Northwestern University

The Institute for the Learning Sciences

CASSANDRA: PLANNING FOR CONTINGENCIES

Technical Report # 41 • June 1993

Louise Pryor
Gregg Collins

Established in 1989 with the support of Andersen Consulting

Cassandra: Planning for contingencies

Louise Pryor and Gregg Collins
The Institute for the Learning Sciences
Northwestern University
1890 Maple Avenue
Evanston IL 60201

This work was supported in part by the AFOSR under grant number AFOSR-91-0341-DEF. The Institute for the Learning Sciences was established in 1989 with the support of Andersen Consulting, part of The Arthur Andersen Worldwide Organization. The Institute receives additional support from Ameritech and North West Water, Institute Partners, and from IBM.

Abstract

A fundamental assumption made by classical planners is that there is no uncertainty in the world: the planner has full knowledge of the initial conditions in which the plan will be executed, and all actions have fully predictable outcomes. These planners cannot therefore construct *contingency plans*: that is, plans that specify different actions to be performed in different circumstances. In this paper we discuss the issues that arise in the representation and construction of contingency plans and describe Cassandra, a complete and sound partial-order contingent planner that uses a single simple mechanism to represent unknown initial conditions and the uncertain effects of actions.

Cassandra uses explicit decision steps that enable the agent executing the plan to decide which plan branch to follow. The decision steps in a plan result in subgoals to acquire knowledge, which are planned for in the same way as any other subgoals. Unlike previous systems, Cassandra thus distinguishes the process of gathering information from the process of making decisions, and can use information-gathering actions with a full range of preconditions. The simple representation of uncertainty and the explicit representation of decisions in Cassandra allow a coherent approach to the problems of contingent planning, and provide a solid base for extensions such as the use of different decision-making procedures.

Contents

List of Figures

1 Introduction

Many plans that we use in our everyday lives specify ways of coping with various problems that might arise during their execution. In other words, they incorporate *contingency plans*. The contingencies involved in a plan are often made explicit when the plan is communicated to another agent, e.g., "try taking Western Avenue, but if it's blocked use Ashland," or "crank the lawnmower once or twice, and if it still doesn't start jiggle the spark plug." So-called *classical planners*[1] cannot construct plans of this sort, due primarily to their reliance on two *perfect knowledge* assumptions:

1. The planner will have full knowledge of the initial conditions in which the plan will be executed, e.g., whether Western Avenue will be blocked.

2. All actions have fully predictable outcomes, e.g., cranking the lawnmower will definitely either work or not work.

By effectively ruling out uncertainty, these assumptions make it impossible even to represent the notion of a contingency plan within a strictly classical framework.

The perfect knowledge assumptions are an idealization of the planning context that is intended to simplify the planning process. Planning under these assumptions may sometimes prove cost-effective, if the planner's uncertainty about the domain is small, or if the cost of recovering from a failure is low. However, it is dangerous to make these assumptions in general, since they may lead the planner to forgo options that would have been available had potential problems been anticipated in advance. For example, if one assumes that the weather forecast predicting sunshine is necessarily correct, one will not take

[1] Systems such as STRIPS (Fikes & Nilsson, 1971), HACKER (Sussman, 1975), NOAH (Sacerdoti, 1977), MOLGEN (Stefik, 1981), and SIPE (Wilkins, 1988).

along an umbrella: if the forecast later turns out to be erroneous, the umbrella option will no longer be available. When the cost of recovering from failure is high, failing to prepare for possible problems in advance can be an expensive mistake. In order to avoid mistakes of this sort, an autonomous agent in a complex domain must have the ability to make and execute contingency plans.

Recently, we and a number of other researchers (Etzioni, et al., 1992; Peot & Smith, 1992) have begun investigating the possibility of relaxing the perfect knowledge assumptions while staying close to the framework of classical planning. Our work is embodied in the Cassandra program. Cassandra is a contingency planner that has a number of advantages over previous systems, including the following:

- A single mechanism handles both uncertainty due to incomplete knowledge, and that due to unpredictable outcomes.
- The circumstances in which it is *possible* to perform an action are distinguished from the circumstances in which it is *necessary* to perform it.
- The plans constructed by Cassandra include specific steps to decide what contingencies are applicable.
- Information gathering steps are distinct from decision steps.
- Information gathering steps may have preconditions.
- Cassandra constructs a single plan that allows for all contingencies, rather than constructing separate plans that must later be merged.

In this paper we present Cassandra, describe its algorithm in some detail, and show how it handles a variety of example problems.

1.1 Planning with contingencies: An example

Consider the following example: An agent must transport a pile of clean laundry from the laundry room to its apartment, having had its laundry basket stolen from the laundry room while its clothes were drying. Carrying the laundry without using a basket makes it likely that the agent will drop some items along the way. If the agent drops an item without seeing it fall, it will lose that item. Possible plans for coping with this contingency include:

1. The agent carries the laundry in its arms and walks backwards so that it will notice when something drops.

2. The agent carries the laundry in its arms and walks forwards, then retraces its route to search for dropped items after it has delivered the laundry to its apartment.

3. The agent carries the laundry in its arms and walks forwards, turning around every so often to look for dropped items.

Each of these is an example of a contingency plan. In every case, there is a part of the plan, namely retrieving a dropped piece of laundry, that will be executed only when a certain contingency arises. Each plan contains at least one step the sole purpose of which is to determine whether the contingency holds or not.

1.2 Issues for a contingency planner

From our example we can see a number of issues that must be addressed in building a contingency planner:

- The planner must be able to anticipate uncertain outcomes of actions, such as that an item of laundry may or may not drop when it is carried.

- The planner must be able to recognize when an uncertain outcome threatens the achievement of a goal, such as that a dropped item will not be carried back to the apartment.
- The planner must be able to make contingency plans for all possible outcomes, such as picking up something that has dropped.
- The planner must be able to schedule sensing actions that detect the occurrence of the contingency, such as looking to see if something has dropped.
- The planner must produce plans that represent the contingencies in which actions should be performed; for example, if nothing has dropped there is no need to pick anything up.

All these issues have been addressed in the design of Cassandra.

One important issue that is *not* addressed in this work is the problem of determining whether a contingency plan should or should not be constructed in the face of a particular source of uncertainty. Since everything in the real world is uncertain to some degree, the number of contingencies against which the agent could potentially plan is unlimited. The agent must therefore decide which sources of uncertainty are actually worth worrying about. Such a decision requires consideration of the relative likelihoods of the various eventualities and the relative costs of planning for the diverse contingencies in advance, among other things. We do not know of any current theory that adequately accounts for all the complexities involved. Cassandra's role is to construct contingency plans once it has been decided which sources of uncertainty merit consideration.

2 Cassandra's plan representation

The main components of Cassandra's representation of contingency plans are:

- An action representation that supports uncertain outcomes.
- A plan schema.
- A system of labels for keeping track of which elements of the plan are relevant in which contingencies.

In the remainder of this section we will consider these separately.

2.1 Action representation

2.1.1 Basic approach

Cassandra represents actions using a modified version of the STRIPS operator (Fikes & Nilsson, 1971). An operator is defined by the *preconditions* for executing an action and the *effects* that may become true as a result of executing it. For each possible effect there is in addition an associated set of *secondary preconditions* (Pednault, 1988; 1991), which specify the conditions under which the action will have that effect. The use of secondary preconditions is critical to Cassandra's ability to represent uncertain effects, as we shall discuss below.

As a simple example of how such operators are designed, let us briefly consider how we might represent the action of operating an electric drill (figure 1). In order to operate the drill at all, the agent must be holding the drill and the drill must be plugged in; these are thus preconditions for the action. In order to drill a hole in a surface, there must be a bit in the drill, and the bit must be held against the surface. These are therefore secondary preconditions for the effect of producing a hole. A drill can also be used to remove a screw if there is a screw-

```
Action:          (operate-drill ?drill)

Preconditions:   (:and   (holding ?agent ?drill)
                         (plugged-in ?drill))

Effects:         (:when (:and  (installed-in-chuck ?bit ?drill)      ; secondary precondition
                               (bit ?bit)
                               (bit-size ?bit ?size)
                               (contact-surface ?bit ?surface ?loc))
                  :effect (hole-in ?surface ?size ?loc))            ; effect

                 (:when (:and  (installed-in-chuck ?s ?drill)       ; secondary precondition
                               (screwdriver ?s)
                               (screwdriver-engaged ?s ?screw)
                               (screw-attached ?screw ?object))
                  :effect (not (screw-attached ?screw ?object)))    ; effect
```

Figure 1 Representation of operating an electric drill

driving attachment in the drill, and the screw-driving attachment must engage the screw. These are therefore secondary preconditions for the effect of removing a screw.

2.1.2 Representing uncertain outcomes

A key element of Cassandra's design is the use of a single format to represent all sources of uncertainty. In particular, uncertainty is assumed to stem from intrinsically uncertain action outcomes, i.e., outcomes that cannot be predicted on the basis of any factor that is included in the planner's representation of the situation. This formulation ignores uncertainty that might stem from outside interference during the execution of the agent's plans.[2] All other sources of uncertainty can, however, be handled within this framework. In particular:

- Uncertainty about initial conditions is handled as a special case of uncertainty about the outcomes of actions; this is possible because Cassandra, in

[2] This is actually a limitation of classical planners in general; all change in the world is assumed to be caused directly by the actions of the agent.

```
Action:          (operate-drill ?drill)
Preconditions:   (:and   (holding ?agent ?drill)
                         (plugged-in ?drill))
Effects:         (:when (:and   (installed-in-chuck ?bit ?drill)      ; secondary precondition
                                (bit ?bit)
                                (bit-size ?bit ?size)
                                (contact-surface ?bit ?surface ?loc)
                                (:unknown ?ok T)
                  :effect (hole-in ?surface ?size ?loc))             ; effect

                 (:when (:and   (not (hole-in ?surface ?size ?loc))) ; secondary precondition
                                (:unknown ?ok F))
                  :effect (not (hole-in ?surface ?size ?loc)))       ; effect
```

Figure 2 Representation of operating a faulty drill

common with many classical planners, treats the initial conditions as though they were the effects of a phantom "start step" action.

- Uncertainty about an effect due to uncertainty about the preconditions of that effect is represented indirectly; it is assumed that such uncertainty stems from the intrinsic uncertainty of some effect of a prior action. Cassandra correctly propagates uncertainty through the plan.

Cassandra represents the intrinsic uncertainty of effects by assigning them *unknowable preconditions*, which are constructed using the pseudo-predicate :unknown. By giving an effect an unknowable precondition, we indicate that it cannot be determined in advance whether that effect will result from executing the action. As an example of an operator with an uncertain effect, consider the action of operating an intermittently malfunctioning drill. According to the representation in figure 2, it is not certain whether executing the operator will result in a hole being drilled. This is expressed in terms of two uncertain effects, one the proposition that there is a hole of the appropriate sort after the operator is executed, the other the proposition that there is not.

Obviously, these effects are related. In particular, the source of the uncertainty is the same in both cases, with the two effects representing different

outcomes of that uncertainty. In order to express this sort of relationship, the :unknown pseudo-predicate must take two arguments, one standing for the source of the uncertainty (?ok in the example), the other for a particular outcome of the uncertainty (T for the outcome in which the drill works, F for the outcome in which it does not). The source of uncertainty is indicated by a variable in the operator schema because each instantiation of the operator will introduce a new source of uncertainty. The planner must bind this variable to a unique name (i.e., a skolem constant) when the operator is instantiated.

If a set of effects are represented as stemming from the same source of uncertainty, the following propositions are assumed:

1. Each outcome name (e.g., T or F) designates a unique outcome for a given source of uncertainty.

2. Different outcomes of a given uncertainty are mutually exclusive.

3. The set of named outcomes is exhaustive.

These propositions license two critical judgments that Cassandra must be able to make in order to construct a viable contingency plan:

1. That two actions or effects may not co-occur because they depend upon different outcomes of the same uncertainty.

2. That a goal will necessarily be achieved by a plan, because it will be achieved for every possible outcome of every relevant uncertainty.

2.2 Basic plan representation

Cassandra's plan representation is an extension of that used in UCPOP (Penberthy & Weld, 1992) and SNLP (Barrett, et al., 1991; McAllester & Rosenblitt, 1991),

which is in turn derived from the representation used in NONLIN (Tate, 1977). A plan is represented as a schema consisting of the following components:

- A set of *steps*, each of which is an instantiation of an *operator* (see previous section).
- A set of *effects*, namely, the anticipated effects of the steps that are scheduled.
- A set of *links* connecting steps that achieve effect with the steps for which those effects are preconditions. Links in effect denote *protection intervals*, i.e., intervals over which particular conditions must remain true in order for the plan to work properly.
- A set of *variable bindings*, made in the course of instantiating the operators from which the plan is constructed.
- A *partial ordering* on the steps.
- A set of *open conditions*, which are preconditions that have not yet been established.
- A set of *unsafe links*, which are causal links the conditions of which could be falsified by other effects in the plan.

A plan is *complete* when there are no open conditions and no unsafe links.

2.3 Representing contingencies

When a plan that has been produced by a conventional planner is carried out, it is assumed that every step in that plan will be executed. In contrast, the whole point of contingency planning is to produce plans in which some of the steps may *not* be carried out. This has several important consequences for the planning algorithm:

- Plan steps must be marked in such a way that the agent executing the plan can determine which steps to perform in a given contingency.

- The planner must be able to determine whether a given effect of a scheduled action will arise in a given contingency, in order to decide whether that effect can be used to achieve a goal.

- The planner must be able to determine the contingencies in which a particular step and a particular link co-occur, in order to determine whether (and when) the step threatens to clobber the link.

Keeping track of which elements of the plan are relevant in various contingencies requires a good deal of bookkeeping, which Cassandra accomplishes by attaching a label to each element of the plan designating the contingencies in which that element plays a role. In order to construct such labels, Cassandra must have a means of designating a particular contingency. A contingency is uniquely described by a pair of symbols, one standing for the source of the uncertainty that gives rise to the contingency, the other standing for a particular outcome of that uncertainty. For example, as described above, an instance of the faulty drill operator introduces a single source of uncertainty with two possible outcomes. For a given instance, the source of uncertainty might be designated OK1; as we have seen, the two outcomes are designated T (the drill works) and F (the drill does not work). The contingency in which the drill functions properly would thus in this instance be designated [OK1: T], while the contingency in which the drill malfunctions would be designated [OK1: F]. This labeling system allows the system to determine by inspection whether different contingencies are associated with alternative outcomes of the same source of uncertainty.

Contingency labels can be attached to actions, effects, and goals. The labels can be either positive or negative. Labelings have the following interpretations:

- *Positive contingency label on an action*: The action must be executed in that contingency.

- *Negative contingency label on an action:* The action must not or cannot be executed in the contingency.

- *Positive contingency label on an effect:* The effect must be established in the contingency.

- *Negative contingency label on an effect:* The effect must not or cannot be established in the contingency.

- *Positive contingency label on a goal*: The goal must be achieved in the contingency.[3]

Contingency labels allow the planner to quickly determine whether a particular plan element is relevant in a particular contingency. Positive contingency labels indicate elements that are definitely part of the plan for that contingency, negative contingency labels indicate elements that are definitely *not* part of the plan for that contingency, while elements that are unlabeled with respect to a given contingency may or may not occur in that contingency.

2.4 Representing decisions

In general, plans involving contingencies *branch*. When an agent executes a branching plan, it must at some point decide which branch to take. Rather than treating this decision as an explicit part of the plan, previous work has in effect simply assumed that the agent will execute those steps that are consistent with the contingency that actually obtains (Peot & Smith, 1992; Warren, 1976). However, the agent cannot make this determination automatically; in order to know which contingency holds during execution, an arbitrarily large amount of work may be necessary, in particular the work of gathering information upon

[3] There is no need for negative contingency labels on goals; the negative labels on effects are used to prevent a goal from being achieved by an effect in an incompatible contingency.

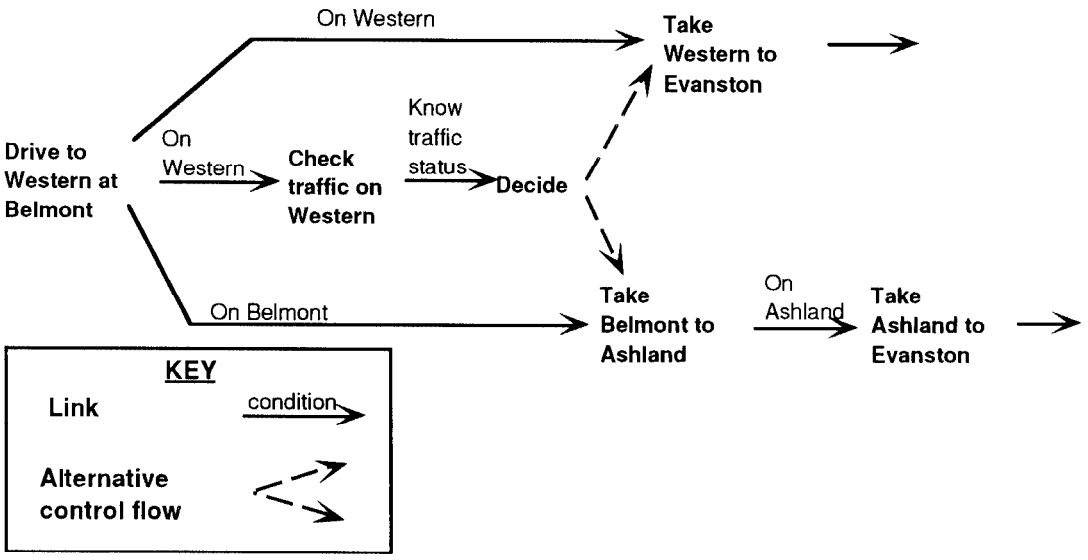


Figure 3 A plan that includes a decision step

which the decision can be based. To ensure a viable plan, the planner must be able to guarantee that the steps required to gather information do not conflict with those required to carry out the rest of the plan. Therefore, the planner must in general be able to include information gathering steps, as well as any other steps that support decision making, in the plan it is constructing. Cassandra achieves this by representing decisions explicitly as plan steps. The preconditions of these decision steps include goals to be in possession of information relevant to making the decision; the scheduling of actions to obtain information is thus handled by the normal planning process.

For instance, consider the contingency plan alluded to above: *"try taking Western Avenue, but if it's blocked use Ashland."* During the execution of such a plan, the agent must at some point decide which alternative branch of the plan to execute. The decision step in this case would have the precondition of knowing whether Western Avenue is blocked or not, which would cause the planner to schedule an information-gathering action to check the traffic status on Western. This operation might in turn have the precondition of being on Western.

Assuming the goal of the plan is to be in Evanston, the final plan might be as depicted in figure 3. Note that solid lines in the diagram represent links, with the operation at the tail of the link achieving a condition for the operation at the head of the link. Dashed lines represent control flow after a decision. In this case, the agent will take Western to Evanston if it decides on one contingency, and will take Belmont to Ashland and Ashland to Evanston if it decides on the other.[4]

Notice that in order to determine the appropriate precondition for a given decision step, the planner must have some way of determining exactly what it will need to know in order to make the decision at execution time. This somewhat complex determination depends in part on how the decision-making process is to be carried out. In Cassandra, decisions are modeled as consisting of the evaluation of a set of condition-action rules of the form:

if *condition 1* then *contingency 1*
if *condition 2* then *contingency 2*

if *condition n* then *contingency n*

Each possible outcome of a given uncertainty gives rise to one decision rule; the condition of this decision rule specifies a set of effects that the agent should test in order to determine whether to execute the contingency plan for that outcome. For example, the decision rules for the driving plan example would look like this:

if *Western Avenue is blocked* then *execute contingency using Ashland*
if *Western Avenue is not blocked* then *execute contingency using Western*

Cassandra's derivation of inference rules in decisions is explained in detail in section 4.

[4]The plan Cassandra produces for this situation is in the Appendix, which shows plans for all the examples described in this paper.

The preconditions for a decision step are goals to know the truth values of the conditions in the decision rules. These goals are treated in the same way as are the preconditions of any other step. Thus, the planner requires no other special provisions to allow the construction of information-gathering plans.

Representing decision steps explicitly also provides a basis for supporting alternative decision procedures. While Cassandra's current model of the decision process is quite simple, more complex decision procedures could be supported within the same framework. For example, the model could be changed to a differential-diagnosis procedure. The representation of decision procedures as templates in the same way that actions are represented as templates would allow the planner to choose between alternative methods of making a decision in the same way as it can choose between alternative methods of achieving a subgoal. An even better approach might be to formulate an explicit goal to make a correct decision, and allow the system to construct a plan to achieve that goal using inferential operators. However, this would in effect require that the goals for these operators be stated in a meta-language describing the preconditions and results of operators. We have not yet addressed this possibility in any detail.

Cassandra's separation of information gathering from decision-making allows one information-gathering step to serve several decisions. This allows a much more flexible use of information-gathering actions; there is no effective difference between such actions and any other action that may appear in a plan.

3 Planning without contingencies

In this section we will briefly review the basic planning algorithm on which Cassandra is based. This follows closely that used in UCPOP (Penberthy & Weld, 1992), which is in turn based on SNLP (McAllester & Rosenblitt, 1991) The principal difference between UCPOP and SNLP is the use of secondary preconditions, an adaptation originally suggested by Collins and Pryor (1992a). Readers wishing for more details on the evolution of this class of planners should see also (Barrett,

et al., 1991; Barrett & Weld, 1993; Chapman, 1987; Collins & Pryor, 1992b; Hanks & Weld, 1992; Tate, 1977).

Cassandra does not attempt to construct a contingency plan until it encounters an uncertainty. Up until this point, it constructs a plan in much the same manner as other planners in the SNLP family. In fact, if no uncertainty is ever introduced into the plan, Cassandra will effectively function just as the UCPOP planner would given the same initial conditions. Planning proceeds through the alternation of two processes: *resolving open conditions* and *protecting unsafe links*. Each of these processes involves a choice of methods, and may therefore give rise to several alternative ways to extend the current plan. All possible extensions are constructed, and a best-first search algorithm guides the planner's exploration of the space of partial plans.

The initial plan consists of two steps: the *start* step, with no preconditions and with the initial conditions as effects, and the *goal* step, with the goal conditions as preconditions and with no effects.[5] The planner attempts to modify its initial plan until it is *complete*: i.e., until there are no open conditions and no unsafe links.

3.1 Resolving open conditions

The planning process is driven by the need to satisfy open conditions. Initially, the set of open conditions consists of the input goals. In the course of planning to satisfy an open condition, new subgoals may be generated; these are then added to the set of open conditions. The planner can establish an open condition in one of two ways: by introducing a new step into the plan, or by making use of an effect of an existing step (see figure 4). If a new step is added, the preconditions of the step become open conditions. The secondary preconditions of the effect that establishes the condition become open conditions as well. Finally, each time

[5] Representing initial conditions and goals in this way is merely a device for simplifying the description of the planning algorithm.

New step	Add a new step to the plan that has an effect that will establish the open condition. Add the step preconditions and the secondary preconditions of the effect as open conditions. The open condition becomes a completed link.
Reuse step	Make the open condition into a complete link from an effect of an existing plan step. Add the secondary preconditions of the effect as open conditions.

Figure 4 Resolving open conditions

an open condition is established, a link is added to the plan to protect the newly established condition.

One way of establishing a condition is simply to notice that the condition is true in the initial state. Because the initial conditions are treated as the results of the *start* operator, which is always a part of the plan, this method can be treated as establishment by use of an action already in the plan; indeed, this simplification is the motivation for representing the initial conditions in this way.

3.2 Protecting unsafe links

Whenever an open condition is established, links in the plan may be jeopardized either because a new step threatens an existing link, or because a new link is threatened by an existing step. The situations in which a link is unsafe are shown in figure 5. In general, if there is an effect in the plan that could possibly interfere with the condition established by a link, that link is considered unsafe.

A link L is unsafe if there is an effect E in the plan (other than the effect LE1 that establishes the condition in the link and the effect LE2 that is either established or disabled by the link) with the following properties:

Unification	One of the postconditions in E can possibly unify with the condition that L establishes or the negation of the condition that L establishes.
Ordering	The step that produces E can, according to the partial order, occur both before the step that produces LE2 and after the step that produces LE1.

Figure 5 Unsafe links

Separation	Modify the variable bindings of the plan to ensure that the threatening effect E cannot in fact unify with the threatened condition.
Ordering	Modify the ordering of the steps in the plan to ensure that the step producing E occurs either before the step that produces LE1 or after the step that produces LE2.
Preservation	Introduce a new open condition in the plan to disable E. This new open condition is the negation of E's secondary preconditions.

Figure 6 Resolving unsafe links

There are three general methods of protecting a threatened link: *ordering*, *separation*, and *preservation* (see figure 6). Ordering means constraining the threatening action to occur either before the beginning or after the end of the threatened link. Separation means constraining the way in which variables may be bound in order to ensure that the threatening action will not in fact interfere with the threatened link. Preservation means generating a new subgoal to disable the effect that threatens the link.

4 Contingency planning

Cassandra proceeds as described in the previous section until the plan is completed or until an uncertainty is introduced. In this section, we will describe how uncertainties are introduced, and how they are handled. As an example of plan involving an uncertainty, let us consider the classic "bomb in the toilet" problem (McDermott, 1987, citing Moore), in which the goal is *disarm bomb*, and the initial conditions are *bomb in package1* or *bomb in package2*. The uncertainty in this case lies in the initial conditions: depending on the outcome of the uncertainty, the *start* operator can either have the effect that the bomb is in package1, or the effect that the bomb is in package2.

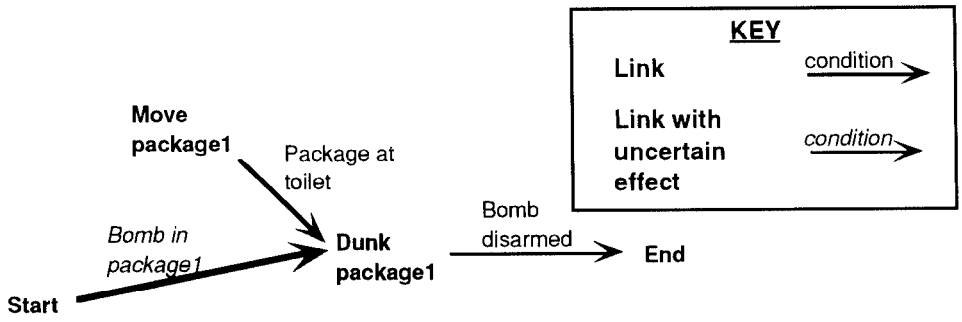


Figure 7 The introduction of uncertainty into a plan

4.1 Contingencies

4.1.1 The introduction of uncertainty

An uncertainty is introduced into a plan when an open condition in the plan is achieved by an uncertain effect, i.e., an effect with an unknowable precondition. In the bomb-in-toilet example, for instance, the planner may achieve the condition *bomb is disarmed* by selecting the *dunk* operator, which has the preconditions *the package is at the toilet*, and *the bomb is in the package*. The condition *the bomb is in the package* can be established by identifying it with *the bomb is in package1*, which is an effect of the *start* operator. However, this condition is uncertain, as the planner can determine by noting that it has an unknowable precondition. Cassandra will attempt to deal with this uncertainty by introducing a new contingency (or new contingencies) into the plan. The state of the plan just after the introduction of the uncertainty is illustrated in figure 7.

4.1.2 Introducing contingencies

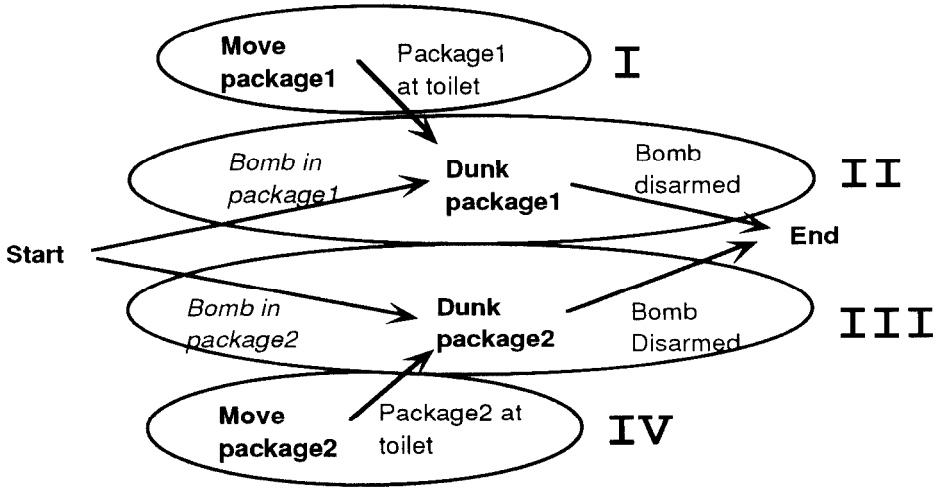
Cassandra notices an uncertainty when its current plan becomes dependent upon a particular outcome of that uncertainty. In effect, the plan that it has built so far

becomes a contingency plan for that outcome. In order to build a plan that is guaranteed to succeed, the planner must construct contingency plans for all other possible outcomes of the uncertainty as well. This means splitting the plan into a set of branches, one for each possible outcome of the uncertainty.

Each contingency plan must ultimately result in the achievement of the overall goal of the plan, and each must enforce the assumption of the particular outcome of the uncertainty. Cassandra thus begins the construction of a contingency by making a copy of the overall goal of the plan, assigning it a label indicating the outcome of the uncertainty that it assumes, and adding it to the set of open conditions. One such goal is constructed for each alternative outcome of the uncertainty.

In planning for these otherwise identical goals, the planner must make certain that no element of the plan for a given goal relies upon an outcome of the uncertainty other than the one assumed by that goal. In other words, for a particular goal, Cassandra must ensure that neither this goal nor any of its subgoals is achieved by any effect that depends, directly or indirectly, upon any outcome of the uncertainty other than the one in the goal's label. In order to make the enforcement of this constraint as efficient as possible, Cassandra labels each element of the plan that depends upon a particular outcome of an uncertainty with all the contingencies in which that element can *not* be used—namely, those associated with other possible outcomes of that uncertainty (see discussion of labeling, above). These *negative labels* make it simple for the planner to check whether a given effect can be used to establish a particular goal. In order to support this reasoning, Cassandra must propagate labels in two different ways. In particular:

- Positive labels indicating the contingency to which a goal belongs must be propagated from goals to their subgoals.



Element label classes

I — In 'package1' contingency.
II — In 'package1' contingency; out of 'package2' contingency.
III— In 'package2' contingency; out of 'package1' contingency.
IV — In 'package2' contingency.

Figure 8 A contingency plan to disarm a bomb

- Negative labels indicating the contingencies in which a plan element may not appear must be propagated from plan elements to other elements that causally depend upon them.

In the bomb in the toilet example, when the plan is made dependent upon the uncertain outcome *bomb in package1*, a new copy of the top level goal *disarm bomb* is added to the set of open conditions, with a label indicating that it belongs to the contingency in which the bomb is in package2.[6] The existing top level goal and all its subgoals will be labeled to indicate that they belong to the contingency

[6]Note that we are describing the contingency in this way for clarity of exposition. The actual label is constructed as described above.

in which the bomb is in package1. The effect *bomb in package1*, the action *dunk package1*, and all effects of the action *dunk package1* will be labeled to indicate that they cannot play a role in the contingency in which the bomb is in package2.[7]

When the planner attempts to achieve the new open condition *disarm bomb*, it may choose the dunk operator once again (notice that it is prohibited from using the effect of the existing dunk operator). This instance of the dunk operator in turn gives rise to a subgoal to have the bomb be in the package that is dunked. This can only be achieved by identification with the effect *bomb in package2*. The plan thus constructed is depicted in figure 8 (the decision step has been omitted for clarity).

4.1.3 Issues in contingency planning

Cassandra's approach to contingency planning raises a number of subtle and difficult issues concerning the desired behavior of such a planner. In this section we briefly consider a few of those issues.

Dependence on outcomes

The fact that a contingency plan assumes a particular outcome of an uncertainty means only that it cannot depend upon a different outcome of the uncertainty. Cassandra does not enforce any constraint that the plan must causally depend upon the outcome that it assumes. For instance, let us reconsider the Western/Ashland example. The plan to take Ashland does not actually depend on Western being blocked; it could be executed successfully regardless of the amount of traffic on Western.

[7] Notice that the action *move package1*, although it plays a role in the contingency plan when the bomb is in package1, does not in fact depend upon the bomb being in package1. It could in principle be made part of the plan for disarming the bomb in the contingency in which the bomb is in package2, were it to prove useful for anything.

Superfluous contingencies

The preceding observation raises a question: If a plan for a contingency turns out not to depend upon any outcome of the uncertainty that gave rise to it, would this not obviate the need for alternative contingencies? For instance, in out example, it might seem sensible to execute the plan to use Ashland whether Western is blocked or not. It might thus seem that the planner should edit the plan in some way so as to eliminate apparently superfluous contingencies. However, it can easily be shown that a version of the plan that does not involve dependence on any outcome of the uncertainty will be generated elsewhere in the search space. In the example, this would mean that the planner would in fact consider a plan that simply involved taking Ashland. If the search heuristics penalize plans involving contingencies appropriately, this other plan should be preferred to the contingency plan, all other things being equal.

One-sided contingencies

The preceding discussion notwithstanding, a plan involving no contingencies is not always superior to a plan involving a contingency. This is why a planner might in fact construct a plan like the Western/Ashland one. To take a more clear-cut example, suppose the planner needs $50 to bet on a horse. It might borrow the $50 from John, but the outcome of this action is uncertain—John might refuse. Alternatively, it could rob a convenience store. While the robbery plan would (we shall stipulate) involve no uncertainties, it is a bad plan for other reasons. It would be better to first try to borrow $50 from John, and then, if that fails, rob the convenience story. As outlined above, Cassandra could generate this plan. In order to make it prefer the plan to the contingency-free alternative, however, its search metric would have to take into account the estimated costs of various actions, and to perform something akin to an expected value computation. (See (Feldman & Sproull, 1977) for a discussion of decision-theoretic measures applied to planning.) In order to execute the plan properly, it would also be necessary for it to have some way of knowing that the borrowing plan should be preferred to the robbery plan when either might be executed.

Identical branches

It is possible that a single plan could work just as well for several different outcomes of an uncertainty. For instance, suppose the action of asking John for $50 has three possible outcomes: Either the planner gets the money and John is happy (at having had the opportunity to do a favor), or the planner gets the money and John is unhappy (at having been obliged to do a favor), or the planner does not get the money at all. If the planner builds a plan in which it tries to borrow $50 from John to bet on a horse, then, assuming that this plan does not depend upon John's happiness (which it might, for example, if the planner needed to get a ride to the track from John), the plan will work for either the "get money + John happy" outcome or the "get money + John unhappy" outcome.

Cassandra could find such a plan, but it would in effect have to find it twice—once for each outcome of the uncertainty—and it would still require a decision step to discriminate between those outcomes. This is inefficient in two ways: the extra search time required to find what is essentially the same plan twice is wasted, and effort is put into making an unnecessary decision. We are looking into ways to avoid the former problem. The latter could be solved by a post-processor that would "merge" identical contingency plans, but we have not implemented this technique.

Branch merging

Rather than having each member of a set of contingencies achieve the overall goal of the plan, it is possible to have each contingency achieve a subgoal of the overall goal, given that no other part of the plan depends upon an outcome of the same uncertainty. In other words, in terms of the branching plan metaphor, it is possible to construct a plan in which branches split, and then reunite. For instance, consider the Western Avenue/Ashland Avenue plan once again. The overall plan in which this goal arises might be a plan to deliver a toast at a dinner to be held in an Evanston restaurant. The contingency due to uncertainty about traffic on Western Avenue would in this case seem to affect only the portion of

the plan concerned with getting to Evanston; it probably has little bearing on the wording of the toast, the choice of wine, and so on. The most natural way to frame this plan might thus be to assume that regardless of which contingency is carried out, the planner will eventually arrive at a certain location in Evanston, and from that point a single plan will be developed to achieve the final goal.

Constructing the plan in this way would result in a more compact plan description, and might thus reduce the effort needed to construct the plan by avoiding, for example, the construction of multiple copies of the same subplan. We are considering methods by which branch re-merging might be achieved, but all the methods we have considered so far seem to complicate the planning process considerably.

4.1.4 Uncertainties with multiple outcomes

Although the algorithm we have described can deal with uncertainties with any number of possible outcomes, we have so far discussed only examples with two possible outcomes. Let us therefore briefly consider an example in which more than two outcomes are possible. Suppose the planner has a goal to be in the room of a museum that contains a particular painting, and that it does not know in which of three rooms the painting is hung.

In this case the initial state will have three uncertain outcomes—*painting in room 1*, *painting in room 2*, and *painting in room 3*—stemming from a single source of uncertainty. In planning for the goal *agent in room and painting in room*, the planner must identify the condition *painting in room* with either *painting in room 1*, *painting in room 2*, or *painting in room 3*. Assume that it chooses room 1. This introduces uncertainty into the plan, leading to the introduction of two new contingencies, one for the outcome in which the painting is in room 2, and another for the outcome in which the painting is in room 3.

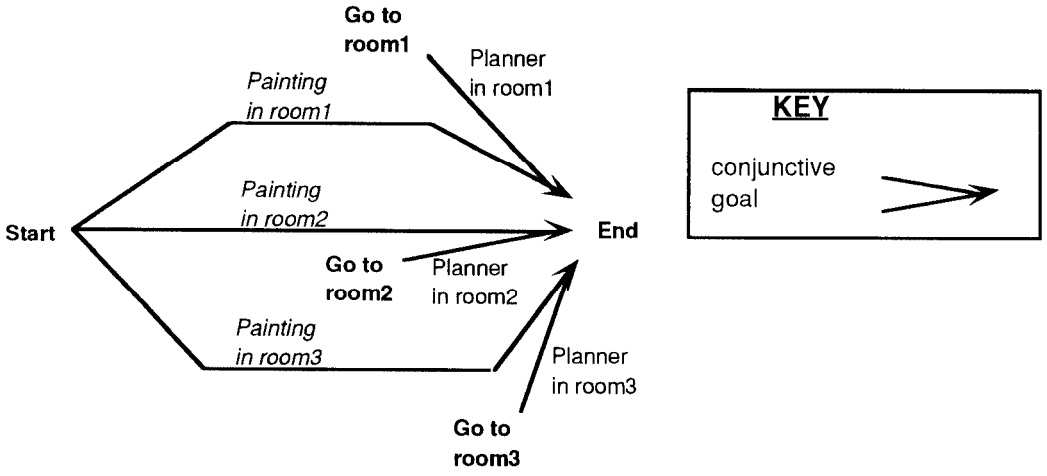


Figure 9 A source of uncertainty with three possible outcomes

In planning for each of these new contingencies, the planner must again establish the condition *painting in room*. In the contingency in which the painting is in room 2, the planner is prohibited from identifying this condition with *painting in room 1* or *painting in room 3*, so it must identify the condition with *painting in room 2*. Similarly, in the contingency in which the painting is in room 3, the condition painting in room can be established only by identification with *painting in room 3* (see figure 9).

4.1.5 Multiple sources of uncertainty

A plan may involve two or more sources of uncertainty. Cassandra's algorithm composes the various contingency plans created when this happens in a straightforward manner. For example, suppose the agent is given the goal of picking up a package that is at one of two locations, and that one of two cars will be available for it to use. Assume the planner first encounters the uncertainty about the location of the package. This will result in the construction of a plan with two contingencies. Call these contingencies A and B (see figure 10).

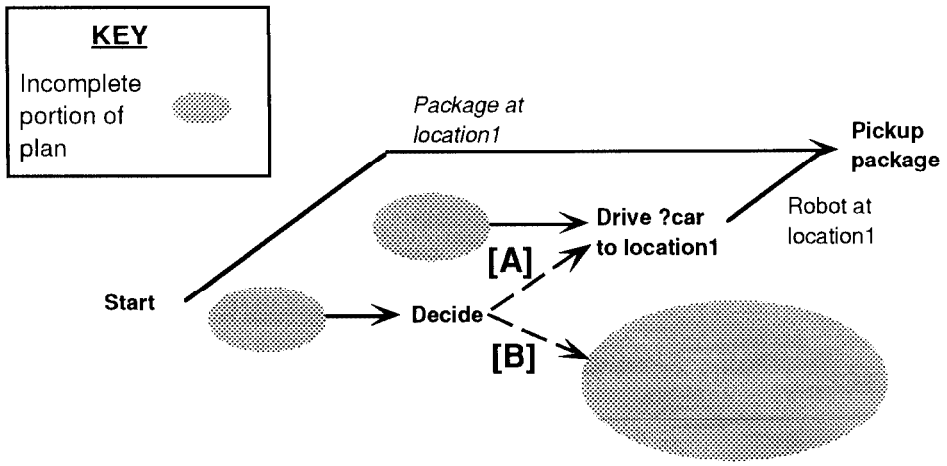


Figure 10 A partial plan to pick up a package

At some point during the construction of the plan for contingency A, the planner will encounter the uncertainty concerning which car will be available, and will make the current plan dependent upon one particular outcome of that uncertainty. Since the new source of uncertainty arises in the context of planning for contingency A, Cassandra must in effect bifurcate contingency A into contingency A1 (the package is at location 1 and car 1 is available) and contingency A2 (the package is at location 1 and car 2 is available). The planner must replace all existing contingency A labels with contingency A1 labels. It must then introduce a new copy of the top-level goal labeled with contingency A2 (see figure 11).

It is tempting to conclude that the planner should copy the original contingency A plan for use in contingency A2. However, doing so would result in a loss of completeness, since it is possible to construct plans in which very different methods are used to achieve the goal in A1 and A2. In fact, it could be necessary. For example, extreme differences between the two cars might affect the routes on which they could be driven or the places in which they could be parked.

Notice that the uncertainty concerning the availability of the cars does not necessarily affect contingency B. If the package were close enough that the agent

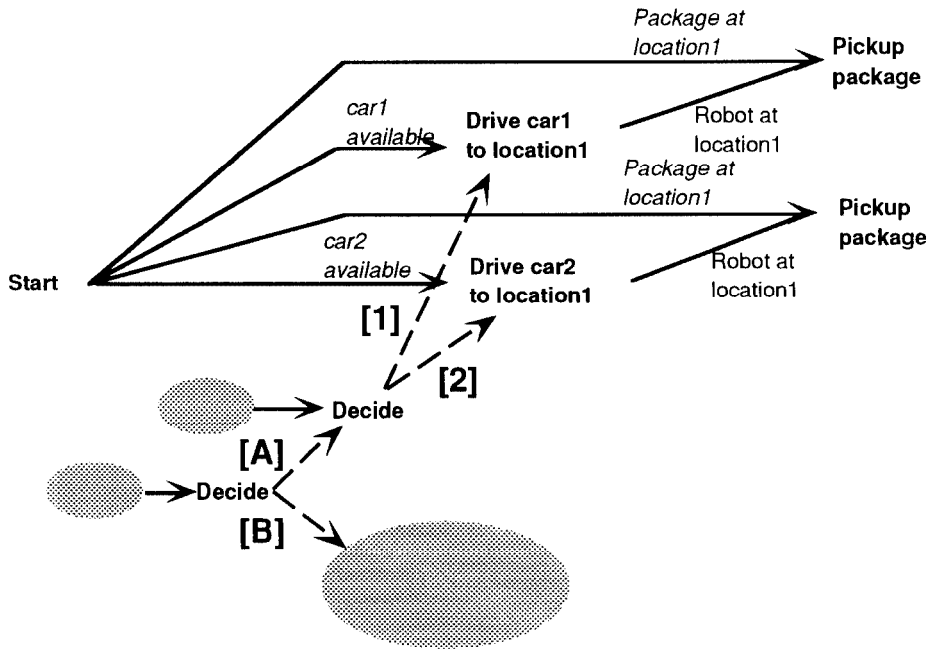


Figure 11 A plan with two sources of uncertainty

could get there without using a car, for example, the final plan might have only three contingencies: location 1 with car 1, location 1 with car 2, and location 2 (on foot). However, if the agent must drive to location 2 as well, the planner will eventually encounter the uncertainty over the availability of cars once again while planning for contingency B.

Although it has previously encountered this same source of uncertainty, the planner must recognize that this occurred in the context of a different contingency, and bifurcate contingency B in exactly the same way as was done previously for contingency A. In general, for a pair of uncertainties, if one uncertainty arises in every contingency of the other, the planner will generate one composite contingency for every member of the cross product of the possible outcomes of the uncertainties.

4.2 Decision steps

4.2.1 Adding a decision step to the plan

When Cassandra introduces a new contingency, it adds a decision step to the plan, which represents the act of determining which contingency plan should be carried out when the plan is executed. The decision step is added to the plan with the following ordering constraints:

- The decision step must occur after the step with which the uncertainty is associated.
- The decision step must occur before any step with a subgoal whose achievement depends on a particular outcome of the uncertainty.

4.2.2 Formulating decision rules

As discussed in a previous section, the action of deciding which contingency to execute can be modeled as the evaluation of a set of condition-action rules of the form:

If *condition1* then *contingency1*
If *condition2* then *contingency2*
If *condition3* then *contingency3*
....

Cassandra annotates each decision step in a plan with the set of decision rules that will be used to make that decision. These rules will be used by the agent during the execution of the plan.

In order to evaluate a decision rule, the agent must be able to determine whether the rule's antecedent holds. For each antecedent condition of a rule, therefore, the agent acquires a goal to know the current status of that condition.

These goals are the preconditions of the decision step. The preconditions of a decision step become open conditions in the plan in the same way as do the preconditions of any other step.

This raises the question of how Cassandra formulates the antecedent conditions for its decision rules. Since the intended effect of evaluating the rules is to choose the contingency that is appropriate given the outcome of a particular uncertainty, the conditions are effectively meant to check for which of the possible outcomes of the uncertainty actually occurred. This simple idea is made more complicated, however, because the agent cannot directly determine the outcome of an uncertainty, but must infer it from the presence or absence of effects that depend upon that outcome.

Cassandra adopts the approach of having the agent check those effects of a given outcome of an uncertainty that are actually used to establish preconditions in the contingency associated with that outcome. In other words, it seeks only to verify that the contingency plan can, in fact, succeed.[8] The condition part of a decision rule is thus a conjunction of all the direct effects of a particular outcome that are used to establish preconditions in the contingency plan for that outcome. We discuss how Cassandra constructs these rules below.

4.2.3 Adding a decision rule in our example

In the bomb-in-the-toilet example, the planner will introduce a decision step to determine whether the bomb is in package1 or not. Since the uncertainty is in the initial conditions, the decision will be constrained to occur after the *start* step. It must also occur before either of the *dunk* actions, since these depend upon par-

[8] It might seem that a more straightforward approach would involve checking all the effects of a given outcome. However, effects may depend upon other conditions. For example, consider our faulty drill. If the drill runs, it can have the effect of drilling a hole, but only if a bit is installed in the drill. Thus, the fact that a particular outcome occurs does not mean that all effects dependent upon that outcome will occur. However, all effects that establish preconditions in the contingency plan for that outcome must occur (or the plan will fail for other reasons).

ticular outcomes of the uncertainty. The *decide* step will have a precondition to know whether the bomb is in package1. If the planner has actions available that would allow it to determine this—X-raying the box, for example—it will fulfill this precondition with one of those actions, and decide on that basis which contingency of the plan to execute.

4.2.4 How Cassandra constructs decision rules

At the point at which Cassandra constructs a decision rule, only one precondition in the plan is known to depend upon a particular outcome of the uncertainty that gave rise to the decision, in particular, the one that led to Cassandra discovering the uncertainty in the first place. The decision-rule set that Cassandra builds thus looks like this:

If *effect1* then *contingency1*
If T then *contingency2*
If T then *contingency3*
...

Cassandra must modify this initial rule set each time an effect depending directly on the source of uncertainty is used to establish an open condition in the plan. In particular, Cassandra must determine the contingency in which that open condition resides, and conjoin the effect with the existing antecedent of the decision rule for that contingency.

Consider, for example, what happens when a coin is tossed. We might say that in theory there are three possible outcomes of this action: the coin can land flat with heads up, flat with tails up, or on its edge (figure 12). Suppose the planner is given a goal to have the coin be flat. This can be established by using the *flat-heads* effect of tossing it. Since this is an uncertain effect, Cassandra introduces two new contingencies into the plan, one for the outcome in which the coin lands tails up, and another for the outcome in which it lands on its edge. The in-

Action:	(toss-coin ?coin)	
Preconditions:	(holding ?agent ?coin)	
Effects:	(:when (:unknown U1 H) :effect (and (flat ?coin) (heads ?coin)))	*; secondary precondition*
	(:when (:unknown U1 T) :effect (and (flat ?coin) (tails ?coin)))	*; secondary precondition*
	(:when (:unknown U1 E) :effect (on-edge ?coin)))	*; secondary precondition*

Figure 12 Representing the action of tossing a coin

troduction of these contingencies mandates the introduction of a decision step. The initial rule set looks like this:

if *(flat coin)*	then [O1: H]	rule for heads up
if T	then [O1: T]	rule for tails up
if T	then [O1: E]	rule for edge

At the same time, a new open condition *(know-if (flat coin))* is introduced as a precondition of the decision, and new goal conditions are introduced that must be achieved in contingencies [O1: T] and [O1: E].

The planner next establishes the goal condition in contingency [O1: T]. The condition cannot be established by the *flat-heads* effect, because it arises in the incompatible contingency [O1: H]. However, the *flat-tails* effect can be used. The decision rule associated with the tails up contingency is thus modified as follows:

if *(flat coin)*	then [O1: H]	rule for heads up
if *(flat coin)*	then [O1: T]	rule for tails up
if T	then [O1: E]	rule for edge

Finally, the goal condition is established in contingency [O1: E]. Neither the flat-heads nor flat-tails effects can be used, as they are known to occur in incompatible contingencies. Instead, a new step is introduced into the plan: the

tip action is used to get the coin flat. The tip action requires the coin to be on edge, and this precondition can be established from the *on-edge* effect of the toss action. Since this effect depends directly upon the uncertainty O1, the decision rule for the edge contingency is modified as follows:

if *(flat coin)*	then [O1: H]	rule for heads up
if *(flat coin)*	then [O1: T]	rule for tails up
if *(on-edge coin)*	then [O1: E]	rule for edge

Since the plan is complete, this is the final decision rule. Notice that this rule does not discriminate the *heads up* outcome from the *tails up* outcome. Note that because the plans for these contingencies do not depend upon any uncertain effects other than the coin's being flat, the decision rule does not in fact discriminate between the outcome *heads up* and the outcome *tails up*. In fact, as discussed previously, either outcome will do, so there is no reason to make this discrimination. Which plan is executed in either of these conditions depends solely upon the order in which the agent that is executing the plan chooses to evaluate the decision rules.

A somewhat more complex problem arises if we give the planner the goal of having the coin be flat and heads up. In this case, the planner can establish both effects using the *toss* action. This will again lead to the introduction of two new contingencies into the plan, one for when the coin lands tails up, and one for when it lands on edge. Although the planner could establish *(flat coin)* in the tails up case, it would fail to complete the plan, because the coin would not be heads up. However, the planner could use the *turn over* action, which will leave the coin flat and heads up given that it was flat and tails up to begin with. At this point the decision rules are as follows:

if *(and (flat coin) (heads-up coin))*	then [O1: H]	rule for heads up
if *(and (flat coin) (tails-up coin))*	then [O1: T]	rule for tails up
if T	then [O1: E]	rule for edge

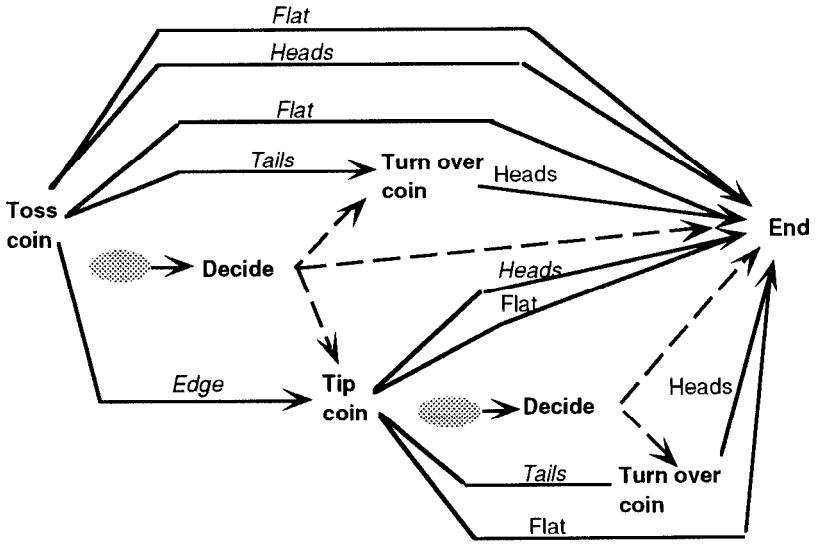


Figure 13 A plan with two decisions

The planner must then plan for the goal in the outcome in which the coin lands on its edge. The planner can establish both of these effects as a result of the *tip* action. However, the result *heads up* is an uncertain effect of the tip action, since the coin might just as easily land tails up. The planner must therefore add another new contingency for when the coin lands tails up after being tipped. In this instance, the goal can be established by using the turn over action, and the tails up precondition of this action can be established by the uncertain result of the tip action. The final decision rule set for the first decision is as follows:

if *(and (flat coin) (heads-up coin))*	then [O1: H]	rule for heads up
if *(and (flat coin) (tails-up coin))*	then [O1: T]	rule for tails up
if *(on-edge coin)*	then [O1: E]	rule for edge

If the on edge contingency is pursued, the planner must make another decision, this one stemming from the uncertain result of tip. Assuming that we name the second source of uncertainty O2, the rules for this decision are:

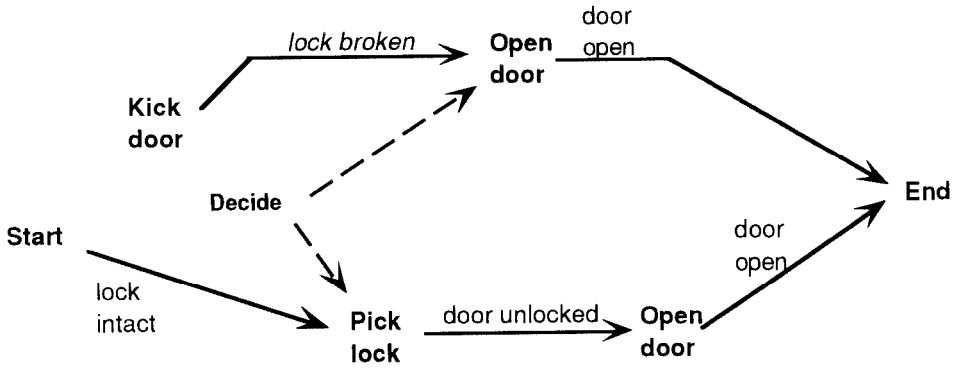


Figure 14 Opening a door

if *(heads-up coin)* then [O2: H]
if *(tails-up coin)* then [O2: T]

The plan is depicted in figure 13.

4.2.5 Decision rules and unsafe links

The fact that Cassandra allows decision rules that do not fully differentiate between outcomes of an uncertainty raises a somewhat subtle issue. Consider the partial plan for opening a locked door shown in figure 14. The action of kicking a door has, let us say, two possible outcomes, one in which the lock is broken and one in which the agent's foot is broken. A plan for the contingency in which the lock is broken is simply to open the door. A plan for the alternative contingency is to pick the lock and then open the door. Since the second plan does not depend causally on any outcome of the uncertainty (the agent's foot does not have to be broken in order for it to pick the lock and open the door), the decision rules based on the above discussion would be:

if *(lock-broken)*	then [O: L]	rule for lock broken
if T	then [O: F]	rule for foot broken

However, notice that in this case the *pick* action depends on the lock being intact, while the *kick* action may have the effect that the lock is no longer intact. In other words, the *kick* action potentially clobbers the precondition of *pick*. However, the planner can arguably ignore this clobbering, because the two actions belong to different contingencies. This is valid, however, only if the structure of the decision rules guarantees that the agent will not choose to execute the contingency involving *pick* when the outcome of *kick* is that the lock is broken. The decision rules above clearly do not enforce this. The solution in such a case is to augment the decision rule for the contingency in which the lock is not broken to test whether the lock is in fact intact. This results in the following decision rules:

if *(lock-broken)*	then [O: L]	rule for lock broken
if *(not (lock-broken))*	then [O: F]	rule for foot broken

Cassandra augments decision rules in this way whenever it notices that a direct effect of an uncertainty could clobber a link in a different contingency.

5 A contingency planning algorithm

In the previous section we discussed how Cassandra handles uncertainty by introducing contingency branches and decisions into the plan. We did not, however, consider in detail the modifications of the basic planning algorithm that are necessary to build such contingency plans. In this section we describe the necessary modifications to the basic algorithm, and consider the properties of the extended algorithm.

New step	Add a new step to the plan that has an effect that will establish the open condition. Add the step preconditions and the secondary preconditions of the effect as open conditions. The open condition becomes a completed link.
Reuse step	Make the open condition into a complete link from an effect of an existing plan step. *The effect must be possible (as indicated by its negative contingency labels) in every contingency in which the new link will be necessary (as indicated by the positive contingency labels of the open condition).* Add the secondary preconditions of the effect as open conditions.
Add contingency	*If the open condition is of type :unknown with a new source of uncertainty, add a new decision step to the plan. Modify one rule in the decision step. Add the new rule antecedents as open conditions. Add new goals as open conditions.*
Modify decision	*If the open condition is of type :unknown with an existing source of uncertainty, update the decision for that source of uncertainty by modifying one rule in the decision step. Add the new rule antecedents as open conditions. If necessary, add new goals as open conditions.*

Figure 15 Resolving open conditions in Cassandra

5.1 The extended algorithm

5.1.1 Open conditions

The introduction of uncertainty necessitates some changes in the procedures Cassandra uses to resolve open conditions (see figure 15—the changes from the basic UCPOP methods that were shown in figure 4 are in italics).

When Cassandra uses an effect of an existing step to establish an open condition, it must check that the effect can co-occur with the condition that it is to establish. This is done by inspecting the contingency labels of the open condition and the effect that is being considered. The establishing condition must have no negative label for any contingency for which the open condition has a positive label.

There is also a new type of open condition to be dealt with—the :unknown conditions that are used to represent uncertainty. Cassandra recognizes that un-

certainty is involved in a plan only when it encounters one of these special preconditions as an open condition. It then uses the procedures we described in section 4.2 to introduce a new decision step (if the source of uncertainty that is encountered is new) or to modify an existing decision step, and to introduce new goals if necessary.

Cassandra has one decision step for each source of uncertainty. When it encounters an open :unknown condition for which there is no existing decision step it introduces a new one which it then modifies, otherwise it simply modifies the old one. The rules in the decision must reflect the preconditions in the plan that depend on the outcome of this source of uncertainty. A precondition comes to depend on the uncertainty through depending on the effect with the :unknown secondary precondition. Cassandra therefore inspects all the links established by the effect, and ensures that the decision includes the appropriate rules. Whenever the effect is used to establish a new link, the :unknown precondition becomes a new open condition again, and when it is resolved the condition in the new link will be included in the rules, as described in section 4.2.2.

When a new antecedent is added to a rule, a new open condition is added as a precondition for the decision step. The open condition takes the form *(:know-if A)*, where A is the antecedent, and signifies that the agent must know the truth value of the antecedent.

Whenever Cassandra resolves an :unknown open condition, it also ensures that the necessary new goal conditions are introduced. All the goals that depend on the effects of that contingency are found. Any that do not mention the source of uncertainty concerned are duplicated as explained in section 4.1.5.

Cassandra must update the plan labeling whenever a new link is introduced into the plan. The new link and the step or effect established by the new link inherit the negative contingency labels of the link's establishing effect. The new link and its establishing effect inherit the positive contingency labels of the step or effect that the link establishes. These labels must be propagated through the

A link L is unsafe if there is an effect E in the plan (other than the effect LE1 that establishes the condition in the link and the effect LE2 that is either established or disabled by the link) with the following three properties:

Unification	One of the postconditions in E can possibly unify with the condition that L establishes or the negation of the condition that L establishes, *or if L establishes a :know-if condition, with that condition's argument or the negation of that condition's argument.*
Ordering	The step that produces E can, according to the partial order, occur both before the step that produces LE2 and after the step that produces LE1.
Labeling	*There is at least one contingency in which both E and L can occur.*

Figure 16 Unsafe links in Cassandra

plan to ensure that all elements in the plan have accurate labels. New open conditions must also be labeled appropriately. They inherit the positive contingency labels of the step or effect they will establish.

5.1.2 Unsafe links

There are two changes that must be made in the way the algorithm handles unsafe links. The definition of an unsafe link must be changed, and there is an additional method of resolving them. The changes are shown in italics in figures 16 and 17.

The definition of an unsafe link must be changed so as to allow for the :know-if preconditions of decision steps.[9] A link that establishes a condition of the form *(:know-if known-condition)* is threatened by any effect that changes the status of *known-condition*.

In contingency plans, a link is threatened by an effect only when the two can co-occur. Consider, for example, a source of uncertainty S with two outcomes, T and F. If an effect has a negative label [S : T], and a link has a negative label [S : F], then there is no contingency in which the effect threatens the link.

[9] Of course, other actions may also have :know-if preconditions.

Separation	Modify the variable bindings of the plan to ensure that the threatening effect E cannot in fact unify with the threatened condition.
Ordering	Modify the ordering of the steps in the plan to ensure that the step producing E occurs either before the step that produces LE1 or after the step that produces LE2.
Disable	*Make sure there is no contingency i n which the effect E and the link L can co-occur by one of:* • *Introduce a new open condition in the plan to disable E. This new open condition is the negation of E's secondary preconditions.* • *Add negative labels to the step producing E to forbid it to occur.* • *Add negative labels to L to signify that it cannot occur. Update the decision rules if appropriate.*

Figure 17 Resolving unsafe links in Cassandra

As well as changing the definition of an unsafe link, the introduction of contingencies suggests a new method of resolving unsafe links: simply forbid the link and the threatening effect to co-occur. This is a simple extension of the idea of disabling a threat by using the secondary preconditions of the threatening effect. In a contingency planner, the threat can be disabled in specific contingencies in three ways: by using secondary preconditions as before, by adding new negative labels to the step that produces the effect (to signify that it must not occur in that contingency), or by adding new negative labels to the link (to signify that it cannot occur in that contingency).

In the latter case, the possibility discussed in section 4.2.5 may arise. The contingency plan involving the protected link now depends on the source of uncertainty, and the decision rules must be updated appropriately.

5.2 Properties

Cassandra is a partial order planner directly descended from UCPOP, which is sound, complete, and systematic.

5.2.1 Soundness

Like UCPOP, Cassandra is sound: all plans that it constructs are guaranteed to achieve their goals. If no uncertainties are involved in the plan, Cassandra is in fact equivalent to UCPOP, and therefore constructs sound plans. When a source of uncertainty is introduced into the plan Cassandra is constructing, the procedures for adding in new goals ensure that the goal is achieved in every possible outcome of the uncertainty. Cassandra is thus sound if the outcomes of every source of uncertainty are fully specified.

5.2.2 Completeness

Cassandra is also complete: if there is a viable plan, Cassandra will find it. Again, this is a simple extension of UCPOP's completeness. If there are no uncertainties involved, Cassandra will always find a plan in the same way as UCPOP. The introduction of a source of uncertainty into a plan leads to the addition of new contingent goals. Cassandra will find a plan for each of these new goals in the appropriate contingency. Thus, if the goal can indeed be achieved in every contingency, Cassandra will find a plan that achieves it.

Fail-safe planning

Cassandra relies on being able to determine, even if only indirectly, the outcome of uncertainty. For example, the plan to disarm a bomb that we described in section 4.1 relies on there being a method of determining which package the bomb is in. In McDermott's (1987) presentation of this example, the two packages are indistinguishable. Even in this case there is a plan that will succeed in disarming the bomb: both packages should be dunked. Cassandra cannot, however, find this plan. The plan works because none of the actions that must be performed to achieve the goal in one contingency interfere with any of the actions that must be performed in the other contingency, and the ability to perform the actions is independent of the outcome of the uncertainty.

This suggests a method for constructing plans in the face of uncertainty when the outcome of the uncertainty cannot be determined (what one might call fail-safe plans). Whenever uncertainty arises it is in principle possible that there might be a noncontingent plan that would achieve the goal whatever the outcome of the uncertainty. To find such a plan, the planner must construct a version of the contingency plan that omits the decision for that source of uncertainty. All actions in the contingency branches arising from the uncertainty will therefore be executed unconditionally, and must therefore be able to be executed unconditionally. This means the planner cannot rely on the separation of contingency branches to stop actions clobbering each other, and it cannot rely on a particular outcome of the uncertainty to enable the execution of any of the actions. Apart from these changes, a fail-safe plan is like any other contingency plan.

Situations in which fail-safe plans are possible are relatively rare. Usually, the ability to perform the required actions does depend on the outcome of the uncertainty, and actions in the different contingency branches do interfere with each other. However, fail-safe plans, if they can be found, are very robust. Fail-safe plans are effectively contingency plans with their branches fully merged (see section 4.1.3).

Contingent failure

Cassandra can produce a plan only if it is possible to achieve the goal of the plan in all possible contingencies. Often, however, the goal cannot in fact be achieved in some outcome of the underlying uncertainty. Consider, for instance, Peot and Smith's (1992) example of trying to get to a ski resort by car, when the only road leading to the resort is either clear or blocked by snowdrifts. If the road is clear, then the goal can be achieved, but if it is blocked, all plans are doomed to failure.

No planner can be expected to recognize the impossibility of achieving a goal in the general case (Chapman, 1987). However, there is a possible heuristic approach, as suggested by Peot and Smith. We could introduce an alternative method of resolving open goal conditions: simply assume the goal in question

fails. If a suitably high penalty were assigned to plans with failed goals, such a plan would be pursued only after Cassandra failed to find a plan in which all goals were achieved.

This method of goal achievement should be used only with contingent goals at the top level, i.e., only with the restatements of the plan's goal that are introduced when a new source of uncertainty is encountered. Applying this constraint will ensure that only complete contingency branches will be allowed to fail.

Introducing this extension would, of course, result in loss of completeness. In setting the penalty to be applied to plans in which one branch is allowed to fail, an assumption must be made about the maximum possible cost of a viable plan. If there is in fact a viable plan that costs more than the penalty, Cassandra would no longer be guaranteed to find it.

5.2.3 Systematicity

UCPOP is in theory systematic (but see below): it will never visit the same partial plan twice while searching.

Cassandra, as described in this paper, is not systematic; it may visit some partial plans in the search space more than once. Consider again the plan to disarm a bomb that we discussed in section 4.1. In this plan, there are two different ways of establishing the goal to disarm the bomb: by dunking package 1, and by dunking package 2. The two methods of establishing the goal are used in two different contingencies. Cassandra can initially choose either way of establishing the goal, leading in each case to the introduction of a contingency and the necessity of replanning to achieve the goal in the other contingency. Both search paths arrive at the same final plan, so the search is not systematic.

Cassandra could be made systematic by insisting on handling the contingencies only in a certain order, the search path that uses the other order being treated as a dead end. There is currently some uncertainty as to the desirability of sys-

tematicity (Langley, 1992; Peot & Smith, 1992), and so this extension has not been added.

The systematicity of UCPOP depends on the exact mechanism used to implement the resolution of unsafe links. In order to be truly systematic, UCPOP must ensure that only one method of protecting a link is used. For example, if it chooses to protect a link by separation (ensuring that the conditions cannot unify) it must then add ordering constraints to ensure that the clobbering condition is forced to occur between the beginning and end of the threatened link, and must also add the effect's secondary preconditions as open conditions to ensure that the effect will in fact happen. This ensures that the search space is genuinely divided into non-overlapping subspaces. If this is not done, the same plan could be encountered twice while searching.

For example, suppose a link L is threatened. Assume there are two ways in which it can be protected: separation and reordering. Let the two partial plans thus derived be P_S and P_O. Suppose that later on in the search process a partial plan derived from P_S has ordering constraints added that have the same effect as the reordering that resulted in P_O, and that a partial plan derived from P_O has codesignation constraints added that have the same effect as the separation that resulted in P_S. If the other modifications to the two partial plans are exactly equivalent (which is perfectly possible), we now have identical partial plans on two different search paths.

The method of handling unsafe links in the current implementation of Cassandra does not preserve systematicity in this way, and we believe the same is true of UCPOP. It would not be difficult to add this mechanism to either planner.

6 Related work

Cassandra is constructed using UCPOP (Penberthy & Weld, 1992) as a platform. UCPOP is a partial order planner that handles actions that have context-dependent effects and universally quantified preconditions and effects. UCPOP is an extension of SNLP (Barrett, et al., 1991; Barrett & Weld, 1993; McAllester & Rosenblitt, 1991) that uses a subset of Pednault's ADL representation (Pednault, 1988; 1989; 1991).

An early contingency planner was WARPLAN-C (Warren, 1976). Contingency planning was more or less abandoned between the mid seventies and the early nineties, until SENSp (Etzioni, et al., 1992) and CNLP (Peot & Smith, 1992). Both SENSp and CNLP are members of the SNLP family: SENSp is, like Cassandra, based on UCPOP, and CNLP is based directly on SNLP.

Not surprisingly, Cassandra, SENSp and CNLP are in many respects very similar. All three use the basic algorithm from SNLP, and all use extended STRIPS representations. However, each modifies these basic components in a different way. In this section we briefly discuss the similarities and differences between these three systems.

6.1 Representation

Because SENSp and Cassandra are based on UCPOP, they both use secondary preconditions to represent context-dependent effects. CNLP does not have this extension to the basic STRIPS representation. Cassandra uses a minor modification of this extension to represent both the uncertain causal effects of actions and unknown initial conditions.

Uncertainty does not appear to be represented explicitly in SENSp. Instead, a distinction is made between ordinary variables and *run-time* variables. Run-time variables are treated as constants whose values are not yet known: in

Cassandra's terms, they can be seen as variables whose values depend on uncertain outcomes. It is not clear from (Etzioni, et al., 1992) whether this scheme can handle the uncertain effects of actions as well as unknown initial conditions.

In CNLP, uncertainty is represented through a combination of uncertain outcomes of actions and the effects of observing the outcome. A three-valued logic is used for conditions: a condition may be true, false, or unknown. For example, the action of tossing a coin would have the postcondition *unk(side-up ?x)*. Peot and Smith then introduce special *conditional* actions with an unknown precondition and several mutually exclusive sets of postconditions. In this example, the operator *observe* would have the precondition *unk(side-up ?x)* with three possible outcomes: *(side-up heads)*,*(side-up tails)*, and *(side-up edge)*.

The representation used in CNLP thus represents the uncertainty as stemming from the action that observes the result, rather than from the action that in fact produces the uncertainty. One consequence of this is that CNLP cannot use the same observation action to observe the results of different actions. For example, CNLP would require different actions to observe the results of tossing a coin (which has three possible outcomes) and tipping a coin that had landed on its edge (which has two possible outcomes). We believe this is a serious drawback in CNLP.

6.2 Knowledge acquisition

An early and influential discussion of knowledge goals was by McCarthy and Hayes (1969). Morgenstern (1987) draws attention to the ability of an agent to execute a plan if it can "make sure" that all the events in the plan are executable: this is the approach we have used in formulating Cassandra's decision rules. Steel (1993) points out that it is not always possible for the agent executing a plan to distinguish the different outcomes of an uncertainty directly.

Cassandra is, we believe, the first planner in which decisions are represented as explicit actions in the plans that it constructs. Knowledge goals are not represented as arising specifically from the need to decide between alternative branches of the plan in either SENSP or CNLP. As a result, neither distinguishes effectively between sensing or information-gathering actions on the one hand, and decision making on the other.

Actions that achieve knowledge goals may have preconditions in both CNLP and Cassandra. However, they may not have preconditions in SENSP: this restriction is required in order to maintain completeness. We believe this is a serious limitation. We have argued elsewhere (Pryor, 1993; Pryor & Collins, 1991; 1992) that knowledge acquisition is a planning task like any other. There is no reason to assume that knowledge goals can always be achieved by a single action that has no preconditions.

The confusion between the source of uncertainty and the observation of uncertain results also limits the ways in which knowledge goals can be achieved in CNLP: they must be achieved through the special observation actions that specify the uncertain outcomes. In CNLP, there is no way to distinguish the need to know whether a particular plan branch will work from the need to know the actual outcome of an uncertainty, as there is in Cassandra.

6.3 Plan branching

Cassandra represents plan branches through its labeling scheme, which is similar to that used by CNLP. CNLP'S scheme is somewhat simpler than Cassandra's in that it attaches labels only to plan steps. Cassandra attaches labels to effects and links as well as to steps. This is necessary in a system that uses secondary preconditions to represent context-dependent effects.

The use of labels allows Cassandra, like CNLP, to consider contingency branches in parallel rather than separately. SENSP, in contrast, constructs sepa-

rate plans that each achieve the goal in a particular contingency. It then combines the separate plans at a later stage, keeping the branches totally separate.

In SENSP plan branches arise from the introduction of observation steps that bind the run-time variables. In CNLP plan branches arise from the use of an outcome of the observation step in which the uncertainty is represented. In both these planners, therefore, plan branches arise from the introduction of steps that have the effect of achieving knowledge goals. In Cassandra plan branches arise from the use of an uncertain outcome to establish a subgoal in the plan, and knowledge goals arise from the need to decide between branches. This is a fundamental difference between Cassandra and the other two planners.

The simple representation of uncertainty and the explicit representation of decisions in Cassandra thus allow a coherent approach to the problems of contingent planning. Both SENSP and CNLP are limited by their representations,[10] which both incorporate a certain degree of conceptual confusion. The representations used in these two planners limit the types of plans that may be constructed: in particular, neither of them allow full planning for information goals.

7 Conclusions

Cassandra is a sound and complete partial-order contingency planner that can represent uncertain outcomes and construct contingency plans for those outcomes. It can construct plans in situations which have caused problems for previous systems. Cassandra is equally effective at handling actions with uncertain effects and unknown initial conditions. Cassandra explicitly plans to gather information and allows information-gathering actions to have a full range of preconditions. The coherence of its design leads to simpler plans than those produced by other planners, and provides a solid base for more advanced capabilities such as the use of varying decision-making procedures.

---

[10] See (Collins & Pryor, 1993) for a discussion of other problems in the representation used by SENSP.

Acknowledgments

Thanks to Dan Weld and Tony Barrett for supplying the UCPOP code, Mark Peot for his comments on an early draft, and Will Fitzgerald for many useful discussions.

References

Allen, J., Hendler, J., & Tate, A. (Ed.). (1990). *Readings in Planning*. San Mateo, CA: Morgan Kaufmann.

Barrett, A., Soderland, S., & Weld, D. S. (1991). *Effect of Step-Order Representations on Planning* Technical Report 91-05-06. Department of Computer Science and Engineering, University of Washington, Seattle.

Barrett, A., & Weld, D. S. (1993). Partial-Order Planning: Evaluating Possible Efficiency Gains. *Artificial Intelligence*(Forthcoming)

Chapman, D. (1987). Planning for Conjunctive Goals. *Artificial Intelligence, 32*, 333-337. Also in (Allen, et al., 1990).

Collins, G., & Pryor, L. (1992a). Achieving the functionality of filter conditions in a partial order planner. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, San Jose, CA: AAAI.

Collins, G., & Pryor, L. (1992b). *Representation and performance in a partial order planner* Technical Report # 35. The Institute for the Learning Sciences.

Collins, G., & Pryor, L. (1993). What are filter conditions for? In *Proceedings of the Working notes of the AAAI Spring Symposium. Foundations of Automatic Planning: The Classical Approach and Beyond.*, : AAAI.

Etzioni, O., Hanks, S., Weld, D., Draper, D., Lesh, N., & Williamson, M. (1992). An approach to planning with incomplete information. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, Boston, MA: Morgan Kaufmann.

Feldman, J. A., & Sproull, R. F. (1977). Decision Theory and Artificial Intelligence II: The Hungry Monkey. *Cognitive Science, 1*, 158-192. Also in (Allen, et al., 1990).

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence, 2*, 189-208. Also in (Allen, et al., 1990).

Hanks, S., & Weld, D. S. (1992). Systematic adaptation for case-based planning. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, (pp. 96 - 105). College Park, Maryland: Morgan Kaufmann.

Langley, P. (1992). Systematic and Nonsystematic Search Strategies. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, College Park, Maryland: Morgan Kaufmann.

McAllester, D., & Rosenblitt, D. (1991). Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, (pp. 634-639). Anaheim, CA: AAAI.

McCarthy, J., & Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer & D. Michie (Eds.), *Machine Intelligence 4* (pp. 463-502). Edinburgh University Press. Also in (Allen, et al., 1990).

McDermott, D. V. (1987). A Critique of Pure Reason. *Computational Intelligence, 3*, 151-160.

Morgenstern, L. (1987). Knowledge Preconditions for Actions and Plans. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan: IJCAI.

Pednault, E. P. D. (1988). Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence, 4*, 356-372.

Pednault, E. P. D. (1989). ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*,

Pednault, E. P. D. (1991). Generalizing nonlinear planning to handle complex goals and actions with context-dependent effects. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, Sydney, Australia

Penberthy, J. S., & Weld, D. S. (1992). UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, Boston, MA: Morgan Kaufmann.

Peot, M. A., & Smith, D. E. (1992). Conditional Nonlinear Planning. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, College Park, Maryland: Morgan Kaufmann.

Pryor, L. (1993) *Decisions and Information in Unpredictable Worlds*. PhD dissertation, in preparation, The Institute for the Learning Sciences, Northwestern University.

Pryor, L., & Collins, G. (1991). Information gathering as a planning task. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, Chicago, IL: Lawrence Erlbaum Associates.

Pryor, L., & Collins, G. (1992). Planning to perceive: A utilitarian approach. In *Proceedings of the AAAI 1992 Spring Symposium on Control of Selective Perception*,

Sacerdoti, E. (1977). *A structure for plans and behavior*. New York: American Elsevier.

Steel, S. (1993). A connection between decision theory and program logic. In A. Sloman, D. Hogg, G. Humphreys, A. Ramsay, & D. Partridge (Eds.), *Prospects for Artificial Intelligence: Proceedings of the Ninth biennial conference of the Society for the Study of Artificial Intelligence and the Simulation of Behaviour* (pp. 281 - 290). Amsterdam: IOS Press.

Stefik, M. (1981). Planning with constraints (MOLGEN: Part 2). *Artificial Intelligence, 16*, 141-170.

Sussman, G. J. (1975). *A computer model of skill acquisition*. New York: American Elsevier.

Tate, A. (1977). Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA: IJCAI. Also in (Allen, et al., 1990).

Warren, D. (1976). Generating conditional plans and programs. In *Proceedings of the AISB Summer Conference*, University of Edinburgh

Wilkins, D. E. (1988). *Practical Planning: Extending the Classical AI Planning Paradigm*. San Mateo, CA: Morgan Kaufmann.

Appendix: Cassandra's plans

This appendix shows the plans constructed by Cassandra for the examples in sections 2, 3, and 4. Each plan consists of initial conditions, plan steps and goals. Cassandra's output is in sans serif: comments are in *italics*.

The initial conditions are shown at the top of the plan. Those that are unknown are shown as depending on a particular contingency. The plan steps are shown next. Each is shown with a number denoting its order in the plan. The numbers in parentheses show the order in which the steps were added to the plan. To the right of each step are its contingency labels. For brevity, we omit the individual effects of each step or the links that establish the step's enabling and secondary preconditions.

Finally, at the bottom of the plan come the goal conditions. First, the goal is stated. Then, each contingency goal is shown with the links that establish it. As usual, contingency labels are to the right.

A.1 A plan to get to Evanston

This is the plan shown in figure 3 and discussed in section 2.4. Note the decision step with a single active decision rule. This is the situation discussed in the comments on one-sided contingencies in section 4.1.3: the route using Western is quicker when it is clear, while the Ashland route is slower but always possible.

```
Initial:  When [TRAFFIC0S: GOOD]      ; An unknown initial condition. The source of uncertainty
            (NOT (TRAFFIC-BAD))       ; is TRAFFIC0S. The outcome on which this effect
                                      ; depends is GOOD
          When [TRAFFIC0S: BAD]     (TRAFFIC-BAD)

          (AND (AT START) (ROAD WESTERN) (ROAD BELMONT) (ROAD ASHLAND))
                                      ; Other initial conditions
Step 1 (4):   (GO-TO-WESTERN-AT-BELMONT)     YES: [TRAFFIC0S: GOOD BAD]

Step 2 (3):   (CHECK-TRAFFIC-ON-WESTERN)

Step 3 (2):   (DECIDE TRAFFIC0S)                  ; A decision step
                 (and  (NOT (TRAFFIC-BAD))        ; A decision rule
                       T          ) => [TRAFFIC0S: GOOD]
                  (and T          ) => [TRAFFIC0S: BAD]

Step 4 (6):   (TAKE-BELMONT)                      YES:  [TRAFFIC0S: BAD]
                                                  NO :  [TRAFFIC0S: GOOD]
                                      ; This step must be executed in the traffic is BAD. It either
                                      ; cannot or must not be executed if the traffic is GOOD
Step 5 (5):   (TAKE-ASHLAND)                      YES:  [TRAFFIC0S: BAD]
                                                  NO :  [TRAFFIC0S: GOOD]

Step 6 (1):   (TAKE-WESTERN)                      YES:  [TRAFFIC0S: GOOD]
                                                  NO :  [TRAFFIC0S: BAD]

Goal:         (AT EVANSTON)           ; The goal statement

          GOAL                                    YES:  [TRAFFIC0S: BAD]
                                      ; A contingency goal in case the traffic is BAD
             5 -> (AT EVANSTON)                   NO :  [TRAFFIC0S: GOOD]

          GOAL                                    YES:  [TRAFFIC0S: GOOD]
             6 -> (AT EVANSTON)                   NO :  [TRAFFIC0S: BAD]
Complete!
```

Figure A.1 Getting to Evanston

A.2 Disarming a bomb

This is the plan shown in figures 7 and 8 and discussed in section 4.1. Note that both moving steps and both dunking steps are always possible, but each is only necessary in one outcome of the uncertainty. A fail-safe plan (see section 5.2.2) is therefore possible.

```
Initial:      When [UNK0S: O2]    (CONTAINS PACKAGE-2 BOMB)
              When [UNK0S: O1]    (CONTAINS PACKAGE-1 BOMB)
              (AND (AT PACKAGE-1 RUG) (AT PACKAGE-2 RUG))

Step 1 (5):   (X-RAY PACKAGE-2)

Step 2 (3):   (X-RAY PACKAGE-1)

Step 3 (2):   (DECIDE UNK0S)
                 (and (CONTAINS PACKAGE-2 BOMB)
                      T          ) => [UNK0S: O2]
                 (and (CONTAINS PACKAGE-1 BOMB)
                      T          ) => [UNK0S: O1]

Step 4 (7):   (MOVE RUG TOILET PACKAGE-1)          YES: [UNK0S: O1]

Step 5 (6):   (MOVE RUG TOILET PACKAGE-2)          YES: [UNK0S: O2]

Step 6 (4):   (DUNK PACKAGE-2)                     YES: [UNK0S: O2]

Step 7 (1):   (DUNK PACKAGE-1)                     YES: [UNK0S: O1]

Goal:         (DISARMED BOMB)

      GOAL                                         YES: [UNK0S: O2]
        6 -> (DISARMED BOMB)                       NO : [UNK0S: O1]

      GOAL                                         YES: [UNK0S: O1]
        7 -> (DISARMED BOMB)                       NO : [UNK0S: O2]
Complete!
```

Figure A.2 Disarming a bomb

A.3 Finding a painting

This is the plan shown in figure 9 and discussed in section 4.1.4. Note that there are three possibilities for the location of the picture, and therefore three decision rules. Again, it is possible to go to any room in any contingency, but it is only necessary to go to one room in each contingency.

```
Initial:      When [PICTOS: R3]    (PAINTING-IN ROOM-3)
              When [PICTOS: R2]    (PAINTING-IN ROOM-2)
              When [PICTOS: R1]    (PAINTING-IN ROOM-1)

Step 1 (2):   (ASK-ATTENDANT)

Step 2 (1):   (DECIDE PICTOS)
                 (and (PAINTING-IN ROOM-2)
                      T          ) => [PICTOS: R2]
                 (and (PAINTING-IN ROOM-3)
                      T          ) => [PICTOS: R3]
                 (and (PAINTING-IN ROOM-1)
                      T          ) => [PICTOS: R1]

Step 3 (5):   (GO-TO ROOM-1)              YES:  [PICTOS: R1]

Step 4 (4):   (GO-TO ROOM-2)              YES:  [PICTOS: R2]

Step 5 (3):   (GO-TO ROOM-3)              YES:  [PICTOS: R3]

Goal:     (AND (IN ?ROOM) (PAINTING-IN ?ROOM))

          GOAL                                YES:  [PICTOS: R3]
             5 -> (IN ROOM-3)
             0 -> (PAINTING-IN ROOM-3)        NO :  [PICTOS: R1 R2]

          GOAL                                YES:  [PICTOS: R2]
             4 -> (IN ROOM-2)
             0 -> (PAINTING-IN ROOM-2)        NO :  [PICTOS: R1 R3]

          GOAL                                YES:  [PICTOS: R1]
             3 -> (IN ROOM-1)
             0 -> (PAINTING-IN ROOM-1)        NO :  [PICTOS: R2 R3]

Complete!
```

Figure A.3 Finding a painting

A.4 Fetching a package

The plan in figure 10 involves one source of uncertainty: it is shown in figure A.4a. Figure A.4b shows the plan in figure 11, which has two sources of uncertainty. These plans are discussed in section 4.1.5.

The plan in figure A.4a has just one source of uncertainty, and only one decision. There are two possible ways of achieving the goal. The plan in figure A.4b has two decisions, and four possible ways of achieving the goal, one for each combination of the outcomes of the two sources of uncertainty.

```
Initial:  (AVAILABLE CAR-1)
          When [LOC0S: B]   (PACKAGE-AT LOCATION-2)
          When [LOC0S: A]   (PACKAGE-AT LOCATION-1)

Step  1 (2):    (ASK-ABOUT-PACKAGE)

Step  2 (1):    (DECIDE LOC0S)
                    (and (PACKAGE-AT LOCATION-2)
                         T            ) => [LOC0S: B]
                    (and (PACKAGE-AT LOCATION-1)
                         T            ) => [LOC0S: A]

Step  3 (4):    (DRIVE CAR-1 LOCATION-1)              YES:  [LOC0S: A]

Step  4 (3):    (DRIVE CAR-1 LOCATION-2)              YES:  [LOC0S: B]

Goal:       (AND (AT ?LOC) (PACKAGE-AT ?LOC))

          GOAL                                        YES:  [LOC0S: B]
             4 -> (AT LOCATION-2)
             0 -> (PACKAGE-AT LOCATION-2)             NO :  [LOC0S: A]

          GOAL                                        YES:  [LOC0S: A]
             3 -> (AT LOCATION-1)
             0 -> (PACKAGE-AT LOCATION-1)             NO :  [LOC0S: B]

Complete!
```

Figure A.4a Fetching a package—one uncertainty

```
Initial:    When [CAR0S: C2]    (AVAILABLE CAR-2)
            When [CAR0S: C1]    (AVAILABLE CAR-1)
            When [LOC0S: B]     (PACKAGE-AT LOCATION-2)
            When [LOC0S: A]     (PACKAGE-AT LOCATION-1)

Step 1 (5):   (ASK-ABOUT-CAR)                    YES: [LOC0S: A B]

Step 2 (4):   (DECIDE CAR0S)                     YES: [LOC0S: A B]
                 (and (AVAILABLE CAR-2)
                      T          ) => [CAR0S: C2]
                 (and (AVAILABLE CAR-1)
                      T          ) => [CAR0S: C1]

Step 3 (2):   (ASK-ABOUT-PACKAGE)                YES: [CAR0S: C2 C1]

Step 4 (1):   (DECIDE LOC0S)                     YES: [CAR0S: C2 C1]
                 (and (PACKAGE-AT LOCATION-2)
                      T          ) => [LOC0S: B]
                 (and (PACKAGE-AT LOCATION-1)
                      T          ) => [LOC0S: A]

Step 5 (8):   (DRIVE CAR-2 LOCATION-1)           YES: [LOC0S: A][CAR0S: C2]
                                                 NO :  [CAR0S: C1]

Step 6 (6):   (DRIVE CAR-2 LOCATION-2)           YES: [LOC0S: B][CAR0S: C2]
                                                 NO :  [CAR0S: C1]

Step 7 (7):   (DRIVE CAR-1 LOCATION-1)           YES: [LOC0S: A][CAR0S: C1]
                                                 NO :  [CAR0S: C2]

Step 8 (3):   (DRIVE CAR-1 LOCATION-2)           YES: [LOC0S: B][CAR0S: C1]
                                                 NO :  [CAR0S: C2]

Goal:    (AND (AT ?LOC) (PACKAGE-AT ?LOC))

         GOAL                                    YES: [LOC0S: A][CAR0S: C2]
            5 -> (AT LOCATION-1)                 NO :  [CAR0S: C1]
            0 -> (PACKAGE-AT LOCATION-1)         NO :  [LOC0S: B]

         GOAL                                    YES: [LOC0S: B][CAR0S: C2]
            6 -> (AT LOCATION-2)                 NO :  [CAR0S: C1]
            0 -> (PACKAGE-AT LOCATION-2)         NO :  [LOC0S: A]

         GOAL                                    YES: [LOC0S: B][CAR0S: C1]
            8 -> (AT LOCATION-2)                 NO :  [CAR0S: C2]
            0 -> (PACKAGE-AT LOCATION-2)         NO :  [LOC0S: A]

         GOAL                                    YES: [LOC0S: A][CAR0S: C1]
            7 -> (AT LOCATION-1)                 NO :  [CAR0S: C2]
            0 -> (PACKAGE-AT LOCATION-1)         NO :  [LOC0S: B]
Complete!
```

Figure A.4b Fetching a package—two uncertainties

A.5 Tossing a coin

In section 4.2.4 we described a plan for ending up with a flat coin: this is shown in figure A.5a. The plan in figure 13, with two decisions, is shown in figure A.5b. The plan in figure A.5b omits information acquisition steps for brevity.

The decision in the first plan does not distinguish between the coin landing head-up and tails-up. The decision in the second plan has extra conjuncts in the decision rules. Note that there are four ways of achieving the goal in second plan, due to two sources of uncertainty.

```
Initial:  (HOLDING-COIN)

Step 1 (2):   (TOSS-COIN)

Step 2 (5):   (INSPECT-COIN)

Step 3 (4):   (INSPECT-COIN)

Step 4 (3):   (DECIDE TOSS2S)
                 (and (FLAT-COIN)
                      T          ) => [TOSS2S: H]
                 (and (FLAT-COIN)
                      T          ) => [TOSS2S: T]
                 (and (ON-EDGE)
                      T          ) => [TOSS2S: E]

Step 5 (1):   (TIP-COIN)                          YES:  [TOSS2S: E]
                                                  NO :  [TOSS2S: H T]

Goal:     (FLAT-COIN)

        GOAL                                      YES:  [TOSS2S: T]
          1 -> (FLAT-COIN)                        NO :  [TOSS2S: H E]

        GOAL                                      YES:  [TOSS2S: H]
          1 -> (FLAT-COIN)                        NO :  [TOSS2S: T E]

        GOAL                                      YES:  [TOSS2S: E]
          5 -> (FLAT-COIN)                        NO :  [TOSS2S: H T]

Complete!
```

Figure A.5a Tossing a coin—ambiguous decision rules

```
Initial:        (HOLDING-COIN)

Step 1 (1):     (TOSS-COIN)

Step 2 (2):     (DECIDE TOSS1S)
                   (and  (FLAT-COIN)
                         (HEADS-UP)
                         T          ) => [TOSS1S: H]
                   (and  (ON-EDGE)
                         T          ) => [TOSS1S: E]
                   (and  (FLAT-COIN)
                         (TAILS-UP)
                         T          ) => [TOSS1S: T]

Step 3 (4):     (TIP-COIN)                          YES: [TOSS1S: E]
                                                    NO : [TOSS1S: T H]

Step 4 (5):     (DECIDE TIP4S)                      YES: [TOSS1S: E]
                                                    NO : [TOSS1S: T H]
                   (and  (TAILS-UP)
                         T          ) => [TIP4S: T]
                   (and  (HEADS-UP)
                         T          ) => [TIP4S: H]

Step 5 (3): (TURN-OVER)                             YES: [TOSS1S: T]
                                                    NO : [TOSS1S: E H]

Step 6 (6): (TURN-OVER)                             YES: [TOSS1S: E][TIP4S: T]
                                                    NO : [TOSS1S: T H][TIP4S: H]

Goal:     (AND (FLAT-COIN) (HEADS-UP))

        GOAL                                        YES: [TOSS1S: E][TIP4S: T]
           3 -> (FLAT-COIN)                         NO : [TOSS1S: T H]
           6 -> (HEADS-UP)                          NO : [TOSS1S: T H][TIP4S: H]

        GOAL                                        YES: [TOSS1S: E][TIP4S: H]
           3 -> (FLAT-COIN)                         NO : [TOSS1S: T H]
           3 -> (HEADS-UP)                          NO : [TOSS1S: H T][TIP4S: T]

        GOAL                                        YES: [TOSS1S: T]
           1 -> (FLAT-COIN)                         NO : [TOSS1S: H E]
           5 -> (HEADS-UP)                          NO : [TOSS1S: E H]

        GOAL                                        YES: [TOSS1S: H]
           1 -> (FLAT-COIN)                         NO : [TOSS1S: T E]
           1 -> (HEADS-UP)                          NO : [TOSS1S: T E]
Complete!
```

Figure A.5b Tossing a coin—distinct decision rules

A.6 Opening a door

In section 4.2.4 we described a plan for opening a locked door without a key, and illustrated it in figure 14. The plan that Cassandra produces for this situation is shown in figure A.6. Even though no preconditions of the *pick* step depend on any effect of the *kick* step, the step cannot be performed if the lock is broken as a result of kicking the door. The decision rules reflect this dependence.

```
Initial:        (LOCK-INTACT)

Step 1 (2):     (KICK)

Step 2 (4):     (LOOK)

Step 3 (3):     (DECIDE KICK2S)
                   (and  (LOCK-INTACT)
                         T          ) => [KICK2S: F]
                   (and  (NOT (LOCKED))
                         T          ) => [KICK2S: L]

Step 4 (6):     (PICK)                          YES: [KICK2S: F]
                                                NO : [KICK2S: L]

Step 5 (5):     (OPEN-DOOR)                     YES: [KICK2S: F]
                                                NO : [KICK2S: L]

Step 6 (1):     (OPEN-DOOR)                     YES: [KICK2S: L]
                                                NO : [KICK2S: F]

Goal:     (OPEN)

        GOAL                                    YES: [KICK2S: F]
           5 -> (OPEN)                          NO : [KICK2S: L]

        GOAL                                    YES: [KICK2S: L]
           6 -> (OPEN)                          NO : [KICK2S: F]

Complete!
```

Figure A.6 Opening a door