Automatic Derivation of World Update Schemes

Joseph Downs
Department of Automatic Control and Systems Engineering
University of Sheffield
Sheffield, S1 4DU
England
E-mail: downs@acse.sheffield.ac.uk

Han Reichgelt
Subdepartment of Computer Science
Department of Mathematics
University of the West Indies
Mona, Kingston 7
Jamaica
E-Mail: han@uwimona.edu.jm

Nigel Shadbolt
Artificial Intelligence Group
Department of Psychology
University of Nottingham
Nottingham, NG7 2RD
England
E-mail: nrs@psyc.nott.ac.uk

Abstract

AI planning systems must be able to update their internal model of the domain in order to determine the effects of a possible action. The most widely used update method is the STRIPS operator. STRIPS operators are extremely efficient. However, the use of STRIPS operators requires that one specify all the consequences of an action beforehand. This becomes almost impossible for complex domains. Work by Pednault (1989) and Wilkins (1988) overcomes the problems associated with ordinary STRIPS operators by enhancing them with context-dependent effects and demons respectively. In this paper we describe a modelling procedure which can incorporate both these features. Moreover, the update method is automatically derived from an axiomatisation of the domain in temporal calculus. This allows the system designer great flexibility in choosing the trade-off point between representational expressivity against computational speed.

1. Introduction

A requirement of AI planning systems is that they are able to update their internal model of the domain so as to simulate the effects of action execution in that domain. The most commonly used means of achieving this is by STRIPS operators (Fikes and Nilsson 1971). In their most basic form, such operators represent the consequences of actions by the specification of add and delete lists for those literals in the world model which are affected by the operator.

The advantage of STRIPS operators lies in the solution they offer to the frame problem (McCarthy and Hayes 1969), the problem of determining which literals are not affected by an action, and therefore keep the same truth value after the execution of an action. Under the STRIPS approach, only those literals in the world model which are explicitly mentioned in the operator's add or delete list will change. Moreover, STRIPS operators are efficient: updating the world model is more a matter of pattern-matching than complex inference.

One limitation however is that STRIPS operators (in their original formulation) do not allow context-dependent effects, and are thus subject to proliferation of operators (Pednault 1989). For example, in the blocksworld domain, putting a block onto the table differs slightly in its effects from putting a block onto another block. In the former case the table remains clear, whereas in the latter the supporting block ceases to be clear. In STRIPS this difference would need to be modelled by means of two separate operators, *put-on-table* and *put-on-block*. The ADL system (Pednault 1989) overcomes this problem by allowing conditional add and delete lists.

A further drawback of using STRIPS operators lies in the fact that the add and delete lists associated with an operator must explicitly state all the consequences of executing the associated action (Ginsberg and Smith 1987). However, as domain complexity increases, it becomes increasingly difficult for the designer of a

planning system to do this. Even when context-dependent STRIPS operators are used, the designer still has to list all the consequences of every action in all possible situations.

This problem is overcome in the SIPE system (Wilkins 1988) by supplementing ordinary (i.e. not context-dependent) STRIPS operators with two types of demons (termed domain-rules). All context-dependent changes are encoded within these demons, which are applied after the STRIPS operator. One type, state rules, enforce constraints within an individual world state; the other, causal rules, concern changes between world states. (Causal effects are always applied before state effects.) Thus it can be seen that one demon in SIPE can potentially encode for effects caused by many operators.

In this paper we describe a modelling procedure which can potentially incorporate both context-dependent STRIPS operators and demons. The update method is derived from an axiomatic description of the domain in temporal calculus. The intention is to allow the designer of the planning system flexibility in deciding the trade-off point between representational expressivity and computational efficiency. For example, an action-centred domain description will result in an efficient but inexpressive update procedure, utilising only STRIPS operators. Moreover it will involve increasing specification difficulty as domain complexity increases. A more object-centred axiomatisation will be easier to specify, and result in a less efficient but more flexible update procedure, involving context-dependent STRIPS operators and/or demons. This point is elaborated upon in later sections.

The outline of the remainder of this paper is as follows. Section 2 describes the modelling procedure in detail. Section 3 compares this procedure with those used by Pednault and Wilkins. Section 4 discusses some limitations of the procedure. Section 5 concludes.

2. Updating the World Model

The execution of actions is simulated by a two-stage method. First, the world model is updated by the application of the STRIPS operator appropriate to the action. (This STRIPS operator may or may not be context-dependent, according to how it was derived from the domain axiomatisation.) However, the world model that results from this first step may only give a partial description of the effects of the action. In this case, further updating of the intermediate world model is performed by application of a general set of demons for the domain. (Again, these demons are derived from the original domain axiomatisation in temporal calculus. If no such demons were derived, the final world model is that which results from application of the STRIPS operator alone.) Any demons whose preconditions match to the world model are "fired" and add their single consequent literal to the world model. The world model that results once all demons reach quiescence (i.e. all demons which can fire have done so), is the new world state that should result from execution of the action in the initial world state. We will now describe these stages in more detail.

2.1. STRIPS operators

STRIPS operator derivation occurs in the following manner: all domain description axioms are first re-written into a normal form of first order logic where each axiom is an implication whose consequent is a single literal and whose antecedent is a conjunction of literals. The resulting axioms are then classified into 3 distinct sets: *universal effects*, *conditional effects* and *domain constraints*. Both universal and conditional effect axioms are characterised by the consequent being true over a temporal interval subsequent to that for the antecedent. Domain constraints are characterised by the consequent being true over the same interval as the antecedent.

Universal effect axioms are distinguished by a single action literal in their antecedent, all of whose arguments are universally quantified. Thus, using reified temporal logic, the following are examples of universal effect axioms:

$(\forall x,y,t)(occurs(pick\text{-}up(x,y), t \rightarrow true(held(x), t+1))$

$(\forall x,y,t)(occurs(pick\text{-}up(x,y), t) \rightarrow true(\neg clear(x), t+1))$

Conditional effect axioms also contain an action literal in their antecedent, but either its arguments are not all universally quantified, or the antecedent contains further literals, or both. Thus, the following is an example of a conditional effect axiom:

$(\forall x,t)((occurs(pick\text{-}up(x,y), t) \wedge \neg(y=table)) \rightarrow true(clear(y), t+1))$

All other axioms are classified as constraints, e.g.

$(\forall x,t)(holds(held(x), t) \rightarrow true(\neg free(hand), t))$

The total set of universal effects for each action is generated next. This can be defined as those effects of an operator that always occur regardless of which particular world-state holds when the operator is applied. Initially the universal effects of an operator are gathered directly from the consequents of the relevant effect axioms. The resultant set is then supplemented by exhaustively forward-chaining on these effects using the constraint axioms, with the proviso that we only match universally quantified variables. Attempts to match all other terms fail. When a constraint is successfully matched its consequent is added to the list of effects.

Thus, assuming the list of universal effect axioms for the *pick-up* operator plus the domain constraint given above, then the initial universal effects set for this operator would be { $held(x)$, $\neg clear(x)$ }, which after the second step would grow to { $held(x)$, $\neg clear(x)$, $\neg free(hand)$ }.

Following generation of the universal effects for an operator, a somewhat similar process occurs to generate the conditional effects of that operator. Whereas each operator may have only one set of universal effects, it may have no, one or many different conditional effects. A single conditional effect is a precondition-effects pair such that each effect is only asserted (i.e. added or deleted) when the operator is applied in a world state in which all the associated preconditions prove true. Initially, conditional effects for an operator are gathered from the conditional effect axioms. For example, from the above conditional effect axioms for *pick-up*, we form the initial conditional effect for the operator:

$< \neg(y = table), clear(y) >$

We then associate with each operator an *effects-tree*. Within this tree, those preconditions which are subconditions of a more general precondition are made child nodes of this less specific precondition. In such child nodes only the extra preconditions need to be stored (i.e. those preconditions not present in any ancestor nodes). This optimises matching costs during later traversal of the effects-tree in order to update the world model. The universal effects of an operator form the (implicit) root node of this tree as a zero-precondition node. Thus, from the *pick-up*(x,y) effect axioms given earlier, we get the following effects tree:

pick-up(x,y)
  conditions: nil
  effects: $held(x)$, $\neg clear(x)$, $\neg free(hand)$
    conditions: $\neg(y = table)$
    effects: $clear(y)$

As before conditional effects are then supplemented by forward-chaining against constraints. However, matching occurs in a breadth-first manner down the tree[1]. This ensures that new effect literals from the constraints are added at the most general possible level of precondition. For matching purposes each node inherits the effects of all its ancestor nodes.

Finally, for each of the effects in any level of the tree, we create add and delete lists in the expected fashion: each atomic proposition under a negation operator goes into the delete list, each positive literal goes into the add list. Thus, the final effects tree for *pick-up*(x,y) becomes:

pick-up(x,y)
  conditions: nil
  effects
    add list: $held(x)$
    delete list: $clear(x)$, $free(hand)$
      conditions: $\neg(y = table)$
      effects
        add list: $clear(y)$
        delete list: nil

Run-time application of the STRIPS operators generated as described above consists of breadth-first descent of the effects tree, asserting those effects in each node into the new world model provided the preconditions prove true in the initial world state. Once a precondition node fails to match to the initial world state that branch is no longer descended.

It can be seen that a domain axiomatisation which produces no conditional effect axioms for an operator results in the derivation of ordinary STRIPS operators. Where conditional effect operators are produced, context-dependent STRIPS operators result. However, in either of these cases, the world state produced by STRIPS operator application does not always give a full description of the state that should result from execution of the equivalent action. The world model that results from STRIPS operator

[1] The case of multiple inheritance, resulting in an effects graph, is avoided. Section 4 discusses this further.

application may be further updated by a demon-firing phase, described in the next section.

2.2. Demons

Constraints that were not used during STRIPS operator derivation (generally those which contained existentially quantified variables) are used to generate demons. A demon in SITACT is a disjunct of conjuncts of triggering patterns. (Obviously, the disjunction is originally formed by grouping together all domain constraints that have the same effect literal.) When any such conjunct proves true in a world state, a single effects literal is asserted, else if all conjuncts prove false the negation of the effect is asserted. (This is of course negation-as-failure.)

Thus, from the constraints:

$(\exists x,t)(holds(on(x,table), t) \rightarrow$
$true\ (obscured(picture), t))$

$(\exists x,t)(holds(on(x,chair), t) \rightarrow$
$true\ (obscured(picture), t))$

we generate the demon
triggers: $on(x,table) \vee on(x,chair)$
effect: $obscured(picture)$

As a result $obscured(picture)$ will be added to the world model as long as there is something on the table or the chair, otherwise it will be removed. (Providing of course that the literal can be removed, it might not present originally anyway since we use negation-as-failure.)

The triggering of an individual demon during the update of a world model is naturally optimised in the standard way. As soon as one triggering pattern proves true, no further patterns are tested. Once a demon is triggered, it is (temporarily) removed from the test set. This ensures that no demon can be applied more than once during the update process for a particular action.

After application of a STRIPS operator to the world model, the resultant intermediate world state is matched against all such demons. Matching continues until demon-firing reaches quiescence, when all untriggered demons have the negation of their effect literal asserted (if possible). The model which now results gives the final world state, i.e. the state which results from simulated execution of an action. At this point all temporarily removed demons are returned to the test set for the next action's effects to be calculated.

The use of demons increases computational costs in comparison to the use of STRIPS operators in isolation. However, the approach is still based more upon pattern-matching than theorem-proving and should not prove intolerably high, provided the demon set is not too large. Furthermore, the use of demons allows the designer much greater flexibility in describing the domain. In particular the designer is not forced to give the purely action-centred formulation that is necessary when STRIPS operators (context-dependent or not) are used alone.

As an example of this increased representational power, consider the axiom[2]:

$(\exists x)\ (on(x,table)) \leftrightarrow obscured(picture)$

This expresses the intuitive notion that if there is anything on the table the picture will be obscured. It can be regarded as an object-centred description, which could be easily encompassed by a demon. It would however be very difficult to express this in the action-centred representation needed by STRIPS operators. One might specify axioms which state that any object being moved to the table causes the picture to become obscured, and any object being moved from the table causes it to become unobscured:

$(\forall x,y,t)(occurs(move(x,y,table), t) \rightarrow$
$true(obscured(picture), t + 1))$

$(\forall x,z,t)(occurs(move(x,table,z), t) \rightarrow$
$true(\neg obscured(picture), t + 1))$

However, this is only valid if the table location can only ever hold a maximum of one object at any time. If the table's capacity is larger, the axioms do not hold. A correct set of action-centred axioms becomes much harder to specify since one must explicitly keep track of the number of objects on the table at all times.

3. Related Work

The update method described in the previous section can be seen to have close links with the context-dependent operators used in ADL (Pednault 1989) and the demons used in SIPE (Wilkins 1988). A comparison with these two systems is revealing.

---

[2] We borrow this from Ginsberg and Smith (1987).

ADL is intended as a hybrid of STRIPS operators and the situation calculus. Syntactically speaking, the representation can be viewed as STRIPS operators containing conditional add and delete lists. ADL's lack of demons makes it less expressive than our update procedure. It must use a purely action-centred description at all times. In fairness however it should be noted that ADL is superior in allowing for partially-specified initial world states, a feature our method does not possess. Although it should also be noted that Pednault advises against such partial specifications wherever possible, since ADL must resort to expensive theorem-proving in such cases.

In addition, ADL does not allow for the nesting of conditions that occurs within our effects tree. Matching costs in ADL will therefore be greater. Nor does it supplement operator effects from constraints, which in our approach eases the encoding process in cases where different operators partly share the same effects.

In contrast SIPE provides an ordinary (i.e. not context-dependent) STRIPS operator notation supplemented by two types of domain rules or demons (which have equal expressivity). Thus SIPE encodes only universal effects within its STRIPS operators, and calculates all context-dependent changes from the domain rules.

The domain rules in SIPE are more sophisticated than the demons in our modelling procedure. They provide greater expressivity in that they can match to the previous world state as well as the current world state. Moreover, matching costs are reduced by means of filters (called triggers) on rules, which test the basic applicability of a rule. If the filter is not passed, no further matching is performed. SIPE gains further efficiency by only applying rules which match to new state changes, whereas our demons are not optimised.

However, our demons are not intended to bear as great a representational burden as SIPE's domain rules must. This is because we utilise context-dependent STRIPS operators, and so much more of the updating of the world model can occur outside of the demons.

It may appear at first sight that our update procedure is intermediate between that of ADL and SIPE. ADL has context-dependent STRIPS operators but no demons; whereas SIPE has more sophisticated demons, but context-independent STRIPS operators. However, we would claim that our method scores over both of these systems in offering greater flexibility for selecting where to trade representational expressivity for computational speed.

While our use of context-dependent STRIPS operators and demons is not original, the automatic derivation of them from axiomatic descriptions seems to be. This allows one to explicitly adjust the trade-off point.

Thus, if computational cost is the over-arching factor, and the designer is willing to painstakingly provide an entirely action-centred domain description, then the resultant update procedure will be maximally efficient. No demons will be used, only STRIPS operators. Such operators can (and probably will) be context-dependent, but ordinary STRIPS operators are also possible.

However, as domain complexity grows (and designer patience decreases !), a less action-centred formulation can be adopted. This will of course pay a price in that less efficient update procedures which incorporate demons may be produced, but the designer is free to reformulate the domain more efficiently if this cost proves too high.

4. Limitations

One flaw in our update procedure in comparison to that of SIPE is the lack of optimisation in the matching of demons. It should be reasonably easy to set up pointers between STRIPS operator effects and the preconditions of demons, enabling only those demons which are relevant to a particular operator to be matched, as occurs with Wilkins' domain rules. The implementation of filters on demons is more problematic. These are hand-coded in SIPE based on the designer's knowledge of the domain. Our temporal calculus notation does not naturally support the encoding of preferential matching on particular preconditions.

From a logician's viewpoint another problem is that we treat the implication operator as meaning "causes" in compiling the update procedure. Since our theorem-prover uses the standard logical interpretation of the operator, the initial normalisation of the domain description can yield what we term "correlational" rather than "causal" axioms. This is because logical implication is a

wider operation than causation[3]. For example, $A \leftrightarrow (B \wedge C)$ yields (among others) $A \rightarrow B$, $A \rightarrow C$, and $B \rightarrow C$ after normalisation. The first two of these expressions describe the actual causality of the domain, while the third is merely incidental. Such expressions need be pruned before construction of the effects trees for STRIPS operators since they would result in unwanted multiple inheritance for conditional effects. In choosing which candidates to prune, we always opt to retain expressions from the designer's original formulation of the domain. In reply to the inevitable objections from logicians, we stress that we do not intend to support the full expressivity of temporal calculus in our update procedure. Rather we use temporal calculus merely as a convenient notation for describing operators and simple domain constraints.

5. Summary

We have described an update procedure which can be adjusted in terms of the expressivity-tractability trade-off. The designer is left to choose where they want to position their system. They will weigh initial specification cost against subsequent computational efficiency. The approach allows for revisions to be easily made in the light of experience. Automatic derivation of the update procedure from an axiomatic description is likely to be very attractive to system developers.

Acknowledgement

The first author was financially supported by a SERC postgraduate studentship.

References

Allen, J.F. 1984. Towards a General Theory of Action and Time. *Artificial Intelligence* 23(2): 123-154.

Fikes, R. and Nilsson, N. 1971. STRIPS: A New Approach to the Application of Theorem-Proving to Problem Solving. *Artificial Intelligence* 2(3-4):189-208.

Ginsberg, M. and Smith, D. 1987. Reasoning about Action I: A Possible Worlds Approach. In Proceedings of the 1987 Workshop on the Frame Problem in Artificial Intelligence, 233-258. San Mateo, California: Morgan Kaufmann.

Levesque, H.J. and Brachman, R.J. 1985. A Fundamental Tradeoff in Knowledge Representation and Reasoning. In R.J. Brachman and H.J. Levesque (eds) *Readings in Knowledge Representation*, 41-70. San Mateo, California: Morgan Kaufmann.

McCarthy, J. and Hayes, P. 1969. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie (eds) *Machine Intelligence 4*, 463-502. Edinburgh: Edinburgh University Press.

Pednault, E.P.D. 1989. ADL: Exploring the Middle Ground between STRIPS and the Situation Calculus. In Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR-89), 324-332. San Mateo, California: Morgan Kaufmann.

Wilkins, D.E. 1988. Causal Reasoning in Planning. *Computational Intelligence* 4(4):373-380.

[3] And it can be argued that "causes" is simply an extra-logical notion similar to defaults.