Inner and Outer Boundaries of Literals A Mechanism for Computing Domain Specific Information

Fahiem Bacchus
Dept. Of Computer Science
University of Toronto
Toronto, Ontario
Canada, M5S 3G4
fbacchus@cs.toronto.edu

Cameron Bruce Fraser
Dept. of Computer Science
University of Waterloo
Waterloo, Ontario
Canada, N2L 3G1
cbfraser@logos.uwaterloo.ca

March 10, 2000

1 Introduction

A number of works have shown that planning can be speeded up, often very significantly, by utilizing extra domain knowledge [KM81, BK00, BK96, KS98, SK98, DK99, Rei99, NCLMA99, vBC99]. The question that immediately arises is "where does this extra information come from?"

We have gained considerable experience with utilizing extra domain information in a planner through implementing numerous planning domains in the TLPLAN system [BK00]. The TLPLAN system is a planning system specifically constructed to utilize extra domain knowledge. In fact, the underlying planning algorithm is very simple—a simple forward chaining engine. Hence almost all of TLPLAN's performance is due to the extra domain knowledge it utilizes.

We have noticed that a number of systematic principles can be applied to uncover extra domain knowledge, and that once one has developed an intuitive feel for these principles it becomes quite easy to identify and write down useful extra domain knowledge.

Some of these principles seem to involve proofs by induction, at which people can be quite accomplished but which seem to be difficult to automate. Nevertheless, other principles seem to involve much simpler reasoning.

In this paper we present an idea that we are working on that appears to have considerable potential for generating non-inductive domain knowledge. We formalize the idea and show how it can be used in various ways to compute extra domain knowledge.

2 Inner and Outer Boundaries of Literals

We start with a specification of the actions available in a domain, and the assumption that these actions are the only mechanisms for changing a state. We want to compute extra information from the specification of the actions (and the fact that the action specification is complete) that can be used to speedup planning.

This problem has been examined before. There have been typically two different kinds of approaches. Approaches based on learning, e.g., [Min88, Kha97], and approaches based on reasoning, e.g., [Etz93, PS93, GS96, FL98, GS98]. The latter approaches have typically employed the solution to the frame problem that is implicit in the STRIPS representation of actions: fluents only change when they appear on the add or delete list of an action. Using this property these works have typically used graph constructions to compute state invariants, operator reachability, etc. Our approach is based on similar ideas, and has been strongly influenced by this previous work.

We introduce the notion of inner and outer boundaries of literals. This notion is implicit in many of the graph based constructions of previous works. A key contribution is to make this notion explicit. In this way we can formally characterize the notion, develop algorithms that *soundly* compute boundary formulas,[1] and gain deeper insights into how boundary information can be used.

We start with a literal ℓ (assume for now that the literal is ground, i.e., it contains no variables). ℓ characterizes a set of states s: $\{s : s \models \ell\}$, those states in which ℓ holds.

Now consider those states s' that do not satisfy ℓ, but at which there is an action a that can transform s' to a state s in which ℓ does hold: the states s' are one action "away from ℓ". This process can be continued, and it is clear that there may be states that are two, three, or more actions, away from a state satisfying ℓ.

We do not want to deal with just ground literals however, as in general ground literals are only available on a problem by problem basis as we specify the initial and goal states. We want to do our reasoning once and for all using just the domain actions, in this way we can simply instantiate our computed knowledge at plan time to deal with the initial and goal states of the particular problem. To this end we make the following definitions:

Definition 2.1 Given a literal ℓ containing some free variables and an (any) instantiation σ for these variables, we define the *outer boundaries* of ℓ to the *formulas* characterized by the following sets:

$$
\begin{aligned}
s \models \mathcal{O}_0[\ell] \quad &\Leftrightarrow \quad s \in \{s : (s,\sigma) \models \ell\} \\
s \models \mathcal{O}_1[\ell] \quad &\Leftrightarrow \quad s \in \{s : (s,\sigma) \models \neg\mathcal{O}_0[\ell] \wedge \exists a_1, s'.\mathit{Poss}(a_1, s) \wedge s' = a_1(s) \wedge (s',\sigma) \models \mathcal{O}_0[\ell] \\
&\vdots \\
s \models \mathcal{O}_i[\ell] \quad &\Leftrightarrow \quad s \in \{s : (s,\sigma) \models \bigwedge_{j=0}^{i-1} \neg\mathcal{O}_j[\ell, s'] \wedge \exists a_i, s'.\mathit{Poss}(a_i, s) \wedge s' = a_i(s) \wedge (s',\sigma) \models \mathcal{O}_{i-1}[\ell]
\end{aligned}
$$

[1] For example, in previous work some of the computations done on the graph's constructed produced only approximations to what we define formally here.

Here, *Poss*(a, s) means that the preconditions of action a hold in state s, and $a(s)$ is the state that is the result of applying a to s.

In other words, the i-th outer boundary of ℓ, $\mathcal{O}_i[\ell]$ is a formula that is satisfied by a set of states that are precisely i-actions away from a state satisfying ℓ. The formula $\mathcal{O}_i[\ell]$ is parameterized by the free variables in ℓ, for every instantiation of these variables, it will characterize a particular set of states.

Every literal ℓ also has an *inner boundary*. This is the set of situations satisfying ℓ that also have the property that they can be reached by a single action from a situation that falsifies ℓ. They can be characterized by the set

$$\{s : s \models \ell \wedge \exists a, s^-.\textit{Poss}(a, s^-) \wedge s^- \models \neg\ell \wedge s = a(s^-)\}.$$

Note that once we have achieved ℓ it might be possible to execute other actions that do not affect ℓ and thus reach states that remain in $\mathcal{O}_0[\ell]$ but leave ℓ's inner boundary. (That is, these actions might move us to states that have no incoming action from a state not already satisfying ℓ.)

We can characterize inner boundaries in terms of outer boundaries. In particular, the above inner boundary is also characterized by the set:

$$\{s : s \models \mathcal{O}_0[\ell] \wedge \textit{Poss}(a, s^-) \wedge s^- \models \mathcal{O}_1[\ell] \wedge s = a(s^-).\}$$

Here we have simply used the observation that since s^- is one action away from a state satisfying ℓ, and yet does not satisfy it itself, it must be in $\mathcal{O}_1[\ell]$.

Like the outer boundaries we can generalize this to define i-th inner boundaries

Definition 2.2 Given a literal ℓ containing some free variables and an (any) instantiation σ for these variables, we define the *inner boundaries* of ℓ as the formulas characterized by the following sets:

$$\begin{aligned}
s \models \mathcal{I}_0[\ell] &\Leftrightarrow s \in \{s : (s, \sigma) \models \mathcal{O}_0[\ell] \wedge \exists a, s^-.\textit{Poss}(a, s^-) \wedge s = a(s^-) \wedge s^- \models \mathcal{O}_1[\phi]\} \\
&\vdots \\
s \models \mathcal{I}_i[\ell] &\Leftrightarrow s \in \{s : (s, \sigma) \models \mathcal{O}_i[\ell] \wedge \exists a, s^-.\textit{Poss}(a, s^-) \wedge s = a(s^-) \wedge s^- \models \mathcal{O}_{i-1}[\ell]\}
\end{aligned}$$

The inner boundaries of ℓ specify conditions that are implied by the fact that we have just made a transition into an outer boundary.

2.1 Complexity

It is immediately apparent that computing these boundary formulas is in general intractable. One need only add a new literal G, a new dummy action that creates G and has as preconditions all of the goal atoms, and then the i-th outer boundary will characterize all initial states from which an optimal plan has length i. Nevertheless, we have found that for many literals in many domains it is in fact practical to compute these boundary sets.

3 Domain Knowledge from Boundary Formulas

Say that we are faced with a planning problem with initial state $\mathcal{I}$, and a goal containing the literal ℓ. Furthermore, say that $\mathcal{I} \models \mathcal{O}_i[\ell]$ (since the boundary formulas are mutually exclusive $\mathcal{I}$ there is only one value of i for which this is true).[2] Now we know that any plan must eventually visit states characterized by the formulas $\mathcal{O}_j[\ell]$, $0 \leq j \leq i$, and $\mathcal{I}_j[\ell]$, $0 \leq j < i$. A similar observation can be made for every other literal in the goal state.

Note that a plan need not make non-decreasing progress towards any literal, as two literals might interfere. In such cases it might be necessary to move further away from one literal in order to achieve another literal. Nevertheless, eventually a correct plan must pass through every boundary on its way to achieving each literal. It is this fact that can be exploited to generate useful extra domain knowledge.

To demonstrate how this can be accomplished we utilize a very simple example: a modified version of the briefcase domain. The actions in this domain are:

```
(def-operator (move-briefcase ?l1 ?l2)
  (pre (and (at briefcase ?l1)
            (location ?l2)
            (not (= ?l1 ?l2))))
  (add  (at briefcase ?l2))
  (del  (at briefcase ?l1)))

(def-operator (put-in ?x)
  (pre (and (at briefcase ?loc)
       (at ?x ?loc)
       (not (= briefcase ?x))
       (not (in-briefcase ?x))))
  (add (in-briefcase ?x))
  (del (at ?x ?loc)))

(def-operator (take-out ?x ?loc)
  (pre (and (in-briefcase ?x)
            (at briefcase ?loc))
  (add (at ?x ?loc))
  (del (in-briefcase ?x))))
```

This domain is like the logistics world with a single truck (the briefcase) and no airplanes. Notice that unlike the original briefcase world `at` and `in` for objects have been made mutually exclusive. (This allows us to eliminate the quantified conditional effect required to move all objects in the briefcase).[3]

The outer boundaries for each of the positive literals are:

- $\mathcal{O}_1[\texttt{(location ?l)}] \equiv$ FALSE.
- `(at ?x ?l)` (`?x` $\neq$ `briefcase`)
  1. $\mathcal{O}_1 \equiv$ `(in-briefcase ?x)` $\wedge$ `(at briefcase ?l)`
  2. $\mathcal{O}_2 \equiv$ `(location ?l)` $\wedge \exists$`?li.` `?li` $\neq$ `?l` $\wedge$ `(at briefcase ?li)` $\wedge$ `(in-briefcase ?x)`

[2] In standard planning problems $\mathcal{I}$ is complete, i.e., it contains an explicit list of all of the atomic formulas it satisfies. This means that it is easy to test for any formula ϕ whether or not $\mathcal{I} \models \phi$, and we can determine the i such that $\mathcal{I} \models \mathcal{O}_i[\ell]$.

[3] Although the conceptual notions are the same, our current computational techniques only work with STRIPS style actions.

3. $\mathcal{O}_3 \equiv$ (location ?l)
   $\wedge \exists$?li. ?li $\neq$?l $\wedge$ (at briefcase ?li)
   $\wedge$ (at ?x ?li) $\wedge \neg$(in-briefcase ?x)
4. $\mathcal{O}_4 \equiv$ (location ?l)
   $\wedge \exists$?li, ?lj. (location ?li) $\wedge$?li $\neq$?l $\wedge$?lj $\neq$?li
   $\wedge$ (at briefcase ?lj) $\wedge$ (at ?x ?li) $\wedge \neg$(in-briefcase ?x)
5. $\mathcal{O}_5 \equiv$ FALSE

- (at briefcase ?l)
  1. $\mathcal{O}_1 \equiv$ (location ?l) $\wedge \exists$?li. ?li $\neq$?l $\wedge$ (at briefcase ?li)
  2. $\mathcal{O}_2 \equiv$ FALSE
- (in-briefcase ?x)
  1. $\mathcal{O}_1 \equiv$ $\exists$?li. (at briefcase ?li) $\wedge$ (at ?x ?li) $\wedge$?x $\neq$ briefcase
     $\wedge \neg$(in-briefcase ?x)
  2. $\mathcal{O}_2 \equiv$ $\exists$?li, ?lj. (location ?li) $\wedge$?li $\neq$?lj $\wedge$ (at briefcase ?lj)
     $\wedge$ (at ?x ?li) $\wedge$?x $\neq$ briefcase $\wedge \neg$(in-briefcase ?x)
  3. $\mathcal{O}_3 \equiv$ FALSE

It should be noted that these boundary formulas have been computed automatically by our implementation.

Below we give some examples of items of domain knowledge that can be computed from these boundary formulas. Some of these items can already be handled by previous techniques. However, what is interesting here is that all of these distinct pieces of information can be generated from the same notion: simple reasoning about boundary crossings.

1. (location ?l) is a static predicate. It cannot be achieved by any action: there are no states in its outer boundary.

2. The briefcase can only be moved to locations. In particular, any instantiation of (at briefcase ?l) that does not hold in the initial state can only be achieved if we also have (location ?l) in the initial state. This follows from the fact that (location ?l) is in the first outer boundary of (at briefcase ?l).

3. For any ?x other than the briefcase, (at ?x ?l) can only be achieved if (a) the instance is true in the initial state, (b) ?x is in the briefcase and the briefcase is initially at ?l, or (c) ?l is a location. This follows from the first and second outer boundaries of (at ?x ?l).

4. In this domain every action has an inverse (given that every object starts off at a `location`). Let s be any state in $\mathcal{O}_i[\ell]$ for some ground literal ℓ. Then for any action a executed in s to produce s' we must have one of (1) $s' \models \mathcal{O}_i[\ell]$, (2) $s' \models \mathcal{O}_{i+1}[\ell]$, or (3) $s' \models \mathcal{O}_{i-1}[\ell]$. That, is every action can only affect the boundary by at most one. That we can only move forward to $i-1$ is immediate from our definition that states in $\mathcal{O}_i[\ell]$ require a minimum of i actions to achieve ℓ. That we can only move backwards to $\mathcal{O}_{i+1}[\ell]$ follows from the fact that a has an inverse that must reachieve $\mathcal{O}_i[\ell]$, thus the state it moves to must be in $\mathcal{O}_{i+1}[\ell]$.

   It is not difficult to categorize for every set $\mathcal{O}_i[\ell]$ all actions into cases (1), (2) and (3) above (our implementation already does this). In fact, a static (i.e., pre-plan time) table can be build containing this information. This suggests the following search heuristic (related to the approach taken by Bonet et at. [BLG97]).

   (a) For every goal literal ℓ, compute the i such that the initial state satisfies $\mathcal{O}_i[\ell]$. Store these numbers in a distance array indexed by the goal literals.

   (b) When an action is taken use the pre-computed table to determine for each goal literal whether or not we have moved forward, backwards, or stayed the same. Update the distance array by -1, 1, 0 accordingly.

   (c) Use the sum of the distances as a heuristic.

   Of course, this strategy will only work in invertible domains that also have the property that the boundary sets are easily computable. Nevertheless, like the recent regressive approach utilized by Bonet and Geffner, it has the advantage of not requiring a continual recomputation of the heuristic.

5. In some domains certain actions will make it impossible to subsequently cross particular boundaries. For example, an action might destroy a condition that can never be created again. If this condition appears in a boundary of a literal, we will not be able to subsequently cross that boundary.

   In the schedule domain, e.g., some actions make an object hot and there is no action for subsequently making it cool again. That the object is cool appears in the first boundary of achieving a particular colour for an object (via a paint action). That is, the object must be cool for it to be painted.

   It is possible to detect these kinds of situations by examining the boundary formulas, and thus one can construct a control rule that prohibits the planner from rendering a boundary uncrossable until it has in fact crossed it, or from moving back across a boundary if that boundary is now uncrossable.

   For example, such an analysis would force the planner to paint an object before allowing it to become hot, and would prohibit the planner from destroying the object's colour if it the object was hot.

4 Current Status

The previous section outlined some of the information that one arrives at by thinking about boundaries. We currently are working on an implementation that is able to compute inner and outer boundary sets given a specification of the domain's operators. The briefcase example given in the previous section was generated by our current implementation. The implementation is also able to compute the boundaries for the rocket domain, and the first set of boundaries for the logistics domain.

Once we have a reasonable mechanism for computing boundaries, we intend to investigate the automation of some of reasoning contained in the examples given above. We are also interested in extending our implementation to work with richer operator descriptions, as it is currently limited to STRIPS operators.

References

[BK96] Fahiem Bacchus and Froduald Kabanza. Using temporal logic to control search in a forward chaining planner. In M. Ghallab and A. Milani, editors, *New Directions in AI Planning*, pages 141–153. ISO Press, Amsterdam, 1996.

[BK00] Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116, 2000.

[BLG97] B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of the AAAI National Conference*, pages 714–719, 1997.

[DK99] P. Doherty and J. Kvarnstrom. Talplanner: An empirical investigation of a temporal logic-based forward chaining planner. In *Proceedings of the 6th Int'l Workshop on the Temporal Representation and Reasoning (TIME'99)*, 1999.

[Etz93] Oren Etzioni. Acquiring search-control knowledge via static analysis. *Artificial Intelligence*, 62(2):255–302, 1993.

[FL98] M. Fox and D. Long. The automatic inference of state invariants in tim. *JAIR*, 9:367–421, 1998.

[GS96] A. Gerevini and L. Schubert. Accelerating partial-order planners: Some techniques for effective search control and pruning. *Journal of Artificial Intelligence Research*, 5:95–137, 1996.

[GS98] A. Gervini and L. K. Schubert. Inferring state constraints for domain-independent planning. In *Proceedings of the AAAI National Conference*, pages 905–912, 1998.

[Kha97] Roni Khardon. Learning action strategies for planning domains. Technical Report TR-10-97, Harvard University, 1997. Available at http://www.dcs.ed.ac.uk/home/roni/pubabs.html.

[KM81] D. Kibler and P. Morris. Don't be stupid. In *Procceedings of the International Joint Conference on Artifical Intelligence (IJCAI)*, pages 345–347, 1981.

[KS98] Henry Kautz and Bart Selman. The role of domain-specific knowledge in the planning as satisfiability framework. In *Proceedings of the International Conference on Artificial Intelligence Planning*, pages 181–189, 1998.

[Min88] Steve Minton. *Learning Search Control Knowledge*. Kluwer Academic Publishers, 1988.

[NCLMA99] Dana Nau, Yue Cao, Amnon Lotem, and Hector Munoz-Avila. Shop: Simple hierarchical ordered planner. In *Procceedings of the International Joint Conference on Artifical Intelligence (IJCAI)*, pages 968–973, 1999.

[PS93] M. Poet and D.E. Smith. Threat-removal strategies for partial-order planning. In *Proceedings of the AAAI National Conference*, pages 492–499, 1993.

[Rei99] Ray Reiter. *Knowledge In Action: Logical Foundations for Describing and Implementing Dynamical Systems*. 1999. Unpublished draft, available at http://www.cs.utoronto.ca/˜cogrobo/.

[SK98] B. Srivastava and S. Kambhampati. Synthesizing customized planners from specifications. *Journal of Artificial Intelligence Research*, 8:93–128, 1998.

[vBC99] Peter van Beek and Xinguang Chen. Cplan: A constraint programming approach to planning. In *Proceedings of the AAAI National Conference*, pages 585–590, 1999.