Some Logical Conundrums for Decision-Theoretic Contingency Planning

John L. Pollock
Department of Philosophy
University of Arizona
Tucson, Arizona 85721
pollock@arizona.edu
http://www.u.arizona.edu/~pollock

Abstract

There are two general approaches to handling contingencies in decision-theoretic planning. State-space planners reason globally, building a map of the parts of the world relevant to the planning problem, and then attempt to distill a plan out of the map. POCL planners reason locally, attempting to build the plan up from local relationships. A planning problem is constructed that humans find trivial, but no state-space planner can solve. This motivates an investigation of decision-theoretic POCL contingency planners. Existing POCL contingency planners attempt to generalize the results of classical POCL contingency planning. However, this paper argues that the nature of contingency planning changes dramatically in decision-theoretic contexts, and results from classical contingency planning are of little relevance. In particular, in classical planning contingencies can only be attached to conditional forks, but in most uses of contingencies in decision-theoretic planning they are attached to single branches of the plan rather than to conditional forks. A criterion of adequacy for contingency planners is formulated, following from ordinary completeness, and it is shown that existing decision-theoretic POCL contingency planners do not satisfy it. Some tentative suggestions are made regarding how to construct a planner that does satisfy the adequacy condition.

1. Introduction

Collins (1987) remarks that planning is the process of deciding what to do in advance of when it can be done. However, some decisions are best left to the time the plan is executed. These are decisions that require knowledge the agent cannot have until the time of execution. In the face of such ignorance, we can still engage in advance planning, but what we plan to do must be made conditional on various contingencies occurring, and so the plan will prescribe different actions depending upon what we know at execution time. Such plans are *contingency plans*, and the process of constructing them is *contingency planning* (also called *conditional planning*).

Contingency planning was first investigated in the context of classical POCL (partial-order causal-link) planning, where it was assumed that the only source of uncertainty is our knowledge of initial conditions. Classical contingency planning retains the assumption that actions have their consequences deterministically, and the objective of planning is to construct a plan that is guaranteed to achieve its goals. Classical contingency planning is now fairly well understood, and various researchers have taken up the task of extending the results of classical contingency planning to decision-theoretic contexts. This paper investigates some of the logical foundations of decision-theoretic contingency planning. Its main purpose is to argue that existing algorithms

This work was supported by NSF grant no. IRI-IIS-0080888.

for decision-theoretic contingency planning are in principle incapable of solving some planning problems that human beings find easy. This is fairly simple to show for state-space planners, and I will take this to indicate that decision-theoretic generalizations of classical POCL planning should be taken seriously. However, I will then go on to show that existing algorithms for decision-theoretic POCL contingency planning are unable to solve some other classes of problems. This will lead to a formal criterion of adequacy regarding how contingencies are used in decision-theoretic POCL planning, and the conclusion that existing planners are not theoretically adequate. I will close with some tentative remarks about how to construct an algorithm for decision-theoretic POCL contingency planning that satisfies the criterion of adequacy.

2. State-Space Planners

Classical POCL planning was once the only game in town, but its image has been tarnished by recent work on high-performance classical planners and the success of MDP (Markov decision process) planning in decision-theoretic contexts. Each of these planning techniques has its place, but I will argue in this section and the next that decision-theoretic generalizations of POCL planning may still provide the best hope for solving planning problems in many complex domains. My conclusions are tentative, however, because for the reasons given later in the paper, there do not currently exist any theoretically adequate POCL planners for decision-theoretic planning.

2.1 MDP Planning

Most work on decision-theoretic planning has proceeded within the framework of MDP planning (see Boutelier, Dean, and Hanks 1999 for an excellent survey). The best understood MDP planners are *fully observable* MDP planners (FOMDP's, introduced by Bellman 1957). FOMDP's are state-space planners. They consider the space of all possible world-states, and the probability that an action performed in any world-state will lead to a transition to any other world-state, and then they construct *optimal policies*, which prescribe the optimal action to perform in each world state.

The "standard" approach to solving FOMDP's is based upon Bellman's *principle of optimality* and linear programming (Bellman 1957), and in effect constructs an optimal policy by regressing backwards from the end of the plan. This approach chooses an optimal "*k*-steps to go" action for each world-state relative to a previously selected (*k*–1)-step optimal policy. The latter prescribes optimal (*k*–1)-steps-to-go actions, optimal (*k*–2)-steps-to-go actions, etc. The optimal 1-step-to-go actions (at a world-state) are simply those having maximal expected-values in that state. Relative to a choice of optimal 1-step-to-go actions, the optimal 2-steps-to-go actions (at a world-state) are those producing maximal expected-values when combined with the 1-step-to-go actions. And so on. An optimal *n*-step policy consists of the optimal *n*-steps-to-go actions, (*n*–1)-steps-to-go actions, ... , 1-step-to-go actions.

In executing a policy, an agent can only respond to those parts of the world-state that are known to it. FOMDP's assume that the world is fully observable, so that the choice of action can be made contingent on all features of the world simultaneously. Obviously, in most contexts fully observability is an unrealistic assumption. This has led to the investigation of *partially observable* MDP's (POMDP's introduced by Aström 1965). Here it is assumed that the agent's knowledge can be represented as a probability distribution over the space of world-states. The probability distributions constitute *epistemic states*, actions lead to transitions with various probabilities from one epistemic state to another, and the task is to find an optimal policy in the space of epistemic states. The general idea is to apply solution techniques for FOMDP's to the space of epistemic states (Smallwood & Sondik 1973).

A generally recognized problem for MDP's and POMDP's is that the size of the state-space grows exponentially with the number of features of the world that are taken into account. To illustrate the difficulty, consider a simple planning problem that generalizes Kushmerick, Hanks and Weld's (1995) "slippery gripper" problem. We are presented with a table on which there are 300 numbered blocks, and a panel of correspondingly numbered buttons. Pushing a button activates a robot arm which attempts to pick up the corresponding block and remove it from the table. We get 100 dollars for each block that is removed. Pushing a button costs two dollars. The hitch is that some of the blocks are greasy. If a block is not greasy, pushing the button will result in its being removed from the table with probability 1.0, but if it is greasy the probability is only 0.1. The probability of any given block being greasy is 0.5. We are given 300 chances to either push a button or do nothing. In between, we are given the opportunity to look at the table, which costs one dollar. Looking will reveal what blocks are still on the table, but will not reveal directly whether a block is greasy. What should we do? Humans find this problem terribly easy. Everyone I have tried this upon has immediately produced the optimal plan: push each button once, and don't bother to look at the table. Note that the order makes no difference, so if plans are identified with linear sequences of actions then there are 300! optimal plans. This is approximately 10^{614}.

We can cast this as a 599 step POMDP. Odd-numbered steps consist of either pushing a button or performing the null action, and even-numbered-steps consist of either looking at the table or performing the null action. World-states are determined by which blocks are on the table (T_i) and which blocks are greasy (G_i). The actions available are *nil* (the null action), P_i (push button i), and L (look at the table). This cannot be cast as a FOMDP, because the agent cannot observe which blocks are greasy. But notice that even if it could, we would immediately encounter an overwhelming computational difficulty. The number of world states would be 2^{600}, which is approximately 10^{180}. To get an idea of what an immense number this is, note that it has been estimated that there are approximately 10^{78} elementary particles in the entire universe. There is no way that an agent could even *represent* an optimal policy in this state-space, much less find one.

The state-space gets larger when we move to POMDP's. In general, POMDP's will have infinite spaces of epistemic states corresponding to all possible probability distributions over the underlying state-space, however a reachability analysis can often produce a smaller state-space with just finitely many possible probability distributions. In the slippery blocks example, it can be shown that in reachable epistemic states prob(T_i) can taken any value in the set $\{1.0, .5\cdot.9, .5\cdot.9^2, \ldots, .5\cdot.9^{300}, 0\}$ and prob(G_i) can take the values .5 and 1.0. Not all combinations of these values are possible, but the number of reachable epistemic states is greater than 301^{300}, which is approximately 10^{744}. So it is computationally impossible for an implemented system to even represent optimal policies in this POMDP, much less find them by linear programming.

This kind of difficulty has led to work on *factored* MDP's (see Boutelier, Dean, and Hanks 1999 for a general discussion of factored MDP's), and techniques for solving them. *Abstraction techniques* for solving factored MDP's observe that there are often differences between states that are not relevant to solving the problem.[1] It may be possible to cluster states together so that the computation of optimal *k*-steps-to-go actions is the same for all the states in any given cluster. In fact, in computing optimal *k*-steps-to-go actions for the slippery blocks problem, the only relevant difference is whether, for the different choices of i, prob(T_i) = 1. P_i is an optimal *k*-steps-to-go action in an epistemic state that will actually be reached by an optimal policy, relative to a terminal sequence of k–1 optimal actions, iff prob(T_i) = 1 and P_i is not a member of the terminal sequence. It is unclear whether algorithms for factoring MDP's will be able to figure this out, but

[1] This has been applied to FOMDP's by Boutlier 1997, Boutlier & Dearden 1996, Boutlier, Dearden & Goldszmidt 1995, Dietterich & Fann 1995, and Hoey , St-Aubin, & Boutilier 1999. Boutilier & Poole (1996) and Hansen & Feng (2000) apply this to the solution of POMDP's.

suppose we have an algorithm that can. Unfortunately, this means that there is still one cluster for each subset of the buttons, and so there are 2^{300} clusters. Furthermore, linear programming algorithms will have to regress backwards through all the 300! orderings of button pushings in order to compute that any of them is optimal. It is computationally impossible for an implemented system to do this.

There is a different technique for solving large MDP's that seems initially like it might be helpful. This is *parallel decomposition* (Boutlier, Dean and Hanks 1999). When the sources of value in an MDP can be split into independent subsources, with the overall value of a state being the sum of the values contributed by the subsources, it is sometimes possible to divide the MDP into MDP's for each subsource, find optimal policies for the parallel MDP's, and then merge them to obtain an optimal policy for the original MDP. The sources of value in the slippery blocks problem can be separated into the discrete values obtained by removing each individual block from the table, so it may seem that parallel decomposition might work. However, if we consider the problem of getting a single block off the table, the optimal policy is to push the corresponding button, look to see if the block is still there, and if it is then repeat the process until we either run out of turns or the block is removed. The optimal policy for the original problem involves no looks and no repeats, so there is no obvious way to construct it out of the optimal policies for these smaller problems.

My conclusion is that the slippery blocks problem is unsolvable using existing strategies for solving POMDP's. It is to be emphasized that this is actually an easy (for humans) problem, and it only becomes hard by casting it as a POMDP. It must be concluded that, at least in this particular case, this is the wrong way to do decision-theoretic planning. I do not want to conclude that there is in principle no way of solving this problem by factoring or decomposing POMDP's. Boutilier, Dean, and Hanks (1999) make a case for the view that classical goal-regression and POCL planning can be viewed as limiting cases of MDP planning, and I am going to suggest below that decision-theoretic generalizations of POCL planning can solve this problem. So my claim will only be that we should take decision-theoretic POCL planning seriously, either as an alternative to POMDP planning or as a particularly important special case of it.

2.2 Decision-Theoretic Planning Based Upon High-Performance Classical Planners

The image of POCL planning in classical contexts was severely tarnished by the creation of GRAPHPLAN (Blum and Furst 1995, 1997) and SATPLAN (Kautz and Selman 1996, 1998), which often outperform POCL planners by several orders of magnitude. There is some recent work on creating decision-theoretic planners based upon GRAPHPLAN and SATPLAN, so let us see how they fare on the slippery blocks problem.

High performance satisfiability planners deriving from SATPLAN convert the problem into a problem of finding assignments to propositional variables that satisfy a complex propositional formula. Majercik and Littman (1998, 1999) have extended this to probabilistic planning with their planners C-MAXPLAN and ZANDER. These are not actually decision-theoretic planners, because they do not take account of values — just the probabilities of achieving goals, and they aim at constructing plans that maximize the probability of goal achievement. It seems likely that the same ideas can be extended to decision-theoretic planning, and I will think of these planners in that way. C-MAXPLAN encodes the planning problem into a propositional formula, and then computes the probability that a plan will achieve its goal by computing all satisfying assignments to the chance variables and summing them. However, there are as many satisfying assignments as there are reachable world-states, i.e., more than 301^{300}, so the problem cannot be solved in this way.

ZANDER uses a different propositional encoding and then computes all possible trees of variable assignments, choosing an optimal subtree. But again, there are as many satisfying assignments as reachable world-states, so this tree cannot actually be built and the problem cannot be solved in this way.

It is also worth noting that both C-MAXPLAN and ZANDER must compute and compare all "plausible candidates" for optimal plans in order to determine which are actually optimal. Sophisticated pruning algorithms may make it unnecessary to compare all possible plans, but even with maximally efficient pruning all of the 300! optimal plans would have to remain unpruned and be compared with each other to verify that none is surpassed by another. A real agent cannot compare 300! (i.e., 10^{614}) plans.

GRAPHPLAN is not really a state-space planner. It is more like a goal-regression planner supplemented with a reachability analysis. GRAPHPLAN gains its efficiency by first constructing the plan-graph (used by the reachability analysis) and then searching backwards through it by goal-regression. DT-GRAPHPLAN (Peterson and Cook 2000) applies the same idea to decision-theoretic planning. To do this it computes probabilities and utilities for each proposition in the plan-graph at each stage of its expansion, including multiple entries for the different possible probabilities of each proposition. The original version of DT-GRAPHPLAN assumed that propositions were independent, and so it could not handle the slippery blocks problem in which there is little independence. More recently (Peterson 2001), probabilistic dependence has been handled by maintaining a Bayesian net in parallel with the plan-graph that computes the non-independent probabilities. To see how this fares, let us modify the slippery blocks problem slightly by supposing that instead of knowing that the probability of a block being greasy is .5, what we know is that half the blocks are greasy, each having the same initial probability of being greasy. The point of the change is that the probability of a block being greasy then changes as other blocks are removed from the table. This changes does not affect what plans are optimal. It is not clear how to cast this problem as a Bayesian net. Bayesian nets have to be acyclic, but if we include nodes for the greasiness of the blocks, acyclicality fails. The probability of a block being on the table after the corresponding button is pushed is influenced by whether it is greasy, and the probability of its being greasy given that the button is pushed is influenced by whether it is still on the table. If we do not include nodes for the greasiness of the blocks, then the nodes just concern which blocks are on the table at each stage and which buttons have been pushed. However, as noted above, the probability of a block being greasy is influenced by what other blocks are on the table, and that in turn affects the probability that the block will still be on the table after its button is pushed. Thus the Bayesian net must encode as a primitive probability every probability of the form **PROB**($T_i / P_i \,\&T_1$ & ... & T_k) where block *i* is not among blocks 1 – *k*.. There are 2^{300} such probabilities, so this Bayesian net cannot actually be built or encoded in a real agent.

3. Decision-Theoretic POCL Contingency Planning

State-space planners attempt to formulate a global map of at least those parts of the world that are relevant to the planning problem, and then distill the plan out of that global map. Initially, this seems like a good idea and it makes the logic of the planning problem clear. But what we are finding is that even for planning problems that humans find terribly easy, the global map can be much too big to be either represented or manipulated by a cognizer subject to realistic limitations. There has to be another way to solve these problems. After all, humans produce exact optimal solutions almost instantaneously.

I will suggest that decision-theoretic POCL planning algorithms constitute a class of candidates at least worthy of further investigation. How might a POCL planner solve the slippery blocks problem? The sources of value in this problem consist of getting the individual blocks off the table. A POCL planner can begin by constructing a plan for getting each block off the table individually, e.g., push the corresponding button. It is probably important that humans do this reasoning just once, for an *unspecified* block, whereas POCL planners must do it individually for each of the 300 blocks. This is something that should be explored as a way of improving POCL planners, possibly dramatically, but I will leave it aside for now. POCL planners then merge these individual subplans into the overall plan of pushing each button once. Then they search for and try to repair flaws in the resulting plan. Classically, this is threat detection and threat resolution, but in decision-theoretic contexts the flaws can be of more general sorts. Basically, the planner makes the defeasible assumption that the expected-value of the overall plan is the sum of the expected-values of the subplans, and then it must search for destructive interactions between different parts of the plan that make that assumption false. If such interactions are detected, an attempt is made to refine the plan to avoid or minimize the interactions. In the slippery blocks problem, there are no destructive interactions, and a POCL planner should be able to determine that quickly.

However, just finding the plan is not sufficient to solve the planning problem. The planner must also verify that the plan is a solution to the problem. Different planners understand the planning problem in different ways. POMDP planners usually take their task to be that of finding an optimal plan. But some, like Mahinur (Onder & Pollack 1997), take their task to be that of finding a plan whose expected-value comes up to a threshold set by the user. If we understand the planning problem in the latter way, and we set the threshold to be the value that is in fact the expected-value of the optimal plan, then merely finding the optimal plan suffices to solve the problem. This way of understanding planning problems is based upon Herbert Simon's notion of *satisficing* (Simon 1955). However, it does not seem to me to be a satisfactory way of understanding planning problems. If we set the threshold too high (higher than the value of the optimal plan), no plan can be found, and if we set it too low the planner will be content with finding plans that are gratuitously inferior to the optimal plan. In effect, we must solve the problem of finding an optimal plan before we know how to set the threshold.

If the objective is to find an optimal plan, the POCL planner cannot stop with just finding the plan that is in fact optimal. It must verify that it is actually optimal. Notice that for a POCL planner there are not 300! optimal plans — there is just one, with the steps unordered with respect to each other. POCL planners only order the steps if there is some reason to do so, and in this case there is none. This gives them an immediate advantage over those state-space planners that insist on comparing all plausible linear plans. However, even though there is only one optimal plan, the details of verifying that it is optimal remain a bit murky. A POCL planner *might* be able to reason that changing any of the button-pushing steps in its plan would result in not pushing one of the buttons (and possibly pushing another one more than once), which would lower the expected-value of the plan; and looking instead of not looking would raise the costs without raising the expected payoff, and so would also lower the expected-value. In this way, a POCL planner *might* be able to solve the slippery blocks problem quickly and efficiently. But as remarked, the details are murky because for the reasons that will be explained below, there are no theoretically adequate decision-theoretic POCL contingency planners that we can actually apply to this problem. Existing planners like B-PRODIGY (Blythe and Veloso 1997) or Mahinur (Onder & Pollack 1997) might well return the desired plan when applied to the slippery blocks problem, but because they are not theoretically adequate decision-theoretic planners, this does not establish that the plan is optimal. C-Buridan (Draper, Hanks, and Weld 1994) may also come to mind, but it is not a decision-theoretic planner — only a probabilistic planner. Furthermore, as will be seen below it is subject to the same theoretical shortcomings as B-Prodigy and Mahinur, and so it is not theoretically adequate even as a probabilistic planner.

I will investigate in more detail below how decision-theoretic POCL planners should work, but even at this preliminary stage we can see an important contrast between them and the state-space planners discussed in section two. The latter planners begin by constructing a complete map of the relevant parts of the world, and then cut the solution to the planning problem out of it. Their reasoning is, in an important sense, global, and the difficulties they encounter arise from the fact that "global" is often just too big. By contrast, POCL planners only engage in local reasoning. They first construct a crude plan on the basis of simple local relationships between actions and goals. Then they search for other local relationships that give rise to destructive interference. Then they search for still further local relationships that will enable the planner to refine the plan to avoid the destructive interference. Finally, they search for additional local relationships that may make it possible to improve the plan, generating a plan with a higher expected-value. In sum, POCL planners are *refinement planners* that start with a crude plan and then make it better. All of the considerations underlying both the discovery of the crude plan and its subsequent refinement are local considerations, and the search for refinements is driven by the *particular plan* that is being refined. Notice that, by way of contrast, the use of reachability analyses and factoring in section two is driven by the *planning problem* rather than by attempts to refine a single candidate plan, and that is much less effective because it must take account of all possible solutions to the planning problem.

These considerations at least suggest that decision-theoretic POCL planning should be taken seriously. There is, however, a well-recognized problem for classical POCL planners. They cannot solve very big problems. Satisfiability-based planners seriously outperform them. But here there may be room for hope. In a classical planning environment, such planners are trying to solve difficult search problems by brute force. It is very difficult to see how the search of a *general-purpose* classical POCL planner could be made more efficient (except by building in ad hoc heuristics for specific domains). Viewed as a brute-force search problem, decision-theoretic planning is harder than classical planning. However, decision-theoretic planners also have much more knowledge at their disposal that may be useful in directing plan search in intelligent ways. This is knowledge about probabilities and utilities. This may, in the end, enable decision-theoretic POCL contingency planners to solve real-world problems. The performance of such a planner will be dramatically influenced by the way it uses probabilities and utilities to direct its search. This is nicely illustrated by some recent work by Onder and Pollack (1997, 1999). By comparison, classical planners plan while wearing blinders, and can do little to improve performance except for employing faster search algorithms. So I do not think we can rule out the possibility that sophisticated decision-theoretic POCL planners may eventually be able to solve problems in truly complex environments.

On problems sufficiently simple to be amenable to solution by the planners of section two, POCL planning may be orders of magnitude slower. But on problems of real-world complexity, of the sort that will be faced by intelligent autonomous agents operating in unpredictable environments of complexity analogous to that facing humans in the real world, POCL planning may still be the only game in town. Whether POCL planners can actually solve such problems remains to be seen, but at least they will not be subject to the same kinds of difficulties as those encountered by the planners in section two. So let us turn to a more careful consideration of decision-theoretic POCL planning. I will begin with a review of classical POCL contingency planning and some general remarks about decision-theoretic POCL planning.

4. Background: Classical Contingency Planning

Traditionally, most work on planning has been more concerned with constructing plans than with executing them, and the structural elements of classical plans (e.g., causal links) were introduced primarily to control the search for plans rather than to control plan execution. These structural features may, of course, have dual uses, but it will be helpful throughout this paper to keep in mind the distinction between controlling plan search and controlling plan execution. The appeal to contingencies in contingency planning is a mechanism for controlling execution. The agent is supposed to do different things depending upon what knowledge is available at the time of execution.

Following the lead of Warplan-C (Warren 1976) and CNLP (Peot and Smith 1992), most contingency planners (e.g., C-Buridan (Draper, Hanks & Weld 1994), Mahinur (Onder & Pollack 1997), B-PRODIGY (Blythe & Veloso 1997)) use "knowledge actions" or "observation actions" that can have different outcomes (e.g., knowledge that *P*, or knowledge that ~*P*), and produce plans with diverging branches, one branch for each outcome. We can think of the branches as subplans or subroutines that are called by the larger plan depending upon the outcome of the observation actions. We might diagram such plans as in figure 1. Following Pryor and Collins (1996), the thin arrows represent causal connections, and the thick arrows represent the flow of control in execution. In classical contingency planning, plans with such structure are produced by goal-regression. *Action*$_1$ has the precondition *P*, the planner is unable to construct a plan for making *P* true, so it takes *P* to be a contingency and then looks for a plan for what to do if *P* is false.

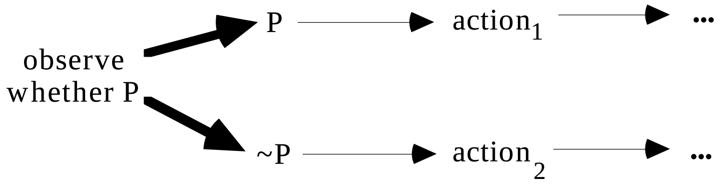

Figure 1

An important advance was made by Pryor and Collins (1996) when they observed that knowledge acquisition can involve much more than single "observation actions". E.g., I might plan for where to eat lunch tomorrow if the weather forecast predicts rain. To find out whether it predicts rain, I may have to execute another plan, e.g., buy a television, tune to the weather channel, and watch the forecast. The steps of that plan might interfere with steps of the lunch plan. For instance, if I buy the television, I may not have enough money left to buy lunch. In that case, a better plan might be to buy a newspaper and read the forecast there. This suggests that contingency planning should involve a knowledge subgoal of the form *I know whether P*, and the plan for achieving that should be integrated into the larger plan. This has the result of inserting an additional node into the above plan, as in figure 2. Here, for purposes of plan search, *know whether P* acts as a subgoal, achieved by an arbitrarily complex plan for acquiring knowledge. The dashed arrows indicate that the knowledge subgoal is ordered temporally after the state of which it is supposed to provide knowledge. That is, in executing this plan the agent does not try to find out whether *P* is true until *P* is expected to be true.

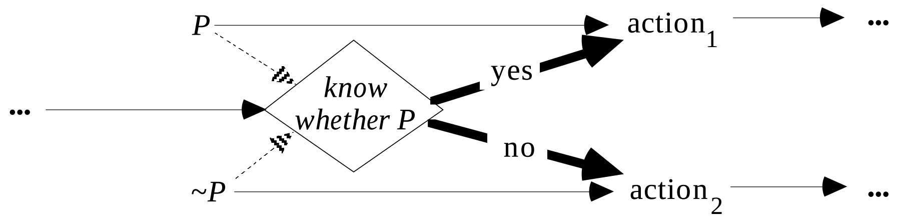

Figure 2

Pryor and Collins' conditional planner Cassandra inserts an additional "decision node" between the knowledge subgoal and the subgoals it provides knowledge of. This makes the plan look a bit more like a classical non-conditional plan, with subgoals connected only via actions. However, that appearance is really illusory, because the decision node doesn't *cause* the subgoal to be true any more than the knowledge subgoal does, so I will not include decision nodes in contingency plans. Such nodes are not required for plan execution, and plans produced by Cassandra can always be stripped of decision nodes by postprocessing.

I have drawn a diamond around the knowledge subgoal, because this node of the plan acts as more than a subgoal. It plays the standard role of subgoals in plan search — initiating further search for subplans that will achieve the subgoals. But it also plays a crucial role in plan execution. Its function there is to turn execution on and off, depending upon the outcome of the knowledge acquisition. I will refer to nodes playing this dual role as *contingency nodes*.

Most authors use the terms "contingency planning" and "conditional planning" interchangeably, with some preference for the latter term. However, for reasons that will become more apparent when we turn to decision-theoretic contingency planning, I will use "contingency planning" as the generic term and "conditional planning" to refer to a specific variety of contingency planning. I will confine the term "conditional plan" to plans in which we plan to do one thing if *P* is true and something else if *P* is false. In conditional planning, the contingency node forms the origin of a *conditional fork*. For reasons that will emerge shortly, classical contingency planning has focussed exclusively on the production of conditional plans.

Classical contingency planning can be generalized in two directions. First, as has often been recognized, we can have contingency plans (conditional plans) for more extensive lists of alternatives. E.g., we may plan for what do if we draw a red ball out of the urn, what to do if we draw a yellow ball, and what to do if we draw a blue ball, producing a plan with a three-pronged conditional fork. As has also often been observed (e.g., in Pryor and Collins 1996), this can be reduced to the case of planning for binary alternatives by embedding further contingencies within the contingency plan, as in figure 3.

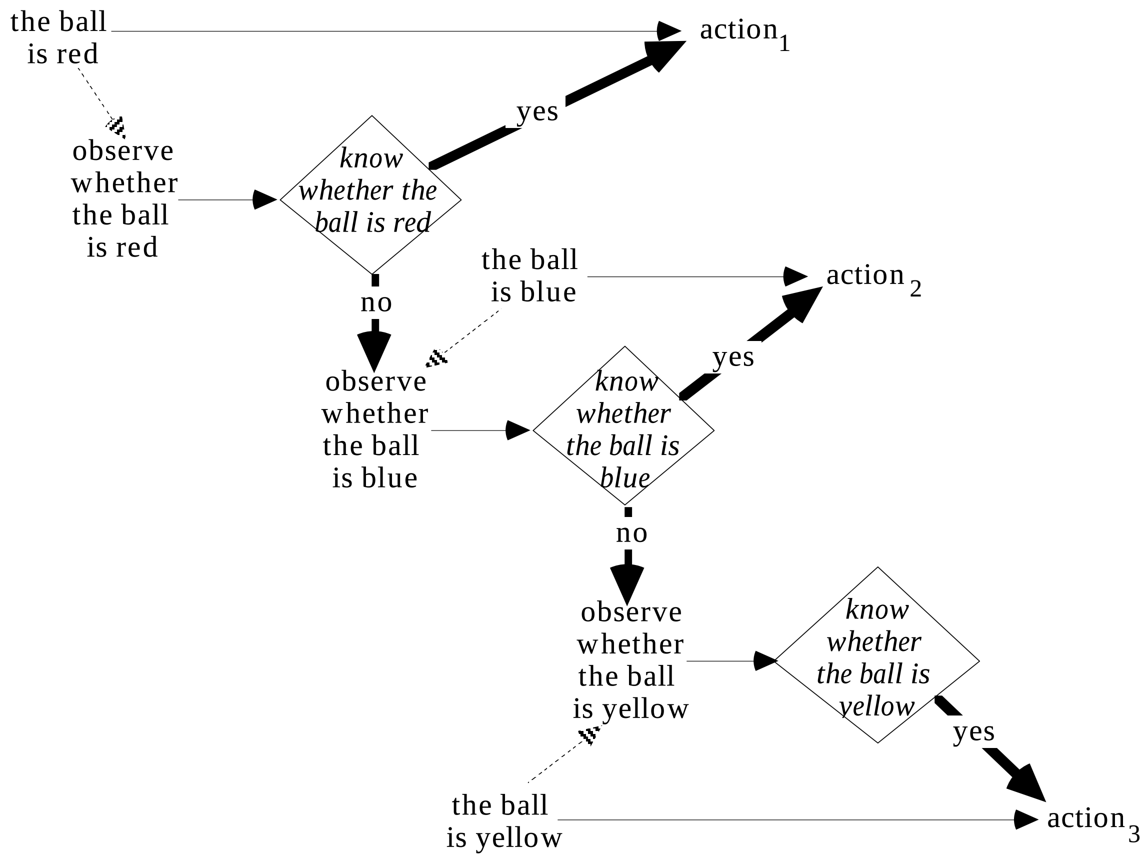
the ball is red
action$_1$
yes
observe whether the ball is red
know whether the ball is red
no
the ball is blue
action$_2$
yes
observe whether the ball is blue
know whether the ball is blue
no
observe whether the ball is yellow
know whether the ball is yellow
yes
the ball is yellow
action$_3$

Figure 3

An observation that will prove more important when we turn to decision-theoretic contingency planning is that sometimes we only plan to take actions contingent on one possible outcome of the knowledge subgoal. E.g., I may plan to open my umbrella if I observe that it is raining, but I make no special plans for what to do if I observe that it is not raining. (Pryor and Collins discuss a similar case in connection with Cassandra.) This can happen in classical contingency planning if our plan for what to do if the contingency fails is a subplan of the plan for what to do if the contingency holds. In that case, it will be executed regardless of the state of the contingency, and so can be represented as non-contingent. In that case we can regard the knowledge subgoal as having the form *if P is true, know P*, and diagram the plan as in figure 4. Here “yes” means “I believe that *P* is true”. In classical planning, where actions have determinate effects, the agent will only believe *P* on the basis of the knowledge acquisition plan if *P* is true, so “yes” could just as well stand for “I know that *P* is true”. However, in decision-theoretic planning, this difference will be important. In computing expected-values, we will want to compute them conditional on the agent believing *P* (perhaps incorrectly) on the strength of the knowledge acquisition plan, not conditional on the agent correctly knowing *P*.

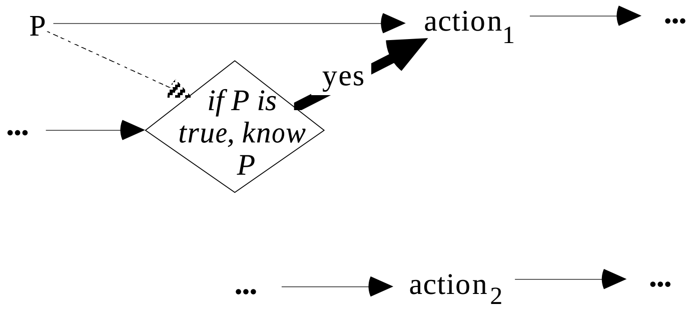

Figure 4

In classical planning, plans must guarantee the achievement of their goals. This requirement is retained in classical contingency planning. Knowledge acquisition plans are plans for determining that *P* is true if it is true. Thus a correct knowledge acquisition plan must be guaranteed to reveal that *P* is true if it is true. If a knowledge acquisition plan does not reveal that *P* is true, we can infer that *P* is false. So a single knowledge acquisition plan will always suffice to reveal *whether P* is true, and hence contingency plans can be written in as in figure 2. However, in more realistic (decision-theoretic) planning, we cannot expect knowledge acquisition plans to be infallible. For example, we might try to discover a telephone number by calling “information”, but that won’t produce an answer if the number is unlisted, and it will produce the wrong answer if the number has been changed. The fallibility of knowledge acquisition plans implies that we may have unrelated plans for knowing *P* if *P* is true and for knowing ~*P* if ~*P* is true. As such, the conditional forks in such plans aren’t really fork-shaped. They are more like sets of two or more parallel subplans, as in figure 5.

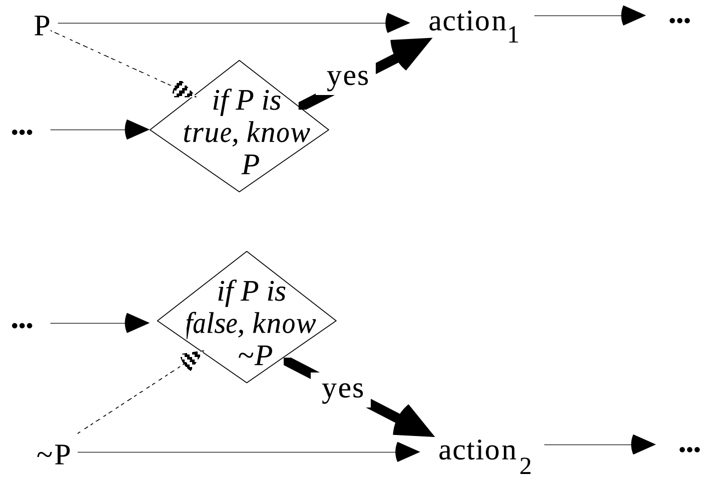

Figure 5

5. Decision-Theoretic POCL Plans

Decision-theoretic POCL planners construct "local" plans aiming at localized goals and having structures analogous to those produced by classical POCL planners. The plans produced by such planners are evaluated in terms of their expected-values. The expected-value of a plan is the sum of the values of its goals discounted by the probability that it will achieve them less the execution costs of its steps discounted by the probability that they will be incurred. (A plan can also have value-laden side effects, but I will assimilate these to execution costs having either positive or negative values.) I will refer to these two constituents of the expected-value as the *expected-payoff* and the *expected-execution-costs*. The exact definition of these concepts is more complicated than generally realized (Pollock 2001), but the details will not be relevant to the present discussion. I assume that the objective of a decision-theoretic planner is at least to produce plans with positive expected-values, and higher expected-values when possible. If there are optimal plans and it is computationally possible to find them, then we can take the objective to be that of finding optimal plans.

Existing POCL contingency planners are refinement planners. They construct an initial plan, typically by goal-regression, and then refine the plan by detecting and correcting "flaws". Most of the existing POCL contingency planners are based on SNLP (McAllester and Rosenblitt 1991). (An exception is B-PRODIGY (Blythe and Veloso 1997), which is based upon PRODIGY 4.0.) Following SNLP, classical contingency planners recognize two kinds of flaws — open conditions and threats. The catalog of tools for repairing flaws includes those used for non-contingency planning (adding a causal link to close an open condition, adding ordering constraints to resolve threats, confrontation to resolve threats involving "conditional effects", etc.), but also contains one new tool. If a way cannot be found of closing an open condition, the planner can instead take the plan with that open condition to be a contingency plan to be executed when that contingency is known to be true, add the knowledge subgoal as a new open condition (initiating search for a plan for acquiring the knowledge), and initiate a search for a subplan for what to do if the contingency fails. Onder, Pollack, and Horty (1998) refer to this as *corrective repair.*

I assume that decision-theoretic contingency planners will likewise by refinement planners. The planners will construct an initial plan based on probabilistic connections, and then refine the plan by detecting and correcting flaws. The flaws that arise in decision-theoretic plans are similar to those that arise in classical plans, but their status is a bit different. Decision-theoretic plans will still have causal-links indicating that one plan step is intended to establish a precondition for another plan step, but the first step may only establish the second with some probability rather than with certainty. Threats arise when other steps lower the probability that the precondition of the second step will be true when it is needed. The list of threats that can arise in decision-theoretic planning is somewhat more extensive than the list of classical threats. I catalogued various kinds of “decision-theoretic undermining” in (Pollock 2000). Technically, such threats are flaws, however, in decision-theoretic planning flaws need not be fatal. Although a threat may lower the probability of the plan working, it may not lower it very much, and the flawed plan may still be a good plan with a high expected-value. In decision-theoretic planning, flaws might better be called “imperfections”.

Note that this is also true of open conditions. If a plan has an open condition that cannot be closed by adding a causal link, we might still know that the open condition will probably come true of its own accord, in which case the plan may have a high expected-value even with the open condition. Technically, in the spirit of SNLP, such conditions may be closed by linking them to a dummy *start* action, but notice that they need not be conditions *true at the start*. Instead, they may become true at the appropriate time, and without the intervention of the agent. For example, if a friend has told me that he will be at a certain place at a certain time, I may plan to meet him there. I cannot be sure that he will be there, and I can do nothing to make it the case that he will be there (and he is not there at the start time), but a plan that assumes he will be there may still be a good plan. I will call such open conditions *dangling antecedents*.

I assume then that a decision-theoretic POCL contingency planner is built upon a decision-theoretic refinement planner. Contingencies are handled by adding refinement tools, just as in classical contingency planning. However, what has generally been overlooked is that the refinement tools that have been used for adding contingency nodes to classical plans are only a subset of the refinement tools of use in decision-theoretic contingency planning (and probably should not be included among the primitive refinement operators). This is because, as I will show in section seven, the addition of contingency nodes can improve decision-theoretic plans in more ways than they can improve classical plans.

It will be convenient to introduce some technical terminology. In classical planning, causal links have the form $\langle step_1, subgoal, step_2\rangle$, where $step_1$ is intended to achieve *subgoal*, which is a condition or part of a condition under which $step_2$ can achieve its purpose. In decision-theoretic planning it is convenient to give causal links a slightly more complex structure of the form $\langle antecedent, step_1, subgoal, step_2\rangle$. Here *antecedent* is a list of formulas comprising the condition under which $step_1$ is expected to achieve *subgoal*. The antecedent of the link is *antecedent*, and the link-probability is prob(*subgoal* / *antecedent* & *action*) where *action* is the action prescribed by $step_1$. $step_1$ is the root step of the causal link, $step_2$ is the consuming step of the causal link, and *subgoal* is the goal of the causal-link.

6. Contingency Completeness

Having argued that POCL contingency planning represents a promising avenue of approach for decision-theoretic planning, I will now raise a general theoretical difficulty for existing algorithms for decision-theoretic POCL contingency planning. I will (1) argue that a certain criterion of adequacy should be imposed on decision-theoretic POCL contingency planners, (2) argue that existing planning algorithms do not satisfy this criterion of adequacy, and (3) make some very tentative remarks about how a planner that satisfies the criterion of adequacy might be built.

Criteria of adequacy are familiar in planning. Classical planners are evaluated (in part) in terms soundness and completeness — are the plans produced by the planner guaranteed (given the assumptions of the planning problem) to achieve their goal, and if there exists a plan that is guaranteed to achieve the goal, can the planner find such a plan? These are generally treated as criteria of adequacy. A planner that is unsound or incomplete is considered a bad planner (although considerations of efficiency might override these constraints in some applications).

When we turn to decision-theoretic planning, plans are no longer supposed to be guaranteed to achieve their goals, so we must replace soundness and completeness by adequacy conditions more suited to decision-theoretic planning. The usual assumption in decision-theoretic planning is that our objective is to produce optimal plans — plans with maximal expected-values. Given that assumption, soundness and completeness can be redefined in terms of the ability of the planner to find optimal plans. However, when defined in terms of optimality, soundness and completeness may not always be reasonable desiderata, for the reasons given in my (2001). In sufficiently complex environments there is no guarantee that there will exist optimal plans, and if they do exist it may be computationally infeasible to require planners to find them. In that case, the objective of the planner should presumably be to find the best plans possible within the constraints (e.g., available time) of the planning problem. Soundness and completeness can then be redefined accordingly. If one is enamored of the satisficing approach to decision-theoretic planning, one could instead define a planner to be sound if any plan it produces is guaranteed to have an expected-value meeting the specified threshold, and complete if it is guaranteed to find such a plan if there is one.

The problem I will address here arises specifically when we add contingencies to decision-theoretic plans. When should contingencies be inserted into plans? In decision-theoretic planning, a contingency should be inserted when doing so produces a plan with a higher expected-value. My proposal is that an algorithm for decision-theoretic contingency planning should be required to be such that, given sufficient time for solving the problem, whenever it produces a plan, if the plan can be improved by inserting a contingency then the algorithm is in principle capable of finding and inserting the contingency. I will call this *contingency-completeness*. If it is assumed that soundness and completeness (defined in terms of either optimality or finding the best plan possible within the constraints of the planning problem) are criteria of adequacy for decision-theoretic planners, then contingency-completeness would seem to be a derived criterion of adequacy. It is hard to see how a decision-theoretic planner could either find optimal plans or verify that the plans found are optimal if it does not exhibit contingency-completeness. So the planner will not be complete if it is not contingency complete. Furthermore, it will most likely not even be sound, because if it is unable to find the plan that is in fact optimal, it will have no way of determining that the best plan it does find is not optimal, and hence it will return that plan as optimal when it is not. Even if the objective is just to find the best plans possible given the time constraints of the planning problem, it seems reasonable to require a planning algorithm to exhibit contingency-completeness. Otherwise, certain avenues for improving its plans will not be open to it. Should we lower our aim to the point of requiring only that our planner find plans whose expected-values come up to a specified threshold, then planners that are not contingency-complete can be sound, but they will presumably still fail to be complete. This is because every plan surpassing the threshold may be one the planner cannot find because it is unable to insert required contingencies into the plans it does find.

7. Contingencies and Plan Refinement

I turn now to a catalog of ways decision-theoretic plans can be improved by adding contingency nodes. Given any refinement planner for decision-theoretic planning, this catalog might be used to generate an algorithm for decision-theoretic contingency planning. Start with the decision-theoretic refinement planner, and then add refinement tools that seek to improve a plan by adding contingency nodes in any of the ways described in the catalog. I will return to this suggestion later, although the point of this paper is not to build a decision-theoretic POCL planner, but to investigate theoretically what such a planner should be capable of doing.

Let us turn then to the reasons there can be for incorporating contingency nodes into decision-theoretic plans. In general, such a reason will be a consideration that increases the expected-value of the plan.

7.1 Dangling Contingencies

It was remarked above that one way in which decision-theoretic planning differs from classical planning is that goal-regression can terminate on *dangling antecedents*, which are subgoals the agent does not *know* to be true, but the agent does know to be probable. If they are sufficiently probable, the plan can have a positive expected-value even though they are not known with certainty. If the goals of the plan are sufficiently valuable, the plan might have a positive expected-value even when the dangling antecedents are quite improbable. For instance, if I am presented with a $100,000,000 lottery with one million tickets, and am offered a ticket for one dollar, I should avail myself of the opportunity. In doing so, I am adopting the plan of figure 6. This plan has only a one in a million chance of achieving its goal, because the dangling antecedent *ticket selected* has a probability of only .0000001 of being true, but it is still an excellent plan. Its expected-payoff is $100, and the expected-execution-cost is $1, for an expected-value of $99.

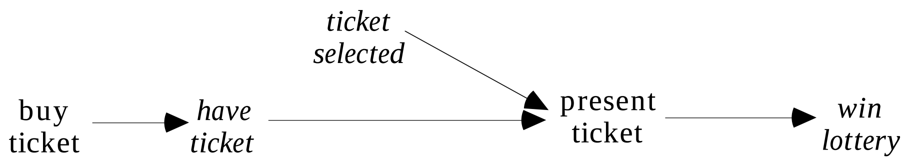

Figure 6

Although a decision-theoretic plan can have dangling antecedents, it will often be possible to improve such a plan by adding a check at the time of execution to see whether the antecedent is true. What this accomplishes is the avoidance of unnecessary execution costs in cases in which falsity of the dangling antecedent makes the goal unattainable, and if the check is not itself too costly, this will increase the expected-value of the plan. I will refer to this use of contingencies as *dangling contingencies*. For example, suppose I order something from a store and am told that it is on order and will probably arrive on Wednesday. I could simply plan to drive to the store on Wednesday and pick it up, implementing the plan of figure 7. But as in figure 8, I can construct a better plan by calling the store first to verify that the order has arrived.

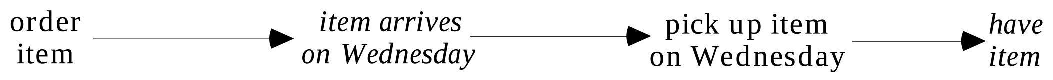

Figure 7

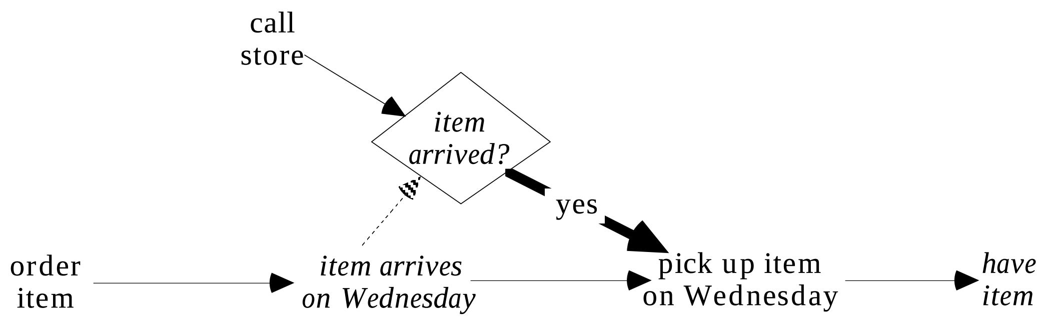

Figure 8

Contingency nodes function by eliminating some of the circumstances under which a plan or part of a plan may be executed. As such, they can reduce expected-execution-costs by preventing them from being incurred when the contingency fails. They may also reduce the expected-payoff if there is some possibility of the plan achieving its goal even when the agent believes that the contingency fails. For example, the store clerk might tell me in error that my order has not arrived. Incorporation of a contingency node will also add execution costs ("knowledge costs") involved in acquiring knowledge of the contingency. If the saving in expected-execution-costs is greater than the sum of the knowledge costs and the decrement in the expected-payoff, then it is worthwhile to incorporate the contingency into the plan.

This use of contingencies is reminiscent of what Onder, Pollack, and Horty call *corrective repair*, but there is an important difference. In corrective repair (following Peot and Smith 1992, and Warren 1976), as in the above example, we select a plan with an open condition (or dangling antecedent) and attach a contingency node. But corrective repair also requires that we add a subplan for what to do if the contingency fails. In classical planning, where we are trying to construct plans that are guaranteed to achieve their goals, we must do this. However, in decision-theoretic planning we are only trying to construct plans with high expected-values. Such a plan might simply acknowledge that if the contingency fails then the goal will not be achieved. So corrective repair requires too much. Accordingly, planners like Mahinur and B-Prodigy that rely upon corrective repair for the insertion of contingencies cannot produce the plan of figure 8. If that plan is optimal then they are unable to find the optimal plan. If completeness is defined in terms of optimality, then it follows that these planners are incomplete. Furthermore, if they cannot find the plan of figure 8, then any plan they do return will fail to be optimal, so they are not sound either. If soudness and completeness are instead defined in terms of finding plans surpassing thresholds, Mahinur and B-Prodigy and sound. That is, any plan they return will genuinely have an expected-value surpassing the specified threshold. However, they are still not complete. The only plans surpassing the threshold might be plans involving contingencies that cannot be inserted by corrective repair. Hence for optimality, these planners are neither sound nor complete, and for satisficing they are sound but not complete. Although their authors never claimed they were sound or complete, it is interesting to see that they fail on such a simple example.

7.2 Anchored Contingencies — Blind or Enlightened Execution

The standard use of contingencies in classical planning is to verify the truth of subgoals the agent cannot affect — dangling contingencies. However, in decision-theoretic plans, it may actually be more common to employ contingencies in connection with subgoals the plan tries to achieve. This gives rise to *anchored contingencies*. This is most easily seen by considering two ways in which plan steps can be executed.

The causal links of a plan tell us how the plan is supposed to work if everything goes according to plan. However, the relationship between the causal links and plan execution is not straightforward. In particular, successful execution of a plan does not require the agent to know whether a plan step successfully achieved its purpose. For example, suppose I want to impress a friend with my culinary skills, and I plan to accomplish this by having him over for cake. I don't want to go to the trouble of baking the cake until I have issued the invitation and know that he can come. If I am going to bake a cake when I get home, I will need sugar. I might plan to achieve that subgoal by asking my wife to pick up sugar at the store. Thus I produce the plan of figure 9. I may have no way of verifying that my wife actually got the sugar until I get home. Still, if I regard her as a reliable procurer of sugar, I may go ahead and issue the invitation. In decision-theoretic planning, this can be entirely reasonable. As long as it is probable that my wife will secure the sugar in response to my request, the plan can have a positive expected-value without my being able to verify that the sugar will actually be available.

On the other hand, suppose the person I want to impress is my boss, and it would be very bad to invite him over for cake and then be unable to bake it because I do not have sugar. In this case I might add a contingency node to my plan. I may decide to call my wife at home and verify that she got the sugar before I issue the invitation. This produces the plan of figure 10.

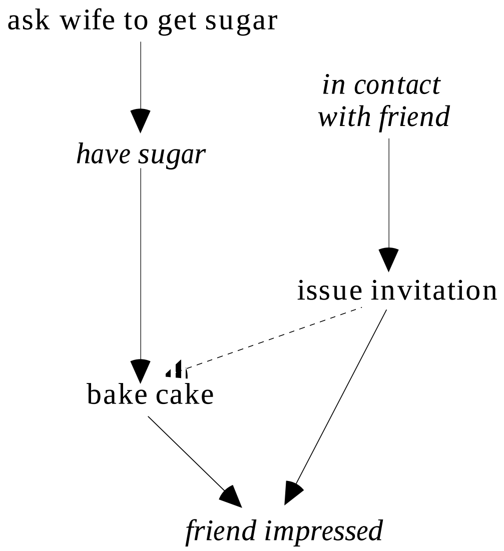

Figure 9

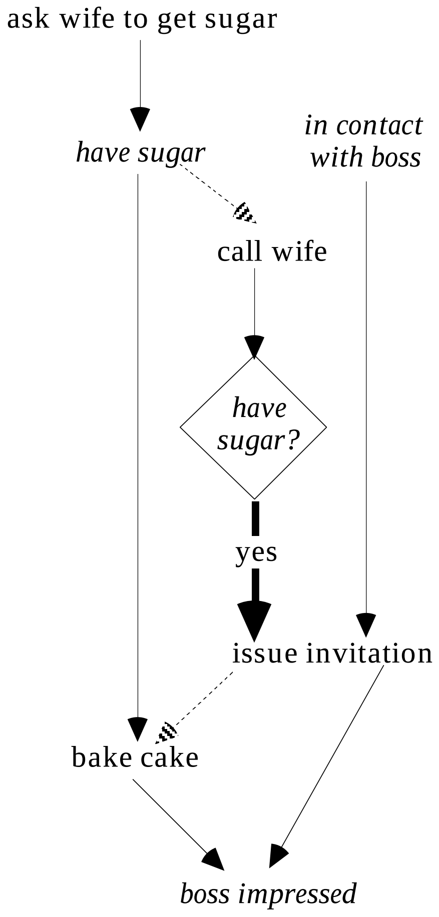

Figure 10

What this example illustrates is that we want a distinction between *enlightened execution*, in which we refrain from continuing plan execution until we verify that a subgoal has been achieved, and *blind execution*, in which we just assume that subgoals have been achieved and continue to execute the plan blindly. As illustrated, enlightened execution can be captured by making the next step contingent on knowing that the subgoal has been achieved. Henceforth, my understanding will be that in the absence of such a contingency node, a plan step is to be executed blindly. Notice that this kind of contingency requires an ordering constraint between the subgoal “have sugar” and the action “call wife”. Asking my wife to get sugar only results in having sugar sometime later, and I do not want to call her to verify the presence of sugar until after that time. (In fact, ordering constraints should always have this form. Classical planning assumes that plan steps achieve their objectives instantaneously, but in real life that is not the case. See my (1998) for a planning algorithm that accommodates this.)

It is worth noting that the contingency check could occur immediately after the plan purports to make it true, or just before it is used (to ensure that it has remained true), or anywhere in between. And for crucial subgoals in important plans, the contingency check might be repeated at each of those times. For example, if I catch a rattlesnake and put it in a cage in the back seat of my car for transport to another location, I will probably look over my shoulder a number of times to verify that it is still there.

The literature on contingency planning has usually made the assumption that we should only engage in contingency planning regarding contingencies we cannot affect — dangling contingencies. In classical planning, where actions are guaranteed to have their effects, that is unobjectionable. There is no reason to check a subgoal that *must* have been achieved. But as this example illustrates, in decision-theoretic planning, where effects are only related to actions probabilistically, it will often be desirable to check that subgoals have been achieved. This gives rise to anchored contingencies.

7.3 Contingencies Aimed at Strengthening the Antecedents of Causal Links

The introduction of a contingency node into a plan has the effect of restricting the cases in which some parts of the plan are executed. As I indicated above, this will typically lower both the expected-payoff and the expected-execution-costs. The plan is improved if the expected-execution-costs are lowered more than the expected-payoff. There are in principle two ways this can be accomplished. The contingency node might restrict execution to cases in which the payoff is higher than usual but the execution costs are either average or not enough higher to offset the increase in the expected-payoff. I will call this a *payoff contingency*. Alternatively the contingency node might restrict execution to cases in which the execution costs are lower than usual but the payoff is either average or not enough lower to offset the decrease in execution costs. I will call this a *cost contingency*. The difference can be usefully diagrammed as in figure 11, where the bottom axis represents the possible circumstances under which the plan might be executed and the grey area represents the restriction of execution to the worlds in which the contingency holds. Both dangling contingencies and anchored contingencies are payoff contingencies. That is, they select for circumstances in which the payoff is higher than average.

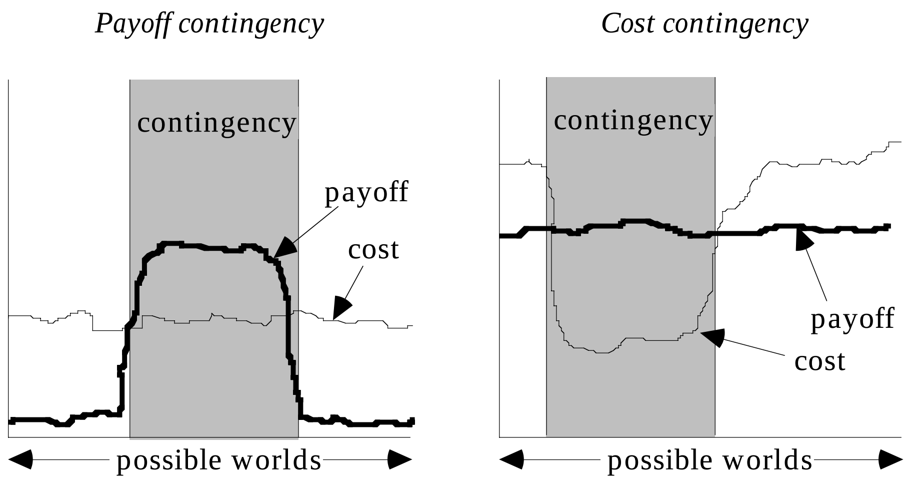

Figure 11. Two kinds of contingencies

Usually, properly-constructed plans will have higher payoffs if all the causal links work, i.e., they all succeed in achieving their purposes. We can increase the probability of a causal link working in either of two ways — by increasing the link probability or by increasing the probability of its antecedents. Both dangling contingencies and anchored contingencies work in the latter way. The information produced by the knowledge subplan makes it more probable that the contingency is true, where the contingency is one of the antecedents of the causal-link. In both dangling contingencies and anchored contingencies, the contingency is that an antecedent of a link was true when it was supposed to be. Accordingly, the contingency node must go between the antecedent being tested and the root step of the causal link. In general, the further back in the plan the contingency node can be placed, the greater the savings to the execution costs. Thus in figure 10, we place the contingency node before *issue invitation*, not just before *bake cake*.

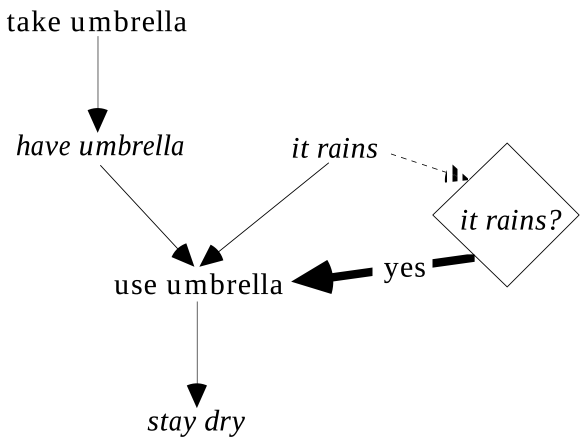

Figure 12

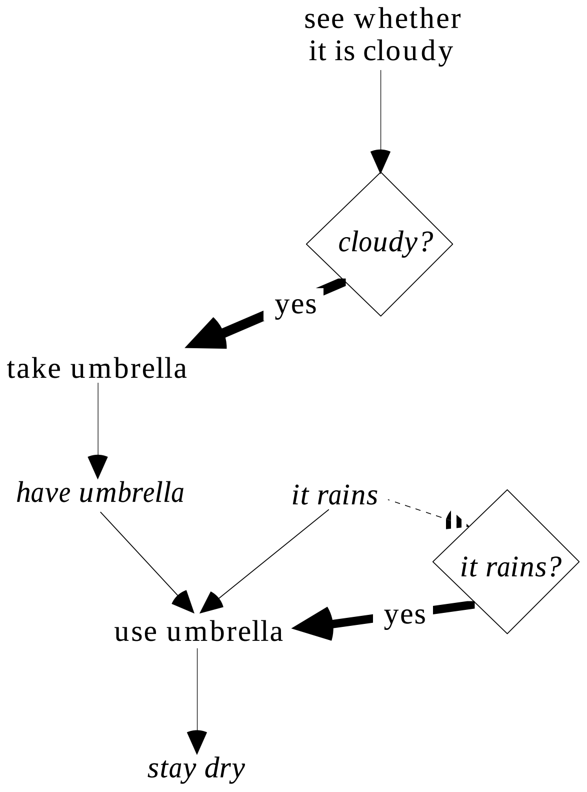

Figure 13

The selection of contingency nodes that strengthen causal links by making their antecedents more probable can be generalized. Instead of choosing the antecedents themselves as the contingencies, we can select contingencies that can be assessed earlier in the plan and make the antecedents more probable. For instance, I may plan to take an umbrella in case of rain, but only if it is cloudy. The plan to carry an umbrella regardless is that of figure 12. The plan to carry it only when it is cloudy is that of figure 13. The contingency *cloudy?* of figure 13 makes it more probable that it will rain, and by placing it before the step *take umbrella* we make it more likely that the umbrella will only be carried when it rains. This is a payoff contingency, selecting for cases in which it is cloudy and hence more apt to rain (which is required for a payoff), and eliminates the execution costs of carrying the umbrella in cases in which it is not cloudy. It may occasionally rain even when it is not initially cloudy, so the contingency may also lower the expected-payoff (the average payoff over all worlds), but by a smaller amount. The difference between the contingency *cloudy?* and the contingency *it rains?* is that the former can be placed earlier in the plan and so eliminates more execution costs. The general refinement operator that produces the plan of figure 13 will look for a contingency that increases the probability of the antecedent of a link being true, and will then place the contingency as early in the plan as possible but subject to the constraint that it not prevent the execution of causal links that contribute to the top level goals independently of the causal link that is being strengthened. Deciding whether the insertion of the contingency actually improves the plan will require computing expected-values, and we may have to explore different locations for the contingency in the plan.

7.4 Contingencies Aimed at Increasing Link Probabilities

The other way in which the insertion of contingency nodes can select for cases in which causal links are more apt to achieve their goals is by increasing the link probabilities. I will call these *strengthening contingencies*. For example, if my wife is now (at planning time) angry with me, I may fear that she will still be angry tomorrow, and the probability of her buying sugar for me will be lower if she is angry. So I may decide to execute my plan for impressing my friend only if I first ascertain that my wife is over her anger. This produces the plan of figure 14. This is an improvement on the plan of figure 9 because the probability of my having sugar given that I ask my wife to get it is higher if she is not angry.

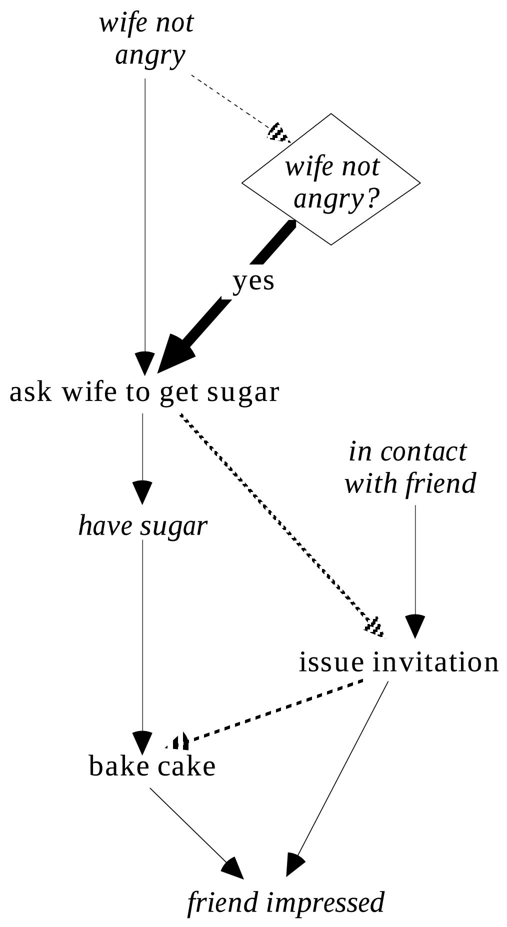

Figure 14

In figure 14, the introduction of the contingency node restricts the execution of the plan to cases in which the link probability is higher than it is in the plan of figure 9. This may not raise the absolute probability of the top-level goal being achieved, because my wife might buy the sugar even if she is still angry. Thus it may actually lower the expected-payoff of the plan because the plan is executed in narrower circumstances. However, the plan also has expected-execution-costs resulting from the possibility of my insulting my friend if I am unable to bake the cake. By restricting when the plan is executed the contingency lowers the expected-execution-costs faster than the expected-payoff, and so raises the expected-value of the plan. As such, this is another payoff contingency.

Dangling contingencies can be viewed as a limiting case of strengthening contingencies. In figure 8, *item arrived* is a strengthening contingency, strengthening the final causal link. Conversely, notice that *wife not angry* in figure 14 is a dangling contingency. In general, if a plan has a dangling antecedent, that antecedent presumably raises the probability that the action to which it is attached will achieve its purpose, and so the addition of a dangling contingency node beside the dangling antecedent produces a strengthening contingency. There is, however, an important difference between the ways the contingencies function in figure 8 and figure 14. In figure 8, if the item doesn't arrive on Wednesday we have no reason to expect that we can acquire it by picking it up on Wednesday. The antecedent *item arrives on Wednesday* is essential to having any causal link at all. On the other hand, in figure 14, there is a generally high probability that I can acquire sugar by asking my wife to get it, and the antecedent *wife not angry* serves only to raise that probability above what it would otherwise be. This suggests that despite the structural similarity in the plans of figures 8 and 13, they are produced by two different refinement operators. The refinement operator in figure 8 works on an existing causal link and selects one of its antecedents to be a contingency. The operator at work in figure 14 selects a *new* antecedent that will raise the link-probability, and builds a contingency out of that.

7.5 Contingencies Aimed at Lowering Execution Costs by Eliminating Side Effects

Thus far I have considered payoff contingencies — contingencies that raise plan expected-values by restricting execution to cases in which the payoff is higher than usual but the execution costs are either average or not enough higher to offset the increase in the expected-payoff. Contingencies can also raise plan expected-values by restricting execution to cases in which deleterious side effects do not occur, so that the execution costs are lower than usual but the payoff is either average or not enough lower to offset the decrease in execution costs. These are cost contingencies, as diagrammed in figure 11. For example, in evaluating a course of medical treatment, it may be noted that it will have highly undesirable side effects (high execution costs) if the patient has a certain unusual condition that may not be obvious to the physician. By testing for that condition and prescribing execution of the plan only when it is known that the patient does not have that condition, it may be possible to lower the expected execution costs and so raise the expected-value of the plan. To construct this plan, we begin by constructing the plan of figure 15. We then note the possible execution costs diagrammed in figure 16 using thick grey arrows. We can avoid these execution costs by adding the contingency of figure 17. Note that the contingency consists of ruling out one of the conditions leading to the execution cost.

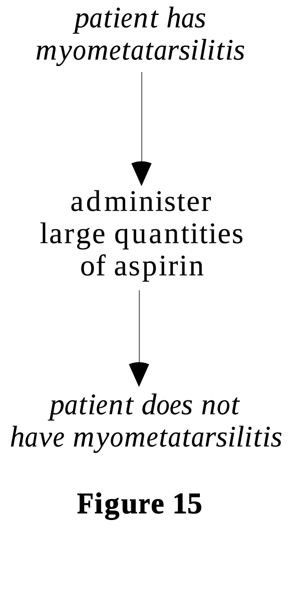
patient has myometatarsilitis
administer large quantities of aspirin
patient does not have myometatarsilitis

Figure 15

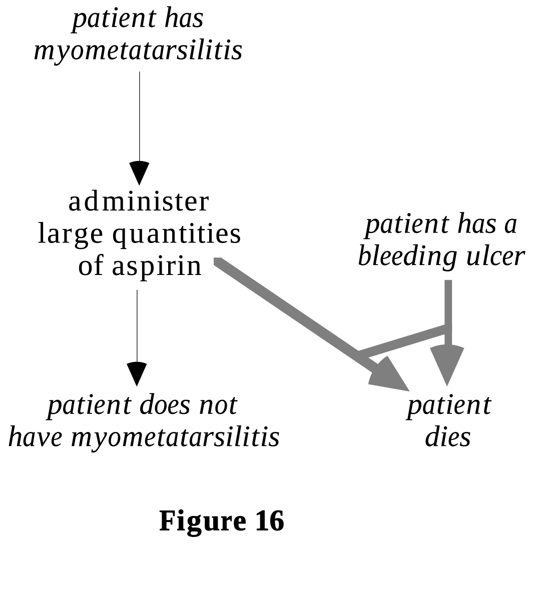
patient has myometatarsilitis
administer large quantities of aspirin
patient has a bleeding ulcer
patient does not have myometatarsilitis
patient dies

Figure 16

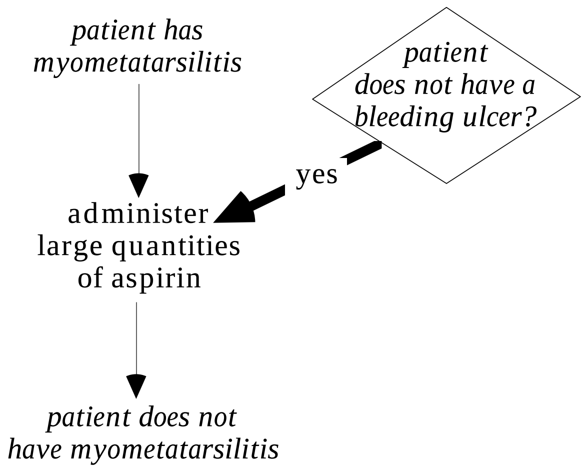
patient has myometatarsilitis
patient does not have a bleeding ulcer?
yes
administer large quantities of aspirin
patient does not have myometatarsilitis

Figure 17

7.6 Backup Plans

There is another way in which the introduction of a contingency node can select for circumstances with lower than average execution costs. In decision-theoretic planning, it will often be desirable to incorporate redundancy. If one plan for achieving a goal is not guaranteed to succeed, we may supplement it by setting a second plan in motion at the same time. For example, knowing that my wife may fail to get sugar, I may also ask my daughter to get sugar on her way home. This produces the plan of figure 18. Assuming that my wife and daughter shop at different stores, so the probability of either getting sugar is independent of the other, then if they each have a 70% probability of getting sugar, this plan gives me a 91% probability of having sugar. This illustrates that redundancy can improve a plan significantly. Decision-theoretic planners should be able to incorporate redundancy into their plans.

The down side of redundancy is that it can increase execution costs. In the preceding plan, there is a 49% probability that my wife and daughter will both get sugar, in which case I will have to pay for extra sugar. To avoid this, I might wait and see whether my wife got sugar before asking my daughter to get sugar, producing the plan of figure 19. The contingency node lowers expected-execution-costs (without lowering the expected-payoff) by calling the redundancy plan only when it is needed.

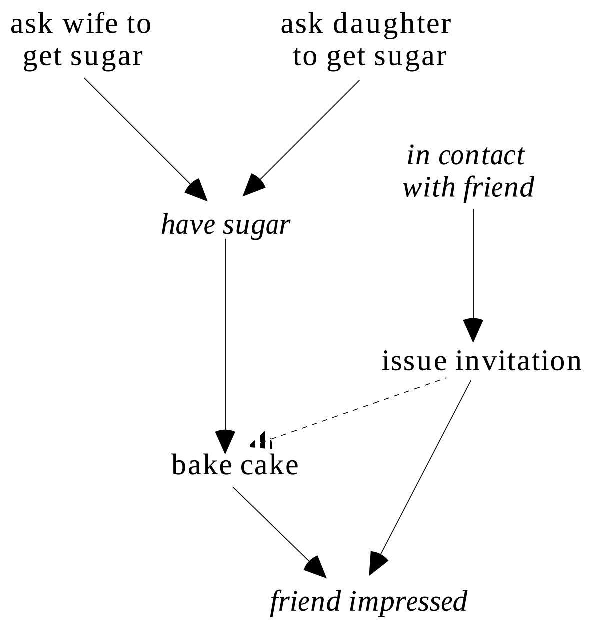

Figure 18

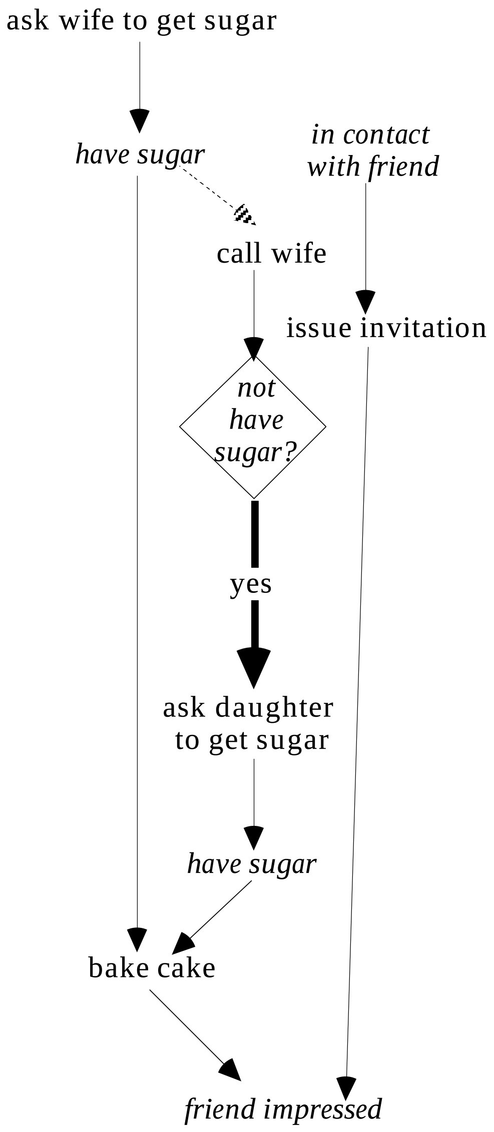

Figure 19

Figure 19 involves a *backup plan*. Just as it may be desirable to verify that an action achieved its desired purpose before continuing execution, it may be desirable to know if it failed to achieve that purpose and adopt a backup plan for what to do in that case. This is like enlightened execution, but instead of aborting execution if a step fails to achieve its purpose, a backup plan is called to achieve the goal in a different way. Note that the contingency in figure 19 is a cost contingency, lowering the expected-execution-costs of the original redundancy plan of figure 18. The proposal is then that the construction of this backup plan proceeds in two steps. First we refine the plan of figure 9 by adding a redundant subplan (increasing both the expected-payoff and the expected-execution-costs), and then we refine the redundancy plan by restricting its execution, thus lowering the expected-execution-costs. The contingency node has the structure of a dangling contingency, but its purpose is different. Dangling contingencies were payoff contingencies, restricting execution to cases in which the payoff is higher, whereas the contingency in a backup plan restricts execution to cases in which the redundant plan is necessary to maintain the payoff at its average level. By eliminating other cases, it eliminates cases in which execution costs are higher than necessary but the payoff is unchanged. So this illustrates the application of a different refinement operator. We might call the resulting contingencies *backup contingencies.*

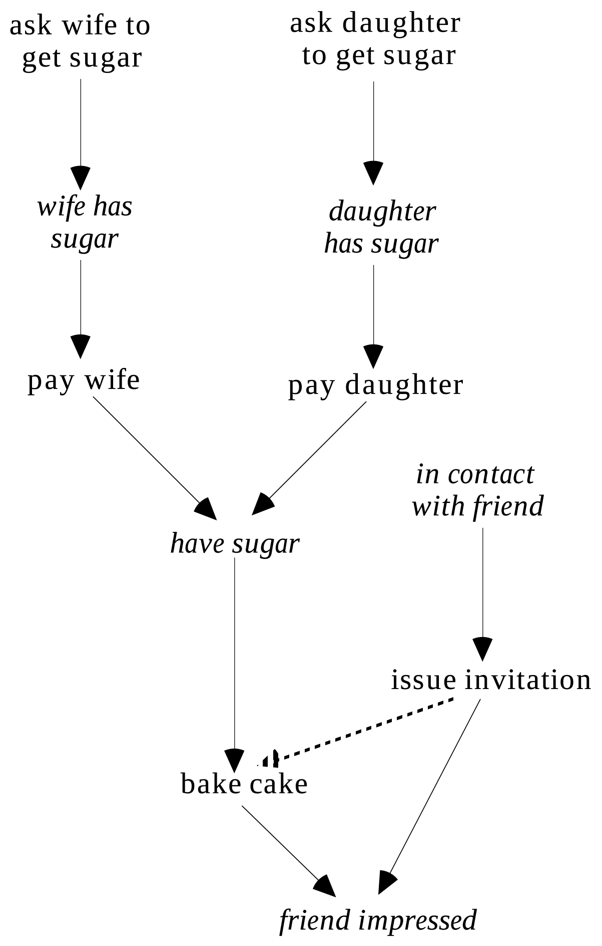

Figure 20

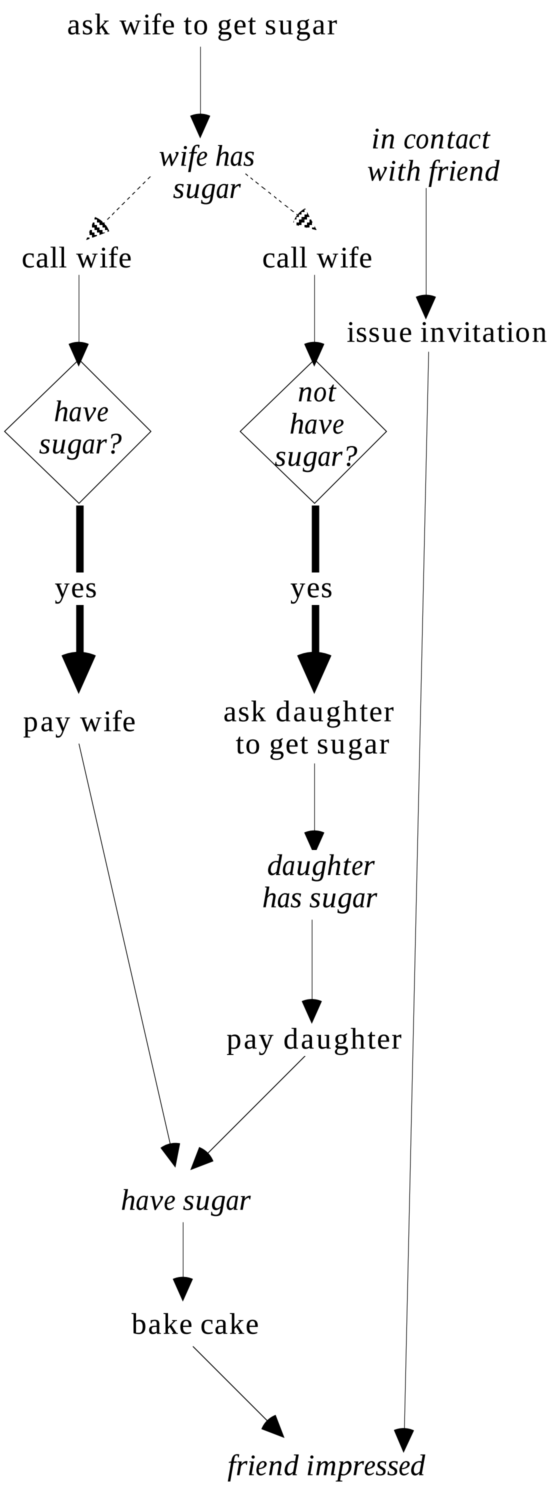

Figure 21

Normally, backup plans involve two contingencies rather than just one. As in figure 19, they have a contingency node that calls the backup plan, but they also have a contingency node that aborts execution of the original subplan. The latter is unnecessary in figure 19, because the original subplan for acquiring sugar is fully executed before the contingency can be checked. But suppose I have to pay my wife or daughter for the sugar before they will give it to me. Then the original redundancy plan is that of figure 20, and turning it into a plan with a backup subplan produces the plan of figure 21. The contingency *have sugar?* in figure 21 is a normal anchored contingency whose purpose is to restrict execution of that part of the plan to cases in which it contributes to a higher payoff. As such, the plan of figure 21 can be constructed from the backup plan having the single contingency *not have sugar?* using the refinement operator that normally produces anchored contingencies. No special refinement operator is required here.

The plan of figure 21 is a conditional plan, i.e., a plan containing a conditional fork. It contains one subplan prescribing what to do if my wife get sugar, and another subplan prescribing what to do if she does not. The production of such plans is the standard purpose of contingency planning in classical conditional planning. The production of plans with conditional forks is the only purpose previous authors have envisioned for contingency planning. Indeed, in classical contingency planning it is the only possible purpose, but as we have seen, in decision-theoretic contingency planning it represents only one of a number of possible uses.

7.7 Corrective Repair

Onder, Pollack and Horty (1998) describe a refinement operator they call *corrective repair* that produces conditional plans in either classical or decision-theoretic planners. This operator closes an open condition by appending a contingency node calling the plan that begins with that open condition only when the open condition is known to be true, and then initiates search for a backup plan for achieving the top-level goal when the open condition is known to be false. This is the way conditional planning works in classical conditional planners (CNLP, Cassandra, B-PRODIGY), and Onder, Pollack, and Horty incorporated it into the decision-theoretic planner Mahinur.

As we have seen, most uses of contingency planning in decision-theoretic contexts do not result in conditional plans. Perhaps the most common decision-theoretic contingencies are dangling contingencies or anchored contingencies. Corrective repair cannot produce these contingency plans. It is worth noting that, as described, corrective repair cannot produce the plan of figure 21 either. Corrective repair will only produce plans in which the first contingency is a dangling contingency and the second contingency concerns what to do if that dangling contingency fails. In figure 21, we instead have an anchored contingency and a plan for what to do if the anchored contingency fails. This seems to be a minor point, however. It appears that the corrective repair operator could be modified to reintroduce the open condition so that further planning could be aimed at its achievement, and that would allow the construction of the plan of figure 21. On the other hand, that will lead to the production of conditional forks even when there is a very secure plan for the achievement of the contingency (there might even be a plan that will achieve it with probability 1). That seems undesirable.

I don't find corrective repair an intuitively natural operator for the production of conditional plans. A more natural way of producing the plan of figure 21 is as described above. We first produce the plan of figure 9. Its expected-value is not high enough to satisfy us, so we add a redundancy plan as in figure 20. Then we refine the redundancy plan to eliminate unnecessary execution costs by adding a backup contingency to one branch of the redundancy plan, and finally we add the anchored contingency on the other branch, producing the plan of figure 21.

7.8 Planning for Ignorance

There is a further case of conditional planning that arises in decision-theoretic contexts but cannot arise in classical contexts. We sometimes plan for what to do if we know that *P* is true, and we plan to do something else if we don't know that *P* is true, i.e., if we either know that ~*P* is true or we are ignorant. For example, suppose our goal is to reach a certain destination, and we can do that by taking any of three trails through the mountains. Other things being equal, we prefer trail_1 to trail_2, and we prefer trail_2 to trail_3. However, we also know that either trail_1 or trail_2 is blocked by an avalanche (but not both). We might then plan to inquire locally about trail_1 when we reach the fork of the three trails. If we come to know that trail_1 is blocked, we will take trail_2. If we know that trail_1 is not blocked, we will take trail_1. But if we cannot ascertain whether trail_1 is blocked, we will not be able to infer anything about trail_2 either, so we will take trail_3. This kind of planning is not uncommon, but ignorance cannot arise in classical contexts where knowledge acquisition plans (like any other plans) are presumed to be guaranteed to achieve their goals.

We can also have backup plans for what to do when we do not know whether a contingency is satisfied, without those backup plans being part of a conditional fork. For example, in figure 19 I planned for what to do if I discover that my wife did not get sugar. However, I might instead plan for what to do if I cannot verify that she got sugar, i.e., if either I discover that she did not or I am unable to determine whether she did. For instance, my wife might not be home when I telephone. If it is sufficiently important that I have the sugar, I may produce a variant of the plan in figure 13 in which I ask my daughter to buy sugar on the way home if, at that time, I do not know that my wife got sugar. In this case the contingency is not *I believe that I do not have sugar*, but rather *I do not believe that I have sugar*. This backup plan will have higher expected-execution-costs, because there is the possibility that my daughter will buy sugar even though my wife already bought sugar, but it also has a higher probability of achieving the goal. We have to weigh the higher expected-execution-costs against the increased expectation of achieving the goal in deciding which of these plans has the higher expected-value.

Planning for the case of ignorance can be assimilated to contingency planning for the case in which the agent does not know that *P*. That will involve the subgoal *if you don't believe that P, believe that you don't believe that P.* At first glance, that seems unduly complex, but note that in order to execute the plan the agent would have to *believe* that he is ignorant — not just *be* ignorant — so perhaps this is a correct assimilation. An important question that remains is when the agent should engage in planning for ignorance (as opposed to planning for knowledge that the contingency is or isn't satisfied). A plausible proposal is that the agent should initially plan for knowledge of the truth or falsity of the contingency, and if that does not produce a plan with a high enough expected-value, then he should try adding a subplan for what to do in the case of ignorance.

7.9 Threat Resolution by the Introduction of Contingencies

Let us change the example of figure 20 a bit. Suppose that in order to get my wife or daughter to buy sugar for me, I must give them the money *first*. I am thus led to the redundancy plan of figure 22. However, suppose I do not have enough money to give them both what is needed to buy the sugar. In this case, giving my wife money undermines the causal link for getting sugar by having my daughter buy it, and vice versa. This undermining (i.e., threat — but see below) is indicated in figure 22 by the fuzzy arrows.

The result of the undermining is that adding the redundant subplan of asking my daughter to buy sugar does not increase the expected-value of the plan. This is because trying to execute either of the redundant subplans will preclude executing the other, so the expected-value of the plan is unchanged by adding the redundancy. One way to fix this is by executing the different redundant subplans under different circumstances. For example, if my wife shops at Safeway, but my daughter shops elsewhere, I might call Safeway to find out whether they have sugar. If they do, I ask my wife to get sugar, and if they don't I ask my daughter to get sugar. This produces the conditional plan of figure 23. The introduction of the pair of contingencies produces a play with a higher expected-payoff but the same expected-execution-costs (except for the knowledge costs).

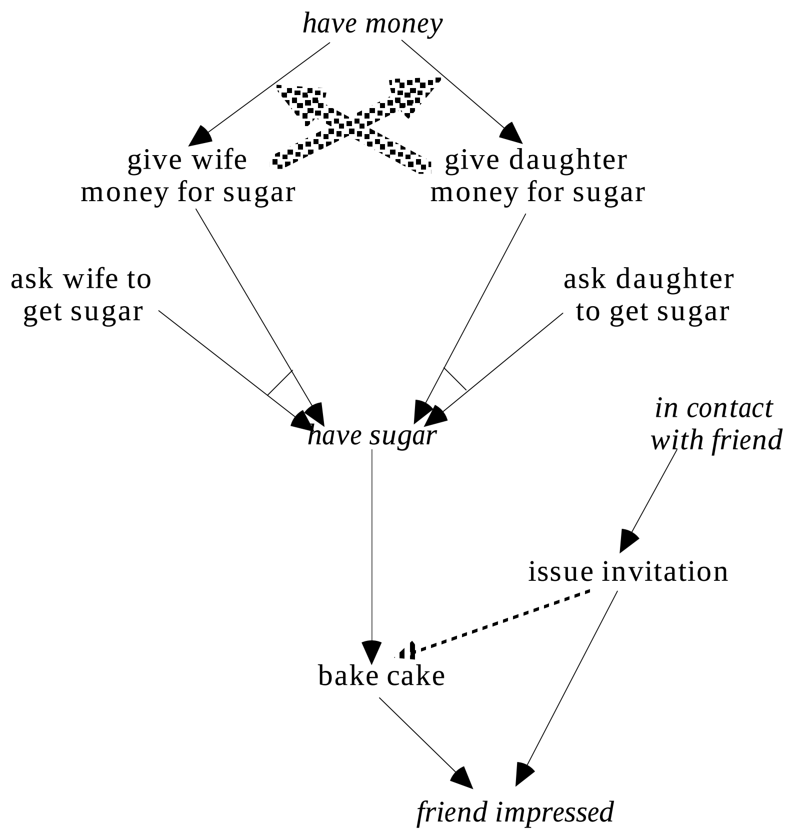

Figure 22

Resolving threats in this way produces a conditional plan. In this example, the threats are classical. That is, giving money to either my wife or my daughter absolutely precludes giving it to the other. However, in decision-theoretic planning there can be a wide variety of kinds of "decision-theoretic undermining" that do not occur in classical planning (for a partial survey, see Pollock 2000). Any of these varieties of undermining can be averted by a similar use of pairs of contingencies to produce a conditional fork.

The refinement process that I have just described is essentially the same as that performed by the sole refinement operator that C-Buridan (Draper, Hanks, and Weld 1994) uses for generating contingency plans (although C-Buridan is a probabilistic planner rather than a decision-theoretic planner, so it just appeals to probabilities and not to expected-values). However, this accommodates only a very special case of contingency planning. First, like corrective repair, this can only produce conditional plans, and not contingency plans with more general structures. Second, it cannot even produce all conditional plans. For example, it cannot produce the plan of figure 21. As before, it follows that a planning algorithm using this as its sole refinement operator for introducing contingencies will be neither sound nor complete.

What are the refinement operators that produce plans like that of figure 23? It appears that we need a primitive threat resolution operator that adds contingencies in pairs rather than one at a time, as does C-Buridan. Let us call these *threat resolution contingencies*. They will be a kind of payoff contingency, although of an unusual sort because the contingencies must be added in pairs. To make this refinement operator precise, we need rules for how to select the contingency. In the above example, we selected a dangling antecedent that increases the security (probability of success) of one of the redundant subplans. Perhaps that should be adopted as the general rule, although this is not entirely obvious.

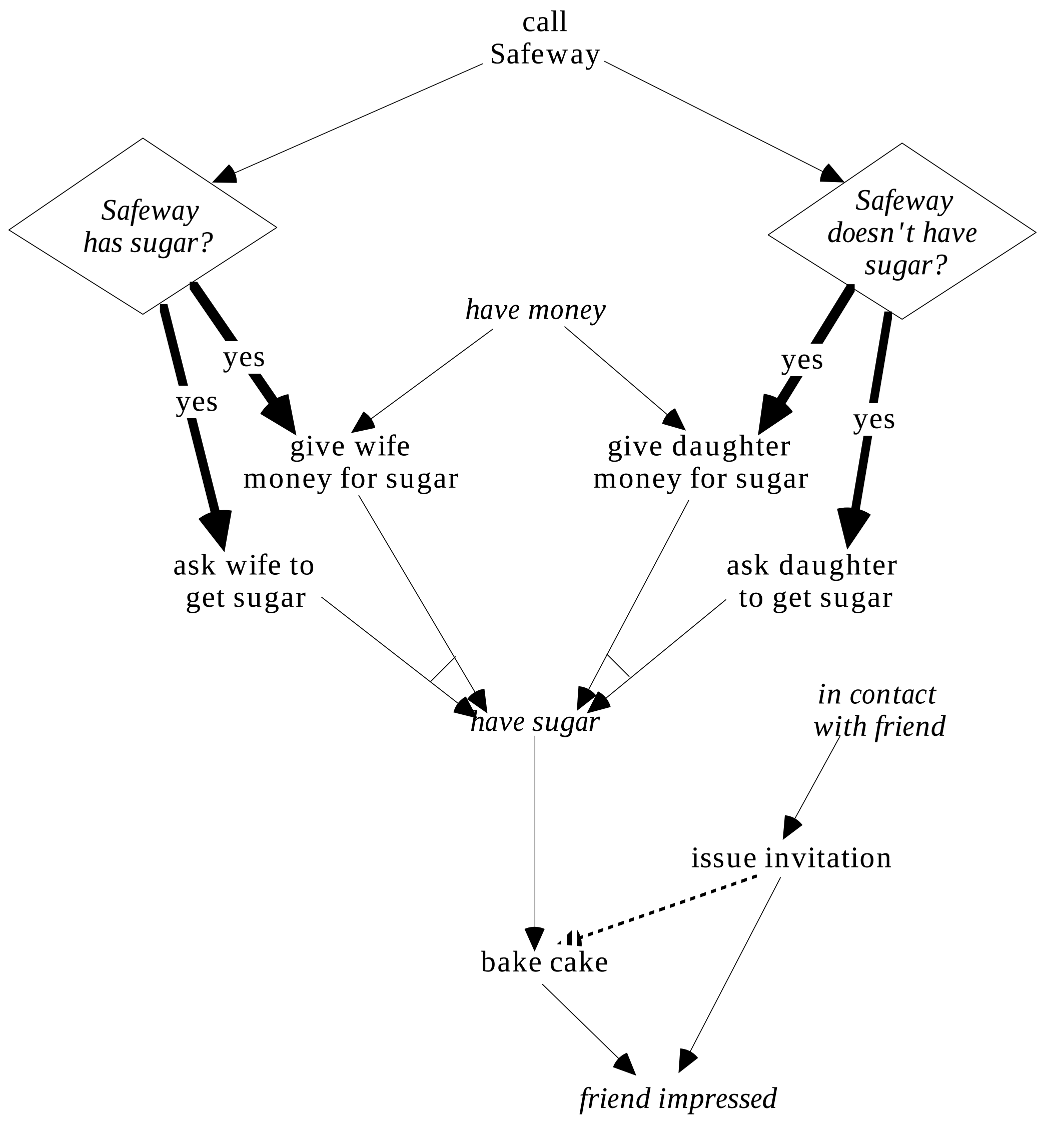

Figure 23

7.10 Contingent Confrontation Plans

In the STRIPS representation of actions, actions are supposed to have a fixed set of results that will ensue whenever they are performed. ADL (Pednault 1987) introduced "conditional effects", according to which actions may have different results when performed in different circumstances. In decision-theoretic planning, this generalizes to the observation that actions may produce different results with different probabilities in different circumstances.

SNLP (McAllester and Rosenblitt 1991) defined a *threat* to a causal link to be a step that can, consistent with the ordering constraints of the plan, occur between the root step and the consumer step of the link, and that prescribes an action having an effect that negates the goal of the link. SNLP resolves threats by adding ordering constraints. With the introduction of conditional effects, additional tools are required for threat resolution. Penberthy and Weld (1992) introduced *confrontation* (originally called *separation*) as a new threat resolution technique, and showed that when it is added to the threat resolution tools of SNLP, the resulting planner (UCPOP) is complete. When the threat results from a conditional effect, confrontation proceeds by adding the negation of an antecedent of the conditional effect as a new open condition for the planner. In effect, the planner searches for a subplan that will guarantee that the conditional effect will not occur.

In my (1998), I observed that in the presence of conditional effects, threats may not be “real”. I defined a subplan to *undermine* a causal link iff, consistent with the ordering constraints, it is a plan for producing the negation of the link goal between the root step and the consumer step of the link. If a threat is not part of an undermining, there is no need to resolve it. Resolving unreal threats can, for example, lead UCPOP to impose unnecessary ordering constraints. Underminings can be averted by either adding ordering constraints or by adding a *confrontation plan* that undermines the undermining subplan. I presume that decision-theoretic planning should work similarly, although as noted above, there are more varieties of possible undermining.

Confrontation in decision-theoretic planning can be illustrated with a modified version of Pednault’s briefcase example (Pednault 1987). Suppose my briefcase is at home, and I want it to be at the office. Suppose further that my briefcase is full of pennies, I fear that my daughter may have mixed my favorite Indian penny in with the pennies in my briefcase, and I want the Indian penny to remain at home. Taking the briefcase to the office will threaten the subgoal of having the penny at home. If I do not have time to search through the briefcase to see whether the Indian penny is there, I may resolve this by adopting a confrontation plan to dump the contents of the briefcase onto my desk and leave them there before taking the briefcase to the office. The process of constructing this plan is diagrammed in figures 24 - 27. In figure 24, a plan is constructed for having the briefcase at the office and the Indian penny at home. In figure 25, a subplan is found that undermines the causal link responsible for the Indian penny being at home. Note that this undermining is probabilistic, because I do not know for sure that the Indian penny is among the pennies in the briefcase. In figure 26, a confrontation plan is found that undermines the undermining plan, and the result is the plan of figure 27.

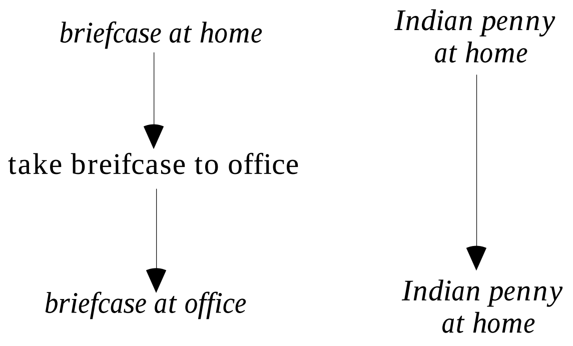

Figure 24

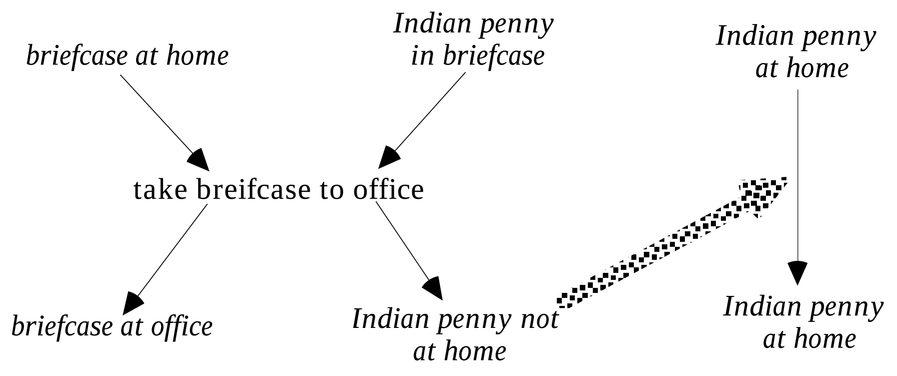

Figure 25

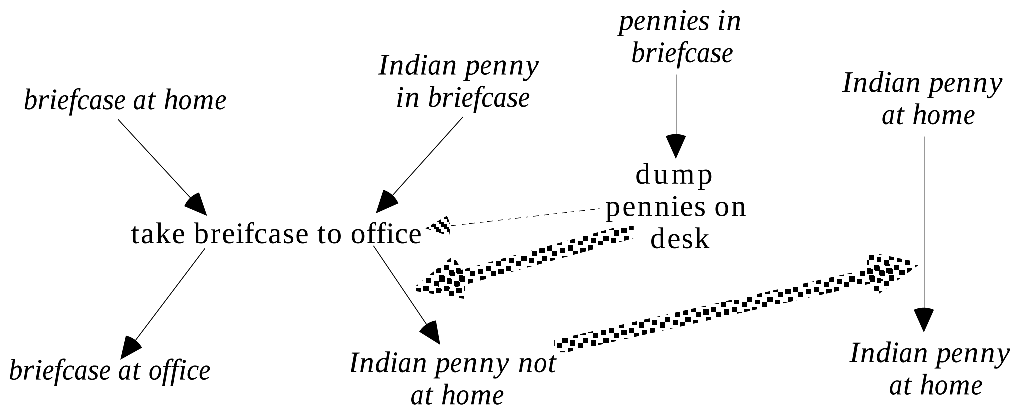

Figure 26

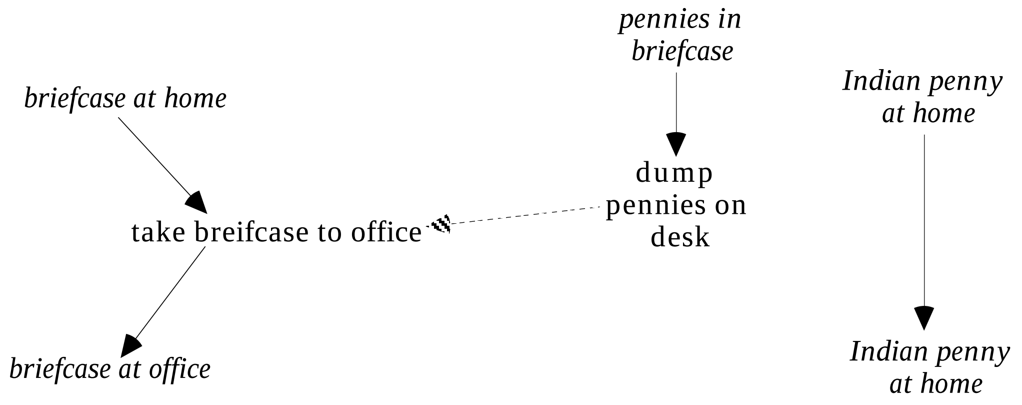

Figure 27

Now change the example. Suppose that it is my dictionary that I want to have at home, and my daughter may have left my dictionary in my briefcase. I could adopt a confrontation plan prescribing the removal of the dictionary from the briefcase by turning the briefcase upside down and emptying its contents onto my desk. However, because the dictionary is easily spotted among the contents of the briefcase, one can lower the execution costs by checking to see whether the dictionary is there before executing this confrontation plan. So the confrontation plan can be made contingent on the dictionary being in the briefcase, as in figure 28.

Interestingly, the addition of contingency nodes to confrontation plans does not require a special refinement operator. The same operators that create dangling and anchored contingencies will handle contingent confrontation plans. In the above example, we are just adding a dangling contingency. If we changed the example so that an earlier action resulted in the dictionary being in the briefcase with a certain probability, then we could generate a contingent confrontation plan by adding an anchored contingency.

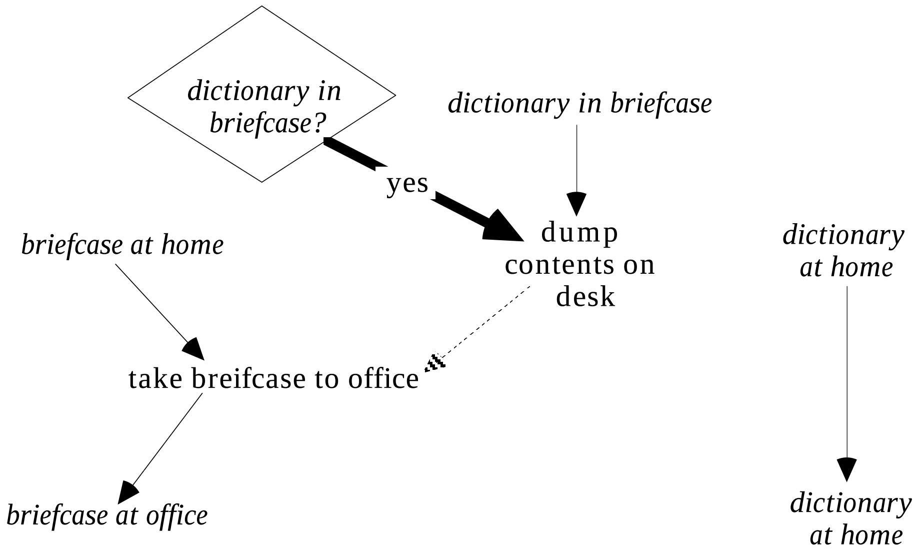

Figure 28

7.11 Contingencies Aimed at Changing Costs and Values

Thus far, all the kinds of contingencies that have been discussed have functioned by changing the probabilities that plan steps will be executed, the probabilities that goals and subgoals will be achieved, or the probabilities of execution costs being incurred. However, the expected-value of a plan is determined not just by these probabilities, but also by the values of the goals and execution costs. These values and costs will typically exhibit some context dependence. For example, if my goal is to eat a dish of vanilla ice cream, the value of that goal will be seriously diminished by eating a dill pickle first. Accordingly, I may be able to improve the expected-value of my plan by making it contingent on my not having just eaten a dill pickle. Similarly, in a transportation plan, the cost of transporting a package already in the truck to a particular destination will be determined in part by how far the truck is from that destination. Thus a plan to deliver the package might be improved by making it contingent on the truck being within a certain distance of the destination. Adding these contingencies may result in the plan not achieving its goals in some cases, but recall that decision-theoretic plans need not be guaranteed to achieve their goals — they just have to have reasonable expected-values.

The context sensitivity of values underlies one of the kinds of decision-theoretic undermining discussed in my (2000). One plan can undermine another by lowering the values of the goals achieved or raising the assessment of the execution costs. This might be called *evaluative undermining.* A fully adequate decision-theoretic planner should contain refinement operators for dealing with evaluative undermining, although no existing decision-theoretic planners do. Among these refinement operators will be operators adding contingency nodes as above. However, the details cannot be worked out until we see how the underlying decision-theoretic planner handles such evaluative undermining in the first place.

7.12 Plans with Loops

The last use of contingencies that I will consider is in plans with loops. Consider a soldier being attacked by an enemy soldier. He wants to stop the attack, and the only way he can think to do it is by shooting at the enemy soldier. So he constructs the plan of figure 29. However, being an imperfect shot, the soldier realizes that he may fail to stop the attack with a single shot. So he adopts interest in a backup plan for what to do if he fails, as in figure 30.

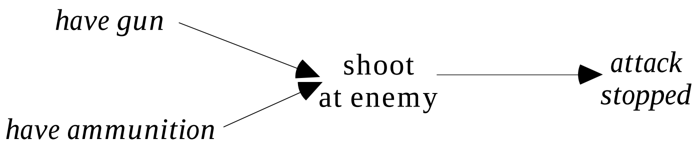

Figure 29

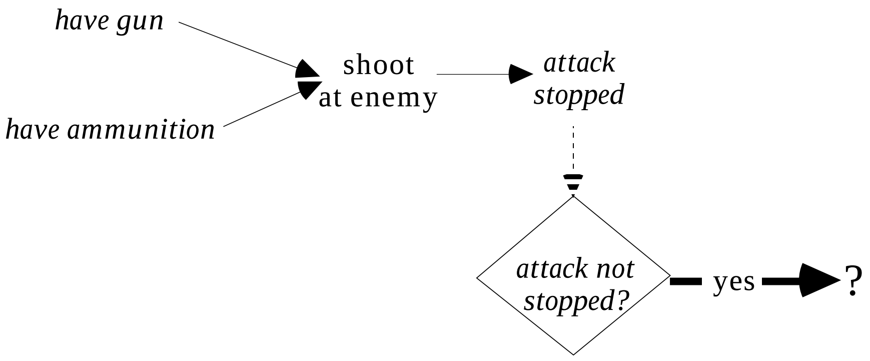

Figure 30

The planning problem becomes that of deciding what to do if the attack has not been stopped. One solution is to shoot at the enemy again, and keep doing so until there is no more ammunition. This produces the plan of figure 31. The contingency node calls a subplan (in this case, the whole plan). There must be a termination condition for the loop, and that is provided by the second contingency node, checking for ammunition. If the contingency *have ammunition* is not satisfied, execution stops. This plan might be embellished further by adding another contingency plan for what to do when the soldier runs out of ammunition.

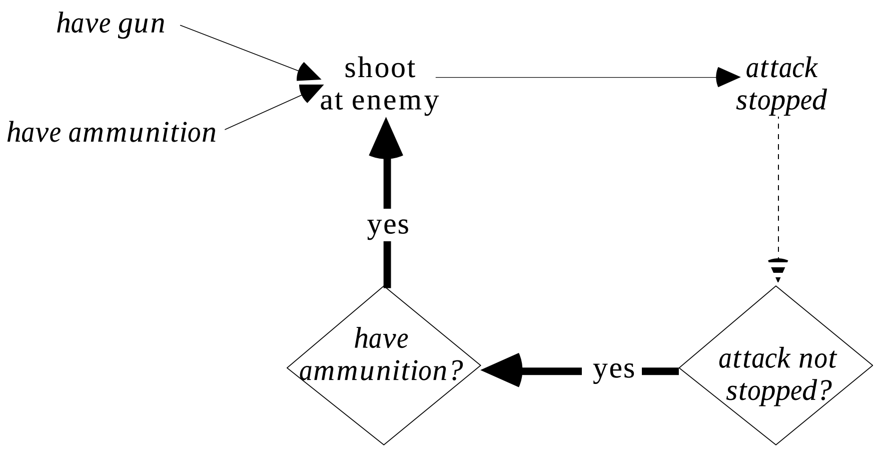

Figure 31

The ability to produce plans with iteration will ultimately be important, and contingency nodes provide the mechanism for doing that. However, the nature of the refinement operator that adds such contingencies is a matter for future investigation.

8. Conclusions

Decision-theoretic contingency planners can be divided roughly into state-space planners and POCL planners. State-space planners first build a world model and then distill a plan out of it. POCL planners try instead to build a plan from the bottom up, relying entirely upon local relationships. State-space planners are in principle incapable of handling some planning problems that humans find trivial. The difficulty traces to their reliance on global reasoning, which in complex cases is just too big. I take this to motivate further investigation of decision-theoretic POCL contingency planners.

Although decision-theoretic POCL contingency planners appear to have no difficulty with the kinds of simple problems that state-space planners choke on, they are subject to other difficulties centering around contingency-completeness. I defined contingency-completeness as requiring of a planning algorithm that, given sufficient time for solving a planning problem, whenever it produces a plan, if the plan can be improved by inserting a contingency then the algorithm is in principle capable of finding and inserting the contingency. I proposed this as a criterion of adequacy for any decision-theoretic contingency planner, and noted that planners that do not satisfy it will in all likelihood fail to be either sound or complete. I then turned to an investigation of what contingency-completeness requires of a decision-theoretic POCL planner.

Classical contingency planning has sometimes been touted as the first step towards the development of a decision-theoretic planner (Pryor and Collins 1996). It is based upon a relaxation of just one of the assumptions of classical planning, viz., the assumption that the planning agent has complete knowledge of the initial conditions. However, this is an awkward compromise. Once it is acknowledged that the initial state is uncertain, it ought to be acknowledged that the results of actions are also uncertain. And from here it is a short step to the conclusion that, Einstein aside, God does play dice with the universe. In other words, rational choice must be based upon decision-theoretic reasoning.

What makes classical contingency planning an awkward compromise is that the only kind of contingency planning possible in classical contexts is the construction of plans with conditional forks, and the only obvious way to construct such plans is with corrective repair in the sense of

Onder, Pollack, and Horty (1998). In decision-theoretic contexts, this is not sufficient to achieve contingency-completeness, and it may be the least common use of contingencies. We no longer search for plans that are guaranteed to achieve their goals. Instead, we search for plans having high expected-values. And most of the ways the introduction of contingency nodes can improve a plan do not involve the construction of conditional forks. Most contingencies are "single contingencies" that turn off execution of parts of a plan when the contingency fails, without telling us to do something else instead. In general, contingencies work by restricting the execution of a plan to cases in which it has higher payoffs or lower execution costs. There are two ways the introduction of a contingency node can increase the expected-value of a plan. It can act as a *payoff contingency*, selecting for cases in which the payoff is higher than usual without the execution costs also being enough higher to offset the increase in the payoff. Alternatively, it can act as an *execution contingency*, selecting for cases in which the execution costs are lower than usual without the payoff being enough lower to offset the decrease in execution costs. We found a number of different circumstances in which payoff contingencies or execution contingencies can be introduced into a plan:

<u>Payoff contingencies</u>

- dangling contingencies
- anchored contingencies
- contingencies that increase the probability of the antecedent of a causal link
- contingencies that increase link probabilities
- pairs of threat-resolution contingencies
- contingencies that increase the value of a goal
- contingencies introducing iteration

<u>Cost contingencies</u>

- contingencies that rule out side effects
- contingencies upon which backup plans depend
- contingencies that decrease an execution cost

In addition, for each of these contingencies there is a distinction between a contingency based upon knowledge and one based upon ignorance. That is, plan execution can be made dependent either on believing that a contingency is satisfied, or on not believing that it fails.

It is unclear whether the above list is complete. It is noteworthy, however, that the single kind of contingency introduction possible in classical contingency planning — corrective repair — does not appear anywhere on this list. It could be introduced as a special-purpose operator, but logically it can be seen to be the result of combining several other operations. First, normal decision-theoretic refinement operators will add redundant subplans for achieving a goal. One of the redundant subplans can be turned into a backup plan by introducing a contingency precluding its execution when it is unnecessary. This is a cost contingency. Then a conditional fork can be generated by adding a contingency to the other redundant subplan precluding its execution when it is unlikely to succeed. This is a payoff contingency.

In classical contingency planning, corrective repair seems to be forced upon us as the only way contingency plans can be generated. However, classical contingency planning is such a strained compromise between classical planning and decision-theoretic planning that there is little reason to expect its properties to generalize to decision-theoretic contingency planning. It wouldn't have been surprising (or shouldn't have been) if it had turned out that there was *no* reasonable algorithm for constructing classical contingency plans. That they were found to be producible by corrective repair might show more about the ingenuity of Warren and of Peot and Smith than about the nature of contingency planning itself.

This is a theoretical paper aimed primarily at raising difficulties for existing approaches to decision-theoretic planning. This paper does not describe an implemented system. It investigates contingency-completeness as a criterion of adequacy for decision-theoretic POCL planners. There are no existing implemented POCL planners that satisfy this criterion of adequacy. In particular, there are no implemented systems that are capable of constructing all of the plans discussed above. The issues discussed in this paper are orthogonal to such issues as what the planning language is, are actions allowed to have conditional effects, how expected-values are to be computed, etc. However, those issues are decided, a decision-theoretic planner should still have to satisfy this criterion of adequacy.

Although this paper does not describe an implemented system, it may provide guidance in constructing planning algorithms that are contingency-complete. Each of the kinds of contingencies listed above were characterized in terms of the circumstances under which they should be introduced into a plan. As such, each corresponds to a refinement operator for decision-theoretic POCL contingency planning. A natural way of constructing an algorithm for decision-theoretic POCL contingency planning is then to begin with a refinement planner for decision-theoretic planning and add refinement operators for introducing contingency nodes into plans. For this to work, some of the contingency operators sketched above must be made more precise. We must also ask whether the above list of contingencies is complete, and whether there might be alternative ways of describing some of the contingencies that would lead to different refinement operators, or possibly to operators subsuming several different kinds of contingencies.

The preceding is just intended to be a plausible suggestion. At this stage it would be silly to ask for some kind of evaluation (experimental or analytic) of the proposal that a planning algorithm be constructed in this way. There is no algorithm to be evaluated, and even if there were there would be nothing to compare it with because existing planning algorithms cannot solve these planning problems.

9. Bibliography

Aström , K. J.
1965 "Optimal control of Markov decision processes with incomplete state estimation", *J. Math. Anal. Appl.* **10**, 174-205.
Bellman, R.
1957 *Dynamic Programming.* Princeton University Press, Princeton, N.J.
Blum and Furst
1995 "Fast planning through planning graph analysis". In *Proceedings of the Fourteenth International Joint Conference on AI,*, 1636-42.
1997 "Fast planning through planning graph analysis", *Artificial Intelligence* **90**, 281-300.
Blythe, Jim, and Manuela Veloso
1997 "Analogical replay for efficient conditional planning", *AAAI97.*
Boutelier, Craig
1997 "Correlated action effects in decision-theoretic regression", *UAI97,* 30-37.
Boutelier, Craig, Thomas Dean, and Steve Hanks
1999 "Decision theoretic planning: structural assumptions and computational leverage", *Journal of AI Research* **11**, 1-94.
Boutelier, Craig, and R. Dearden
1996 "Approximating value trees in structured dynamic programming", *Proceedings of the Thirteenth International Conference on Machine Learning,* 54-62.
Boutelier, Craig, R. Dearden, and M. Goldszmidt
1995 "Exploiting structure in policy construction", *IJCAI95*, 1104-1111.
Boutelier, Craig, and David Poole
1996 "Computing optimal policies for partially observable decision processes using compact

representations", *AAAI96*, 1168-1175.

Collins, G. C.

1987 "Plan creation: using strategies as blueprints", Technical report YALEU/CSD/RR 599, Department of Computer Science, Yale University.

Diettrich, T. G., and N. S. Flann

1995 "Explanation-based learning and reinforcement learning: A unified approach", *Proceedings of the Twelfth International Conference on Machine Learning*, 176-184.

Draper, Denise, Steve Hanks, and Daniel Weld

1994 "Probabilistic planning with information gathering and contingent execution", *Proceedings of AIPS94.*

Hansen, E. A., and Z. Feng

2000 "Dynamic programming for factored POMDP's", *AIPS2000 Workshop on Decision-Theoretic Planning*, 41-48.

Hoey, J., R. St-Aubin, and C. Boutilier

1999 "SPUDD: Stochastic planning using decision diagrams", *UAI99.*

Kautz, H. A., and B. Selman

1996 "Pushing the envelope: planning, propositional logic, and stochastic search", *Proceedings of AAAI-96*, 1194-1201.

1998 "Blackbox: a new approach to the application of theorem proving to problem solving", in *AIPS98 Workshop on Planning as Combinatorial Search*, 58-60.

Kushmerick, Hanks and Weld

1995 "An algorithm for probabilistic planning". *Artificial Intelligence* **76**, 239-286.

Majercik and Littman

1998 "MAXPLAN: A new approach to probabilistic planning", *AIPS98*, 86-93.

1999 "Contingent planning under uncertainty via stochastic satisfiability", *AAAI99.*

McAllester, David, and Rosenblitt, David

1991 "Systematic nonlinear planning", *Proceedings of AAAI-91*, 634-639.

Onder, Niluger, and Martha Pollack

1997 "Contingency selection in plan generation", *ECP97.*

1999 "Conditional, probabilistic planning: a unifying algorithm and effective search control mechanisms", AAAI 99.

Onder, Niluger, Martha Pollack, and John Horty

1998 "A unifying algorithm for conditional, probabilistic planning", *AIPS1998 Workshop on Integrating Planning, Scheduling, and Execution in Dynamic and Uncertain Environments.*

Pednault, Edwin P.

1987 Toward a Mathematical Theory of Plan Synthesis, PhD dissertation, Stanford University.

Penberthy, J. Scott, and Weld, Daniel

1992 "UCPOP: a sound, complete, partial order planner for ADL". *Proceedings 3rd International Conference on Principles of Knowledge Representation and Reasoning*, 103-114.

Peot, M. A., and D. Smith

1992 "Conditional nonlinear planning", *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, 189-197.

Peterson, Gilbert

2001 *Decision-Theoretic Planning in a Robot Architecture*, PhD dissertation, University of Texas at Arlington.

Peterson, Gilbert, and Diane J. Cook

2000 "Decision-theoretic planning in the Graphplan framework", *AIPS200, Proceedings of the Workshop on Decision-Theoretic Planning* 69-76.

Pollock, John

1998 "The logical foundations of goal-regression planning in autonomous agents", *Artificial Intelligence* **106**, 267-335.

2000 “Locally global planning”, *Proceedings of the Workshop on Decision-Theoretic Planning*, AIPS2000.

2001 “The logical foundations of decision-theoretic planning in autonomous agents”, in preparation, available at http://www.u.arizona.edu/~pollock.

Pryor, Louise, and Gregg Collins

1996 “Planning for contingencies: a decision-based approach”, *Journal of Artificial Intelligence Research* **4**, 287-339.

Simon, Herbert

1955 “A behavioral model of rational choice”, *Qart. J. Economic*s, **59**, 99-118.

Smallwood, R. D., and E. J. Sondik

1973 “The optimal control of partially observable Markov processes over a finite horizon”, *Operations Research* **26**, 282-304.

Warren, D.

1976 “Generating conditional plans and programs”, in *Proceedings of the Summer Conference and AI and Simulation of Behavior.*