What are filter conditions for?

Gregg Collins and Louise Pryor
The Institute for the Learning Sciences
Northwestern University
1890 Maple Avenue
Evanston IL 60201
[collins, pryor]@ils.nwu.edu

Abstract

Many extensions to the basic STRIPS action representation have been proposed. Many of them amount to the use of *filter conditions*, preconditions which do not become subgoals. The purposes to which filter conditions have been put and the justifications for their use have seldom been made clear. In this paper we clarify and examine the various arguments that appear to lie behind their use. We conclude that there are two major flaws in their use: their interpretation is inconsistent with commonly used planning techniques, and their use often leads to incompleteness. Moreover, their use generally addresses special cases of more general problems, and is not extensible to the general case.

1 Introduction

Planning systems solve problems by reasoning about actions and their expected effects; the representation of actions has therefore long been a central issue in planning research. Many current planning systems use representation schemes based on the STRIPS operator (Fikes & Nilsson, 1971). Such systems often incorporate modifications to the STRIPS operator aimed at improving its efficiency, its expressiveness, or both. A common modification is the use of *filter conditions* (Charniak & McDermott, 1985; Tate, 1977; Wilkins, 1988).

It has however seldom been made clear what the purpose of filter conditions is. Charniak and McDermott state that "It is usually obvious what conditions belong in which category [filter conditions or enabling preconditions]" but do not elaborate further. In general other researchers are not much more forthcoming on the subject.[1] In this paper we shall clarify the various arguments that appear to lie behind the use of filter conditions. Furthermore, we shall argue that, whatever the end being served, filter conditions are in general a flawed means of achieving it.

There are two major flaws. First, the interpretation of filter conditions is at best problematic and at worst inconsistent with classical planning techniques. Second, upon investigation filter conditions usually turn out to be methods of pruning the search space, based on assumptions that may fail in some circumstances. The use of filter conditions can therefore lead to incompleteness. Moreover, filter conditions usually address special cases of more general problems that must in any case be addressed in the design of a planning algorithm. If a general mechanism is available, there is no point in using filter conditions to deal with the special case; and where there is no general mechanism, there is no way in which filter conditions can be generalized.

[1]Possible exceptions are (Feldman & Morris, 1990), and (Currie & Tate, 1991); we will consider their work in some detail below.

We start our discussion with a description of what filter conditions are and their intended interpretation, and demonstrate that this interpretation is impossible to integrate effectively into most planners. The second half of the paper consists of an analysis of the various uses to which filter conditions have been put and their suitability for those purposes.

2 What are filter conditions?

A filter condition is a special type of precondition that can be used to extend the basic STRIPS representation[2] of actions. The preconditions of an operator become new goals (or *sub*goals) for the planner when that operator is included in the plan. We shall call preconditions that become subgoals *enabling* preconditions.

A filter condition, in contrast, is a precondition that does *not* become a subgoal, but rather states a requirement that must be met if the operator is to be selected for inclusion in the plan to begin with. For instance, an operator to turn on a light might have a precondition that the switch be *off* initially, but it would make no sense to turn the switch off in order to apply the operator. Filter conditions are useful because in certain situations planning to achieve an unmet subgoal may lead to inefficient or nonsensical plans. The precondition *switch off* in our example should be made a filter condition rather than an enabling precondition because it would not make sense to make it into a subgoal.

In many systems, the basic STRIPS operator has been extended to allow the specification of filter conditions as well as enabling conditions. This is particularly common in systems that are intended to be of practical use (Currie & Tate, 1985, 1991; Tate, 1977; Wilkins, 1988). Such systems typically incorporate other extensions to the basic STRIPS representation as well, mainly involving the introduction of different types of preconditions and annotations to the preconditions. Many of the modifications that have been introduced are in fact special cases of filter conditions, as we shall see later on in this paper.

Filter conditions are meant to act as constraints on the selection of operators. If the operator's filter conditions are true, it may be selected; if they are not, it may not. This deceptively simple description hides some complexity, however. In particular, the condition on selection is more accurately stated as "the operator's filter conditions would be true at the time when it would be executed, were the operator in fact selected". The computation of this complex condition may be tricky.

For example, consider the following problem involving a hungry graduate student. She can satisfy her hunger by eat-

[2]The basic STRIPS operator comprises the *preconditions* for executing an action, a list of the facts that become true as a result of executing the action (the *add-list*), and a list of the facts that become false as a result of executing the action (the *delete-list*).

Action:	eat-dry-cereal	eat-wet-cereal	buy-cereal
Filter conditions:	lack-milk	have-milk	
Preconditions:	have-cereal	have-cereal	lack-cereal
Delete list:	hungry	hungry	lack-cereal lack-milk
Add list:	thirsty		have-cereal have-milk

Figure 1: Modified STRIPS representation of a cereal domain

ing cereal. If she has milk, she will eat the cereal with milk, but if she has no milk she must eat it dry, in which case she becomes thirsty. If she has no cereal, she can go to the store to buy some; whenever she is at the store, she also buys milk. The modified STRIPS operators for this (somewhat contrived) domain are shown in figure 1.

Now suppose that the student is hungry, and lacks both cereal and milk. Her goal is to assuage her hunger. There are two operators she could use: she could eat cereal either wet or eat cereal dry. In order to decide which she will do at the time that she is constructing her plan, she must make a judgment as to which operator will be applicable. Examining the filter conditions at planning time, she sees that the *eat-wet-cereal* operator requires that there be milk. Since she lacks milk, she rejects this operator and decides on the *eat-dry-cereal* operator instead. Since she does not have any cereal, she sets up a new subgoal to acquire some, which she can achieve by using the *buy-cereal* operator. At this point her plan consists of the two operators: *buy-cereal* and *eat-dry-cereal*, in that order. Although all the preconditions in the plan are now satisfied, it turns out that the filter condition for *eat-dry-cereal* will not in fact be satisfied at the time when the action comes to be executed, because the action of buying cereal results in the purchase of milk as well as the purchase of cereal, so that the condition *lack-milk* will not be true. The planner has thus made a mistake due to its inability to guess correctly whether a filter condition would hold at the time when an operator was executed.

The problem in the above example came about because the planner added an operator to the plan affecting the value of the filter condition of an operator that had previously been scheduled. If new operators can be inserted at any point in the plan, then it is impossible to be sure what the situation will be when a particular action is executed until the plan is complete. It is therefore impossible to use filter conditions in such a planner to rule out alternative operator choices, since they cannot be used to rule out partial plans. An important aspect of the rationale for filter conditions—ruling out unreasonable uses of individual operators in partial plans—is therefore missing. This argument implies that filter conditions will be ineffective in all non-linear[3] planners (because the scheduling of operators may be changed during the planning process). We know of no strictly linear planners that are the subject of current or recent research, leading us to the conclusion that the use of filter conditions is in general ineffective in present-day planning systems.

3 What's the use of filter conditions?

Why, then, are filter conditions used? As we noted above, the reasons for their use are generally left implicit by researchers in the field. In this section we clarify and make explicit the reasons that we believe lie behind the various examples of their use. When these reasons are analyzed, it appears that most of them rely on assumptions about the outcome of the planning process. By making these assumptions, the planner can ignore parts of the search space that do not correspond to them and thus enhance the efficiency of the planning process. However, as the assumptions cannot be guaranteed to hold (after all, planning is undecidable), a planner relying on them is not complete[4].

[3]Nonlinear planners are those in which actions aimed at achieving the preconditions of different goals can be interleaved. Note that all partial order planners are non-linear.

Action:	(move-from-wall ?block)
Filter conditions:	(glued-to-wall ?block)
Preconditions:	
Delete list:	(glued-to-wall ?block)
Add list:	(on ?block table)

Figure 2: Modified STRIPS operator to move a block from a wall (adapted from Charniak & McDermott, 1985)

A secondary theme that emerges from our analysis is that filter conditions usually handle a few special cases of a more general planning problem. Since the generalized problems must be addressed in the design of planning algorithms anyway, the use of filter conditions appears to be superfluous.

3.1 Plan efficiency

Filter conditions can be used to block the construction of inefficient plans by preventing the use of particular operators in certain situations. For example, Charniak and McDermott describe a blocks world in which there is a special operator for getting a block onto the table when it is glued to the wall.[5] They argue that this operator should be given the filter condition "the block is glued to the wall" (see figure 2), preventing its use in cases in which the block is not initially glued to the wall. The alternative would be to make "the block is glued to the wall" an enabling precondition for the operator, which would allow the possibility that the planner would glue a block to the wall and then remove it in order to get that block onto the table. Charniak and McDermott's point is that such a plan is obviously inefficient, and the planner should be prevented from generating it.

The general idea behind the use of filter conditions to avoid inefficient plans is this: if you know in advance that in certain circumstances operator O will not be the best method of achieving its result, then don't use operator O in those circumstances. Such knowledge in effect anticipates the outcome of the planning process. That is, it involves a claim about the availability of alternative plans to that operator. Since in general it is not possible to find easy-to-compute features that infallibly predict the outcome of the planning process[6], in general such assertions will be *assumptions*. For example, it is a plausible assumption that gluing a block to the wall and then ungluing it will not be the best way to get the block onto the table, but it is not necessarily true. Its truth depends upon the availability of

[4]"Condition typing allows information to be kept [about the role of a condition]. However use of this information itself will almost certainly commit the planner to prune some of the potential search space thereby losing claims of completeness [if inappropriate condition types are used]." (Currie & Tate, 1991).

[5]SIPE makes similar use of filter conditions to avoid inefficient plans (Wilkins, 1988).

[6]If it *is* easy to find such features, then it is probable that "planning" in the domain in question can be carried out by an algorithm that is much simpler than a general-purpose planner.

Action:	(move ?block ?new)
Filter conditions:	(on ?block ?base) (inst ?base block) (inst ?new block)
Preconditions:	(clear ?block) (clear ?new)
Delete list:	(on ?block ?base) (clear ?new)
Add list:	(on ?block ?new) (clear ?base)

Figure 3: Modified STRIPS operator to move a block from one block to another (adapted from Charniak & McDermott, 1985)

other methods for moving the block in question. Clearly, building such an assumption into the planner means that the planner cannot be complete, since it may fail to find viable plans in cases in which the assumption fails.

Moreover, the method of avoiding inefficient plans using filter conditions is *ad hoc*—it does not suggest a general method for avoiding such plans. A better approach to avoiding inefficient plans is to control the search process so that more efficient plans are found first. All that is needed is a function that can be used to estimate the cost of completed plans, and to use some form of best-first search. This approach is preferable to the use of filter conditions for two reasons. First, no valid partial plan is ever completely ruled out; relatively unpromising plans are simply avoided until there are no better alternatives. Second, rather than requiring the programmer to discover propositions that imply the unfitness of particular operators in particular circumstances, this approach simply requires methods for estimating the costs of executing operators and achieving subgoals.

3.2 Indicating unachievable goals

Filter conditions can be used to prevent the planner from pursuing subgoals that cannot be achieved by any known action. For example, figure 3 shows an operator for moving a block from one supporting block to another. Since this operator is only applicable if both the original location and its destination are other blocks, "the current support is a block" and "the target support is a block" must be either filter conditions or enabling preconditions. Since there is no action in the standard blocks world that results in the creation of a new block, it has been suggested that such preconditions should in fact be filter conditions (Charniak & McDermott, 1985).[7]

We believe that the argument for this use of filter conditions is specious. The method relies on an implicit assumption that a particular condition such as "the object's support is a block" is unachievable by the planner. If we view the set of planning operators as being open rather than closed, then such an assumption may be invalidated at any time by the addition of a new operator. Building assumptions of this type into planning operators means that when a new operator is added any number of old operators may be invalidated. In effect, using filters in this way constitutes a breakdown of modularity in the definition of operators, and has the undesirable effect that lack of modularity has in any design process.

[7]O-Plan supports a similar use of filter conditions, termed *only-use-if* preconditions (Currie & Tate, 1991).

Action:	(pickup ?block)	(put-down ?base)
Preconditions:	(clear ?block) (empty-hand) (on ?block ?base)	(clear ?base) (holding ?block)
Delete list:	(empty-hand) (on ?block ?base)	(clear ?base) (holding ?block)
Add list:	(holding ?block) (clear ?base)	(on ?block ?base) (empty-hand)

Figure 4: STRIPS operators in a blocks world domain

Such a violation of modularity might be acceptable if it were offset by a corresponding benefit, but there is no such benefit in this case. If an operator is given an enabling precondition that is unachievable by any known operator, the planner will simply fail to find an operator to achieve that precondition, and the plan will fail if the condition is not true *a priori*. The planner will of course pay the cost of checking to see whether any operator can achieve the condition, but there is no reason to believe that in general the processing overhead incurred in treating the condition as a filter condition would be any less than the overhead incurred in treating it as an enabling precondition. The use of filter conditions thus appears to have no advantage over relying on the general search mechanism in this case.

3.3 Loops

Filter conditions can be used to avoid certain kinds of loops. Consider, for example, a blocks world domain with two actions: picking a block up, and putting a block down on another block (see figure 4). Suppose that a plan is required that will move block A from block B to block C (see figure 5). Block C must be clear before block A can be put on it, and since block C is clear in the initial configuration common sense suggests that the planner should simply take advantage of this and proceed to move block A. However, there is another possible approach, which is to ensure that block C is clear by removing a block from it. For example, consider the following plan:

1. Pickup A
2. Put A onto C (so that something is on C)
3. Pickup A (so that C is clear)
4. Put A onto B (so that the hand is empty)
5. Pickup A
6. Put A onto C (to achieve the main goal)

This plan is in fact valid, but it is apparent to the observer that it cannot possibly be the best plan, because it contains a loop: the condition A on C is achieved in support of the preconditions of the action of putting A on C. A planner can avoid generating the inefficient plan simply by finding a better plan first, namely:

1. Pickup A
2. Put A onto C (to achieve the main goal)

The order in which plans are generated depends on the accuracy of the planner's cost-estimating functions. If the top level goal is simply "get A on C", then the planner is unlikely to extend the undesirable looping plan very far. If, however, the top level goal was something like "get A on C and achieve G", where G is a subgoal the achievement of which will not interact significantly with the achievement of "get A on C", then the situation would be different because the planner would have much more of an opportunity to make erroneous cost estimates. For example, suppose that the planner initially underestimates the cost of achieving G.

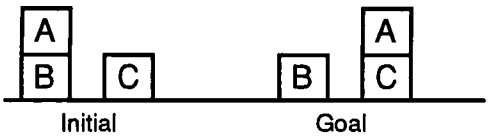

Figure 5: A blocks world problem

This will tend to cause it to expand all possible alternatives for achieving "get A on C", since as each partial plan is extended to deal with *G* it will begin to look more and more expensive. Thus the planner may generate a partial plan containing the loopy plan for getting A on C described above before its attention shifts back to the preferable alternative. The cost of considering such plans may be very high.

This loop, and others like it, can be avoided through the use of filter conditions. If "the block to be picked up is on another block" is treated as a filter condition for the pickup action, instead of an enabling precondition, the planner can never put a block on another block simply in order to take it off again. This change would prevent the planner from generating step 2 of the loopy plan described above.[8]

The use of filter conditions in avoiding loops was recognized by (Feldman & Morris, 1990). The result of their analysis is to eliminate two simple types of loops: those in which a goal is an immediate subgoal of itself, and those in which a goal repeats after two steps. However, these loops are eliminated only in certain special circumstances that apply only in very restricted domains.[9] It would not, for instance, work in any of the common representations of blocks world.

Once again we have an application of filter conditions to solve a problem that is a special case of one that the planner must solve in any case. This use of filter conditions does not address the general question of loop detection and elimination, but only rules out certain simple cases of loops. Furthermore, there does not seem to be anything especially natural about using filter conditions in this way: they are simply a device for causing the planner to skip some portions of the search space, and any such device could in principle be manipulated in such a way as to prevent some loops.

3.4 Accounting for effects

An action will in general have different effects in different contexts, and any scheme of action representation must be able to handle these context-dependent effects. In STRIPS operators, context-dependence is accounted for through the use of variables, which may appear in the preconditions, add list, and delete list of the operator. Each possible instantiation of these variables represents a different context in which the action associated with the operator might be executed. For example, consider the TIP operator in figure 6. Here the container that is to be tipped is represented by a variable (*?container*) so that instances of tipping one container can be differentiated from instances of tipping another. Since the operator's effects are stated in terms of the variables, different effects will be asserted in different contexts, allowing the planner to represent, for example, the fact that the container

Action:	(tip ?container)
Preconditions:	(container ?container) (full ?container) (holds-liquid ?container) (holding robot ?container) (above ?container ?target)
Delete list:	(full ?container) (holds-liquid ?container)
Add list:	(empty ?container) (wet ?target)

Figure 6: A STRIPS operator for TIPping a container

that is tipped is the same container that is subsequently empty.

Some of the variables in an operator are instantiated when the operator is chosen, if a particular binding of a variable is necessary in order to achieve the goal. For example, if the TIP operator is chosen to achieve the goal "Container A is empty", then clearly the variable representing the container must be bound to A. Other variables, however, may remain unbound. For example, the variable representing the place into which the container is poured (*?target*) need not be bound to any particular location initially, since any location would do to satisfy the goal. Because this variable remains unbound, the planner will have the option of binding that variable in various ways when it is searching for methods to achieve preconditions that depend upon that variable. For example, the TIP operator has the precondition "the container to be picked up is directly above a particular object", represented as:

(directly-above ?container ?target).

The planner's options in establishing this precondition are: (1) bind *?target* to whatever object the container is directly above at the start of the plan, and assume that the container will remain positioned over that object, (2) bind *?target* to an object that the container came to be above as the result of a previous action in the plan, or (3) bind *?target* to another object entirely, and schedule an operator designed to position the container over that object. Notice that the third possibility seems inefficient: since the operator will achieve its goal no matter what target is bound to, why choose a binding that necessitates an extra action?

This represents an efficiency problem similar to that described in the section on loops. While it is clear that for any plan in which an action is taken solely to position the container there is a more efficient alternative plan in which the action is not taken, the planner may nonetheless waste considerable amounts of time in considering plans that involve the less efficient choice. One way to fix this problem is to represent as filter conditions those preconditions that establish the bindings of variables not bound at selection time. For example, the condition "the container is directly above a particular object" could be made a filter condition for the TIP operator. This would mean that the planner would never move the container in order to achieve this condition. The filter condition would still fulfill the objective of discriminating the context in which the operator is being applied, by binding the variable representing the object under the container.

The argument for this approach is that, since it does not matter to the achievement of the goal which context the operator is executed in, there is no point in expending effort solely to make such a switch. Notice, however, that this reasoning applies only to the use of an operator to achieve a

[8] A similar use of filter conditions is made in the SIPE domain of (Wilkins, 1988).

[9] The example they present in their paper does not seem to meet the conditions for the analysis they propose.

particular goal. Since a STRIPS operator may be used to achieve any condition on its add list, this is problematic. For instance, in our example the TIP operator was chosen to achieve the condition "a particular container is empty", but it might equally well have been chosen to achieve the condition "a particular object is wet". In this case it would not make sense to have "the container is over a particular object" be a filter condition. In fact, it would be essential that it be an enabling precondition, since the planner should consider plans in which the container is moved in order to position it over the object that is to be made wet. In effect, such a use of filter conditions commits the planner to an assumption about the uses to which a particular operator will be put.

3.5 Information goals

One of the basic assumptions lying behind most research in classical planning is that the planner has full knowledge of the initial conditions in which the plan is to be executed. However, classical planning techniques can be extended to handle situations in which this assumption does not hold (see e.g. Etzioni, et al., 1992; Peot & Smith, 1992; Pryor & Collins, 1993; Warren, 1976). In such extensions, the resulting plans must include steps to gather information about the situation in the world at the execution time of the plan, as well as the courses of action contingent[10] on the information-gathering.

It seems reasonable to assume that satisfying an information goal such as *(know P)* where P is some state in the world should not involve changing P. To this end Etzioni et al. introduce two types of precondition that they use to specify information goals: *hands-off* and *find-out* conditions. Find-out conditions are essentially filter conditions; the state of P may not be changed in order to satisfy a subgoal of the form *(find-out P)*, but may be changed in order to achieve other subgoals. They may be satisfied by observing P at any relevant stage of the plan. Hands-off conditions are stronger, in the sense that if there is a goal of the form *(hands-off P)*, P may under no circumstances be changed by actions in the plan. Hands-off goals may only be satisfied by observation of an initial condition.

Each of these conditions will suffer from the same incompatibility with non-linear planning noted above. In particular, since actions that change the value of a piece of information may be added to the plan at any point, it will not be clear what the value of a *find-out* or *hands-off* condition will be until the plan is complete. Etzioni et al. note this problem in passing, but do not pursue the issue. We believe that it will prove to be a serious problem for their approach.

4 Conclusions

In this paper we have attempted to clarify the uses to which filter conditions have been put in various approaches to planning. We have concluded that in nearly every case the use of filter conditions is suspect for two reasons: first, because it is difficult to define a sensible interpretation of such conditions in a non-linear planner, and, second, because in using a filter condition in place of an enabling precondition the author of an operator is typically building an assumption into the system that will not be valid in all cases. We have seen, moreover, that in most cases there is reason to believe that an alternative method would achieve the aim the filter conditions were meant to achieve, without posing the same difficulties. In short, we believe that the mechanism of filter conditions is flawed, and that their use, particularly in non-linear planners, is generally a mistake.

Acknowledgements: Thanks to L. Birnbaum, M. Brand and M. Freed for many useful discussions. This work was supported in part by the AFOSR under grant number AFOSR-91-0341-DEF, and by DARPA, monitored by the ONR under contract N-00014-91-J-4092. The Institute for the Learning Sciences was established in 1989 with the support of Andersen Consulting, part of The Arthur Andersen Worldwide Organization. The Institute receives additional support from Ameritech and North West Water, Institute Partners, and from IBM.

References

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

Charniak, E., & McDermott, D. (1985). *Introduction to artificial intelligence.* Reading, MA: Addison-Wesley.

Collins, G., & Pryor, L. (1992a). Achieving the functionality of filter conditions in a partial order planner. AAAI 92.

Collins, G., & Pryor, L. (1992b). *Representation and performance in a partial order planner* ILS TR # 35.

Currie, K., & Tate, A. (1985). O-Plan—Control in the Open Planning Architecture. *Expert Systems, 85*, 225-240.

Currie, K., & Tate, A. (1991). O-Plan: the open planning architecture. *Artificial Intelligence, 52*, 49-86..

Etzioni, O., Hanks, S., Weld, D., Draper, D., Lesh, N., & Williamson, M. (1992). An approach to planning with incomplete information. KR 92.

Feldman, R., & Morris, P. (1990). Admissible criteria for loop control in planning. AAAI 1990.

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

Peot, M. A., & Smith, D. E. (1992). Conditional Nonlinear Planning. AIPS 1992.

Pryor, L., & Collins, G. (1993). Cassandra: Planning for contingencies. Submitted to IJCAI 1993.

Tate, A. (1977). Generating project networks. IJCAI 1977. Also in (Allen *et al* 1990).

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

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

[10]We prefer the term "contingency plan" to the more commonly used "conditional plan" as it avoids confusion with "conditional effects" (a.k.a. context-dependent effects).