g is a literal in the goal expression.

Starting from the null plan, C-BURIDAN performs two

[1] In other words, two contexts are compatible if their conjunction is satisfiable.

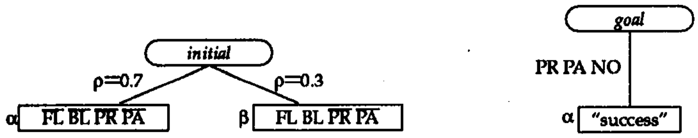

Figure 3: A_0 and A_G encode the initial state distribution and the goal.

operations:

1. *Plan Assessment:* Determine if the probability that the current plan will achieve the goal exceeds τ, terminating successfully if so.[2]
2. *Plan Refinement:* Otherwise, try to increase the probability of goal satisfaction by nondeterministically choosing to support a subgoal (by adding a causal link to a new or existing action) or to protect a threatened link. Fail if there are no possible refinements, otherwise loop.

Refining a plan with conditional and probabilistic actions differs from classical plan refinement (*e.g.* SNLP [McAllester and Rosenblitt, 1991]) in two important ways. First, where SNLP establishes a *single* causal link between a producing action and a consuming action, C-BURIDAN may require *several.* Any SNLP link alone assures that the supported literal will be true. In our representation, a link $A_{p\alpha} \xrightarrow{d} A_c$ ensures that d will be true at action A_c only if the trigger $\mathcal{T}_{p\alpha}$ holds with probability one at A_p, and the consequence's probability $\rho_{p\alpha} = 1$. But when no single link can make the literal sufficiently likely, several links (representing different situations under which the literal might be made true) may suffice. We lose SNLP's clean distinction between an "open condition" and a "supported condition," in return for the ability to represent cumulative support from actions with uncertain consequences.

The second difference lies in how C-BURIDAN resolves threats. Like classical planners, C-BURIDAN may *promote* or *demote* a threatening action by ordering it before the producer or after the consumer of the threatened link. Like BURIDAN or UCPOP [Penberthy and Weld, 1992], C-BURIDAN may also *confront* a threat: when the threatening action has benign as well as threatening consequences, C-BURIDAN can adopt the triggers of one of the benign consequences as subgoals, which has the effect of decreasing the probability of the threatening consequences.

Finally, C-BURIDAN has an additional threat-resolution technique, *branching,* unique to a contingent planner.[3] Intuitively, branching ensures that the agent will never *execute* the threatening step when the link's consuming step is depending on an effect generated by the producing step. We will explain the branching technique in detail in Section 3.1, but first let us examine what progress the planner could make without it:

If (non-contingent) BURIDAN was applied to our example, it would add a paint action to support PAinted, a ship action to support PRoccessed, and a notify action to support NOtified. Assessment would show that the plan has probability of only 0.665, since ship only achieves the desired PRoccessed outcome when the part is not $\overline{\text{FL}}$awed. If BURIDAN tried to provide additional support for PR by adding a new reject action and linking it to the goal, it would produce the plan shown in Figure 4. The problem with this plan is that it has a pair of irreconcilable threats (shown in grey): reject makes PR true, which threatens the link from initial to ship, and likewise ship makes PR true, threatening a link from initial to reject. Adding orderings can resolve only one of these two threats, and confronting the threat would mean that the planner would be trying to achieve two mutually exclusive consequences at once. The predicament becomes apparent: the planner needs to be able to execute either ship or reject but not both, and needs some way to decide under which conditions each step should be executed.

3.1 Threat resolution by branching

"Branching" works by introducing *branches*—a new kind of plan element—into a plan. A branch connects an information-producing action to a subsequent action, indicating which observation labels of the first permit execution of the second. In Figure 5, for example, there are two branches: inspect=**ok**⇒ship and inspect=**bad**⇒reject. The first means that ship should be executed only if the execution of inspect generates an observation label of **ok**, the second means that reject should be executed only if the execution of inspect generates **bad**.

We will use our example to illustrate the branching procedure, attempting to resolve the threat posed by reject_α to the link $\text{initial}_\alpha \xrightarrow{\overline{\text{PR}}} \text{ship}$.

1. We can separate the context of the threatening step A_t = reject from the context of either the link's consumer or its producer, so first choose a step A_s to separate. We will choose A_s = ship.[4]

[2] [Kushmerick *et al.*, 1993] and [Draper *et al.*, 1993] discuss plan assessment in detail. Here we will mention only that correlated information is discovered during assessment. The assessor generates alternative execution profiles, and it notes, for example, that sequences in which FL is initially true are likely to cause inspect to generate an observation of bad, and that subsequently executing reject is likely to succeed, and conversely for $\overline{\text{FL}}$, **ok**, and ship. As a result, the assessor reports that a plan in which reject is executed when **bad** is received and ship is executed when **ok** is received has a high probability of success. The correlation between FL and BL is thus detected by assessment, although an explicit connection between the two propositions is never made.

[3] [Peot and Smith, 1992] call this technique "conditioning." We adopt an alternative term to avoid confusion with "conditional effects" in the action representation.

[4] All choices are nondeterministic—as a practical matter the planner must be prepared to backtrack. For the sake of brevity we will illustrate one correct series of choices.

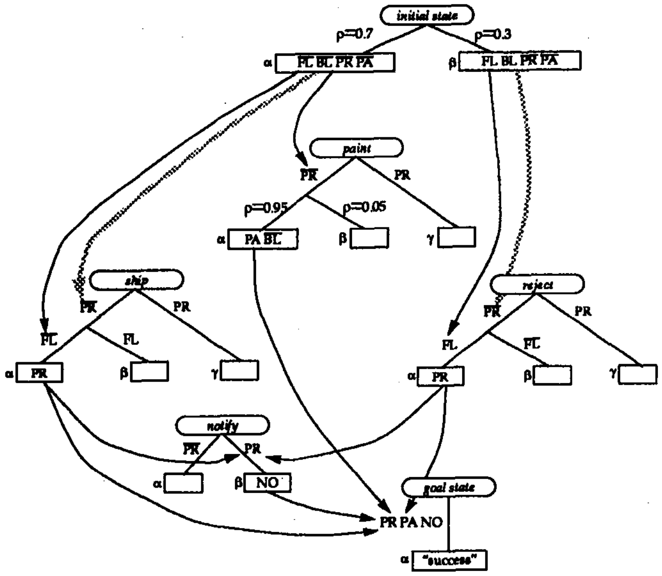

Figure 4: ship and reject threaten each other (indicated by grey links).

2. Choose some new or existing information-producing action A_i that can be ordered before both A_s and A_t, and has a context compatible with context(A_s) and context(A_t). We choose to add a new inspect action to the plan, ordering it before ship and reject. All three actions have empty contexts, so inspect is compatible with both.
3. Choose two observation labels c and c' from A_i.[5] We choose c = **bad**, c' = **ok**.
4. Add the branches $A_i{=}c{\Rightarrow}A_t$ and $A_i{=}c'{\Rightarrow}A_s$ to the plan. Thus we add inspect=ok⇒ship and inspect=**bad**⇒reject.
5. Update the contexts of A_s and A_t to include the new observation labels: context(A_t) := context(A_t) $\wedge$ c, and context(A_s) := context(A_s) $\wedge$ c'. Specifically, context(reject) := {**bad**} and context(ship) := {**ok**}.
6. Adopt each of A_i's triggers as subgoals—we adopt ⟨BL, inspect⟩ and ⟨$\overline{\text{BL}}$, inspect⟩.

Now ship and reject are restricted to mutually exclusive execution contexts, but as yet there is no ordering constraint between inspect and paint. If paint is executed first, however, it will destroy the correlation between BLemishes and FLaws. C-BURIDAN discovers this problem when it supports the subgoal ⟨BL, inspect⟩ with a link from the initial step's β consequence, and finds that paint$_\alpha$ threatens this link. C-BURIDAN can promote the threat, yielding the plan shown in Figure 5. The assessment algorithm determines that the success probability of this plan is 0.9215 > τ, and returns it as a solution. (The plan fails only if paint fails to make PA true or if the widget was initially blemished and inspect incorrectly reports **ok**.) Note that notify will be executed regardless of what inspect reports, even though both ship and reject are subject to contingent execution. This illustrates how C-BURIDAN allows execution sequences to diverge and later rejoin.

3.2 Context propagation

Branching restricts steps to different contexts only when one threatens another. This policy results in plans that are correct, but possibly inefficient: the agent may end up executing actions which are not actually *useful*, even though they do not interfere with other steps in the plan. Suppose, for example, that the ship action had an additional precondition—to have a box—produced by an action get-box. C-BURIDAN would produce the plan fragment in the left of Figure 6, in which the get-box action is always executed, whether or not ship is executed. We would prefer to restrict the context of get-box so it is executed only under the same circumstances as ship, as in the right half of Figure 6. The contexts in which an action is useful depend on the contexts of the actions to which it is connected by causal links. Thus we can determine when an action will be useful by propagating contexts along causal links, and we can restrict an action's context based on the propagated information. [Draper *et al.*, 1993] defines precisely when an action is "useful" in a plan, and develops a propagation algorithm that restricts an action's context accordingly. The algorithm is similar to to the way CNLP propagates context labels, but is adapted to our more general plan structure.

4 Summary and Related Work

C-BURIDAN is an implemented probabilistic contingent planner, combining probabilistic reasoning about actions and information with symbolic least-commitment planning techniques. Causal and informational effects can be freely mixed, and the planner correctly distinguishes between them. The action representation models noisy

[5] More precisely we choose any partition of A_i's observation labels; technically, this requires the more complex definition of context presented in [Draper *et al.*, 1993].

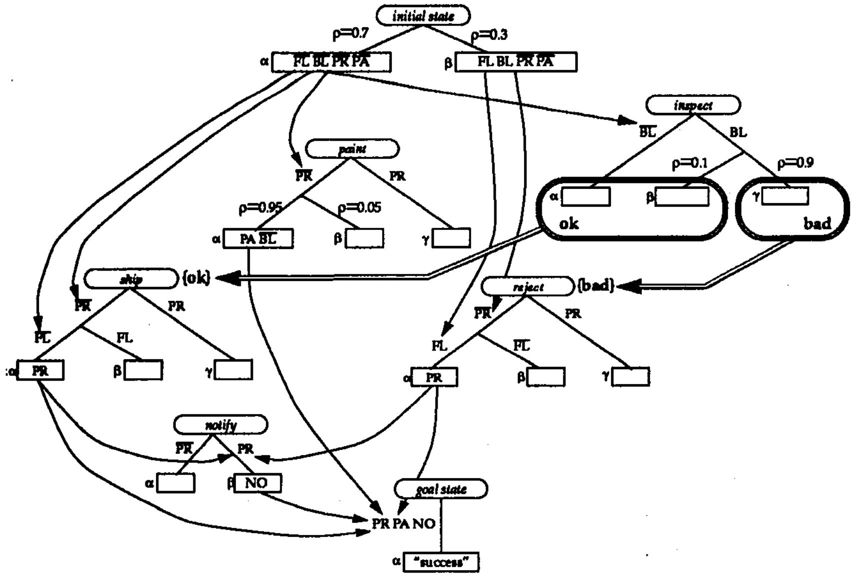

Figure 5: A successful plan

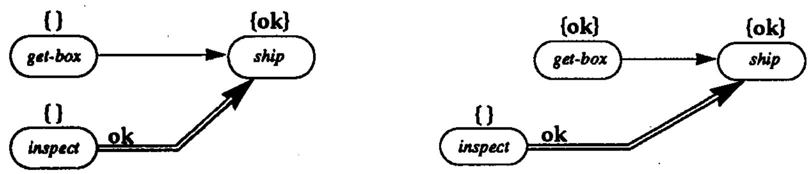

Figure 6: Using propagation to constrain an action's context.

and context-dependent information sources, and allows reasoning about correlated information. C-BURIDAN generates contingent plans in which different actions are executed depending on the result of prior observations, and the representation allows execution sequences to diverge and rejoin.

Related work in conditional planning includes work in decision analysis as well as previous AI planning systems. C-BURIDAN uses a standard Bayesian framework for assessing the value of information and reasoning about sequential decisions [Winkler, 1972], but our emphasis is on automated plan construction from schematic action descriptions and an input problem, whereas work in the decision sciences emphasizes modeling issues.

Our approach to contingent planning borrows much from the CNLP algorithm of [Peot and Smith, 1992]. In particular, branching derives from CNLP's method of conditioning. CNLP uses a very different action model, closely related to the STRIPS representation. CNLP's action model cannot represent a situation in which an action behaves differently depending on the prevailing world state or on unmodeled (chance) factors. CNLP therefore cannot model noisy sensing actions such as `inspect`. We also treat contingencies differently: in CNLP, every time a new execution context is introduced into the plan (by conditioning or branching) a new instance of the goal step is also added with that context—CNLP's plans are thus completely tree-structured.

Cassandra [Pryor and Collins, 1993] is another deterministic causal-link contingency planner. It manipulates a more expressive action representation than CNLP, but uses similar mechanisms for generating branching (contingent) plans.

Future work is oriented toward increasing C-BURIDAN's expressive power (extending the action representation and allowing plans to be evaluated using explicit utility models) and toward building effective applications (developing heuristic methods for controlling the plan-generation and assessment process that allow the solution of larger problems).

References

[Allen *et al.*, 1990] J. Allen, J. Hendler, and A. Tate, editors. *Readings in Planning.* Morgan Kaufmann, San Mateo, CA, August 1990.

[Draper *et al.*, 1993] D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. Technical Report 93-12-04, University of Washington, December 1993.

[Etzioni *et al.*, 1992] O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson. An Approach to Planning with Incomplete Information. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, October 1992. Available via anonymous FTP from `~ftp/pub/ai/` at `cs.washington.edu`.

[Kushmerick *et al.*, 1993] N. Kushmerick, S. Hanks, and D. Weld. An Algorithm for Probabilistic Planning. *Artificial Intelligence*, 1993. To appear. Available via anonymous FTP from `~ftp/pub/ai/` at `cs.washington.edu`.

[McAllester and Rosenblitt, 1991]
D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proc. 9th Nat. Conf. on Artificial Intelligence*, pages 634–639, July 1991. internet file at ftp.ai.mit.edu: /pub/users/dam/aaai91c.ps.

[Penberthy and Weld, 1992] J.S. Penberthy and D. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 103–114, October 1992. Available via anonymous FTP from ~ftp/pub/ai/ at cs.washington.edu.

[Peot and Smith, 1992] M. Peot and D. Smith. Conditional Nonlinear Planning. In *Proc. 1st Int. Conf. on A.I. Planning Systems*, pages 189–197, June 1992.

[Pryor and Collins, 1993] L. Pryor and G. Collins. CASSANDRA: Planning for contingencies. Technical Report 41, Northwestern University, The Institute for the Learning Sciences, June 1993.

[Warren, 1976] D. Warren. Generating Conditional Plans and Programs. In *Proceedings of AISB Summer Conference*, pages 344–354, University of Edinburgh, 1976.

[Winkler, 1972] Robert L. Winkler. *Introduction to Bayesian Inference and Decision*. Holt, Rinehart, and Winston, 1972.