Precondition Control*

Fahiem Bacchus
Dept. Of Computer Science
University Of Waterloo
Waterloo, Ontario
Canada, N2L 3G1
fbacchus@logos.uwaterloo.ca

Michael Ady
Winter City Software
Edmonton, Alberta
Canada, T5R 2M2
winter.city@v-wave.com

Abstract

Exploiting domain specific information is a promising and perhaps essential means for improving the performance of AI planning systems. A number of important issues arise when investigating the use of such information. Among the most important concern the questions of how domain specific information is to be represented and how planning algorithms can be created or modified to take advantage of such information. In this paper we explore the simple mechanism of encoding such information as extra operator preconditions. We examine what is required of the operator description language to make this feasible and provide some empirical results on the effectiveness of this approach. We find that precondition control requires a very expressive operator description language, but that when it can be applied it can yield speedups that are orders of magnitude better than other methods of encoding domain specific information.

1 Introduction

Although much of the research in planning has concentrated on domain independent planning algorithms and planning systems, it is acknowledged that exploiting domain specific information has a key role to play in achieving improved performance. In fact most planning formalisms are sufficiently general to be intractable in the worst case [ENS92], so it can be argued that finding ways to exploit domain specific information is essential if we ever hope to build planning systems efficient enough to solve real problems.

There has been some work on utilizing domain specific information in planning systems. A significant branch of planning research has concentrated on HTN planners (e.g., [Wil88]). Such planners encode much more domain specific information that do classical planners (i.e., those based on STRIPS or ADL operator representations). In particular, configuring an HTN planner for a particular domain involves uncovering a plausible collection of task decompositions suitable for solving problems in that domain. The hierarchical structure of the task network embeds considerable domain specific knowledge that the planner can utilize. Work on creating mechanisms for adding domain specific knowledge to classical planners has also been pursued. The PRODIGY [CBE+92] system is an end-means planner that includes mechanisms for encoding search control knowledge. The emphasis there was on learning [Min88] and computing [Etz93] control information automatically. This information was expressed as a set of rules in a rule-based system that is used to guide the non-deterministic choices made during search.

More recently Bacchus and Kabanza [BK96] have pioneered an approach in which domain specific information is encoded *declaratively* in a temporal logic. These logical expressions are then used to prune bad action sequences from the search space of a forward chaining planner. In a recent study [BK98] they have demonstrated that useful information can be obtained in a range of domains, that their formalism can represent this information in a fairly natural way, and that with this information exponential speedups can be achieved over planning systems that do not exploit domain specific information.[1] Kautz and Selman [KS98] have also recently explored using domain specific information to speed up planning. In their approach the planning problem is encoded as a theory of propositional logic. The theory has the property that any satisfying model can be easily translated into a plan. They have explored encoding domain specific information with additional propositional formulas (hence like Bacchus and Kabanza their representation of this information is also declarative). By conjoining these additional formulas to the base theory it is often possible to speed up the process of finding a model. In their work they have reported speed ups of up to a factor of 7 gained by adding domain specific information (although they have not as yet obtained the exponential speedups reported by Bacchus and Kabanza). Finally, Srivastavan and Kambhampati [SK98] have presented a scheme where domain specific knowledge, represented in a declarative fashion, is used as input to an automated programming system. This knowledge is used by the system to construct a domain specific planner. These approaches share

*This research was supported by the Canadian Government through their IRIS project and NSERC programs.

[1] Specifically, planning problems that require exponential time using the fastest domain independent planners can be solved in polynomial time once appropriate domain specific information is provided.

some commonality in that they all exploit information that helps the planner avoid considering some plans that are bound to be flawed. This is accomplished by making explicit certain implicit constraints on plans in the domain.

In this paper we explore another way of representing domain specific control knowledge: as extra preconditions in the operators.[2] Such preconditions can be used to stop the planner from considering particular actions in circumstances where we know that such actions are inappropriate. Encoding control knowledge in this manner has a number of potential advantages. First, it can make the knowledge easier to understand, as one only needs to understand the basic operator representation. Second, it can potentially be used by any planning system; no modifications to the system's algorithms are needed. We will show that the first advantage can be achieved, but that there are difficulties to be overcome in achieving the second. In particular, we show that precondition control can be very effective when using a forward chaining planner with a very expressive operator representation.

Certain types of control information are not so easily expressed as extra preconditions. So precondition control does not seem to be as expressive as the temporal logic proposed by Bacchus and Kabanza. This means that precondition control cannot, as yet, be utilized in as wide a range of different domains. There are also difficulties that need to be addressed before we can utilize such control with other types of planning algorithms. Nevertheless, despite these limitations, precondition control possesses one very large advantage: when it is applicable it can yield tremendous speedups over competing approaches for representing domain specific control.

In this paper we concentrate on comparing the performance of precondition control to the temporal logic approach developed by Bacchus and Kabanza. In their recent study they have shown that their approach using temporal logic yields speedups that are considerably superior to the other approaches for speeding up classical planners mentioned above, and that in many cases their approach provides an exponential speedup over planners that are not provided with domain specific information [BK98]. Hence, since precondition control (when it can be applied) can offer significant improvements over the temporal logic approach, it seems to be a very promising mechanism for moving classical AI planners closer to the point where they can address realistic planning problems.

In the next section we illustrate how precondition control can be applied, using the standard logistics domain as an example. Then we present some empirical data showing the speedups it can yield. Finally, we discuss the issues that must be addressed before precondition control can be utilized in other planning algorithms.

2 The Logistic Domain

The logistics domain is a standard test domain for planners that utilize STRIPS operators. The domain is a simple transportation domain where we have trucks and planes. Trucks can move to different locations within the same city and planes can move between airports. To move an object it has to be loaded into a vehicle, that vehicle has to be moved, and the object unloaded. The operators in this domain are shown in figure 1.

In our experiments we utilize Bacchus's TLPLAN planning system [Bac98]. This planner is a forward chaining planner that has an extremely expressive representation. Operators can be encoded using an extended version of Pednault's action description language, ADL [Ped89]. The system's expressiveness is a key factor in being able to express domain specific information as extra preconditions. We will introduce some of the features of the TLPLAN system as we utilize them.

TLPLAN allows one to define new predicates as abbreviations for (arbitrary) first order formulas. For example, we can define the predicate **vehicle** that is true of either trucks or airplanes:

```
(def-predicate (vehicle ?v)
  (or (truck ?v) (airplane ?v)))
```

This predicate makes for concise operator descriptions. The operators are specified by providing a name, e.g., **load**, a precondition, and then a sequence of formulas that include specifications of add and delete operations. The precondition starts with a sequence of variables (symbols commencing with a `?`) and ranges for these variables. For example, in the **load** operator, `?obj` and `?loc` are allowed to range over the set of pairs of constants that satisfy `(at ?obj ?loc)` in the current world, `?v` is allowed to range over all constants that satisfy `(at ?v ?loc)`, where `?loc` has already been bound by the previous range. The precondition terminates with a first-order formula that must evaluate to true in the current world. In this case the bindings for `?obj` and `?v` must satisfy the conjunction `(and (vehicle ?v) (object ?obj))`. Every distinct set of bindings of the precondition variables that satisfies the terminal formula (if there is one) generates a particular action, i.e., an instance of the operator. For each action the successor state under that action is computed by evaluating each formula in the body of the operator (with the precondition variables bound). These formulas can include **add** and **del** clauses, which cause particular atomic formulas to be added to and deleted from the state's STRIPS database.

Other planners have implemented ADL operators, e.g., UCPOP [PW92] and IPP [KNHD97]. However, these systems do not allow disjunctive or existential preconditions. For example, one cannot specify a single **load** operator in these systems as this operator contains a disjunction in its precondition (hidden in the `vehicle` predicate). Instead these systems must implement the logistics domain with two load operators, one for trucks and another for airplanes.

A number of domain specific rules or constraints on plans are immediately apparent in this domain.

1. Load and unload a vehicle prior to moving it.
2. Don't move a vehicle to a irrelevant location.
3. Don't load or unload objects from a vehicle unless we need to.

[2] Preconditions in operators have previously been used for a number of different purposes, including control. Such a use of preconditions occurs in the SIPE system [Wil88]. However, we have not seen any previous attempt to quantify the effectiveness of this approach.

```
(def-adl-operator (load ?obj ?v ?loc)
  (pre (?obj ?loc) (at ?obj ?loc)
       (?v)        (at ?v ?loc)
       (and (vehicle ?v) (object ?obj)))
  (add (in ?obj ?v))
  (del (at ?obj ?loc)))

(def-adl-operator (unload ?obj ?v ?loc)
  (pre (?obj ?v) (in ?obj ?v)
       (?loc)    (at ?v ?loc))
  (add (at ?obj ?loc))
  (del (in ?obj ?v)))

(def-adl-operator (drive ?t ?from ?to)
  (pre (?t)    (truck ?t)
       (?from) (at ?t ?from)
       (?city) (in-city ?from ?city)
       (?to)   (in-city ?to ?city)
       (not (= ?from ?to)))
  (add (at ?t ?to))
  (del (at ?t ?from)))

(def-adl-operator (fly ?p ?from ?to)
  (pre  (?p)    (airplane ?p)
        (?from) (at ?p ?from)
        (?to)   (airport ?to)
        (not (= ?from ?to)))
  (add (at ?p ?to))
  (del (at ?p ?from)))
```

Figure 1: Operators for the Logistics World

Here we utilize the formalization of these constraints originally given by Bacchus and Kabanza [BK98]. Both Kautz and Selman [KS98] and Srivastavan and Kambhampati [SK98] have described similar types of control information for this domain.

We start out with a collection of auxiliary predicates. Given its current location, `?c-loc`, an object, `?obj`, is in the wrong city if it has a goal location and that location is in a different city.[3]

```
(def-predicate (wrong-city ?obj ?c-loc)
  (exists (?g-loc) (goal (at ?obj ?g-loc))
          (?city)  (in-city ?c-loc ?city)
          (not (in-city ?g-loc ?city))))
```

This predicate utilizes another feature of TLPLAN—its ability to refer to properties that hold in the goal. This is accomplished with the system's "goal" modality. In particular, the existential allows `?g-loc` to range over all constants that satisfy the predicate `(at ?obj ?g-loc)` *in the goal state*.[4] Thus this variable will be bound to the object's final destination if there is one. If no destination has been specified in the goal, the existential, and thus the predicate `wrong-city`, will evaluate to false. After binding `?g-loc` to the object's final destination and `?city` to the city it is currently located in,[5] we test if the object's final destination is in its current city.

We need to move an object by truck if it is in the wrong city and not at an airport (in which case it needs to be moved by truck to an airport); or if it is in the right city, it has a final destination, and it is not currently at its final destination (in which case it needs to be moved to its final destination by truck).[6]

```
(def-predicate (move-by-truck ?obj ?c-loc)
  (if-then-else
   (wrong-city ?obj ?c-loc)
   (not (airport ?c-loc))
   (exists (?g-loc) (goal (at ?obj ?g-loc))
           (not (= ?g-loc ?c-loc)))))
```

There is a similar predicate for airplanes. In this case we only need to move an object by airplane if it is in the wrong city.

```
(def-predicate (move-by-plane ?obj ?c-loc)
  (wrong-city ?obj ?c-loc))
```

We need to unload an object from a truck if it is in the wrong city and the truck is at the airport (in which case we get it ready to load into an airplane), or if the truck is at the object's final destination.

```
(def-predicate (unload-from-truck ?obj ?c-loc)
  (or
   (goal (at ?obj ?c-loc))
   (and (wrong-city ?obj ?c-loc)
        (airport ?c-loc))))
```

The similar predicate for airplanes simply needs to determine if the object is in the right city.

```
(def-predicate (unload-from-plane ?obj ?c-loc)
  (not (wrong-city ?obj ?c-loc)))
```

With these predicates in place it is easy to implement precondition control in the logistics domain by augmenting the operator preconditions. Figure 2 displays the new operator preconditions with the additional precondition clauses highlighted. With these new preconditions we are only allowed to load or unload an object if the object should be loaded or unloaded from that type of vehicle at that location. The preconditions use two implications, one for trucks and one for planes, and the auxiliary predicates to enforce this condition. The preconditions for moving a vehicle ensure that we do not move the vehicle if there is an object at its current location that needs to be loaded in the vehicle, or an object in the vehicle that needs to be unloaded at the current location. Furthermore, the end location of the vehicle must either be the goal destination of the vehicle, or there must be an object that needs to be picked up or dropped off at that end location.[7] These preconditions realize the three constraints on logistics plans mentioned above: we aren't allowed to move a vehicle until it is loaded and unloaded, we don't move a vehicle to

[3] Note that objects that do not have a specified final destination are never in the wrong city.

[4] The goal modality can be used whenever the goal is specified as a set of atomic facts to be achieved.

[5] Quantifiers in the TLPLAN system allow for nested variables. In this case the existential variable `?city` lies within the scope of the existential variable `?g-loc`.

[6] `(if-then-else f1 f2 f3)` is simply an abbreviation for `(and (implies f1 f2) (implies (not f1) f3))`.

[7] One could imagine a more complex precondition that did not move the vehicle to its goal destination while there still existed objects that could be transported by the vehicle.

```
(def-adl-operator (load ?obj ?v ?loc)
 (pre (?obj ?loc) (at ?obj ?loc)
      (?v)        (at ?v ?loc)
      (and (vehicle ?v) (object ?obj)
           (implies (truck ?v)
              (move-by-truck ?obj ?loc))
           (implies (airplane ?v)
              (move-by-plane ?obj ?loc))))

(def-adl-operator (unload ?obj ?v ?loc)
 (pre (?obj ?v) (in ?obj ?v)
      (?loc)    (at ?v ?loc)
      (and (implies (truck ?v)
             (unload-from-truck ?obj ?loc))
           (implies (airplane ?v))
              (unload-from-plane ?obj ?loc))))

(def-adl-operator (drive ?t ?from ?to)
 (pre (?t)    (truck ?t)
      (?from) (at ?t ?from)
      (?city) (in-city ?from ?city)
      (?to)   (in-city ?to ?city)
      (and (not (= ?from ?to))
           (not (exists (?obj) (at ?obj ?from)
                  (move-by-truck ?obj ?from)))
           (not (exists (?obj) (in ?obj ?t)
                  (unload-from-truck ?obj ?from)))
           (or (goal (at ?t ?from))
               (exists (?obj) (at ?obj ?to)
                 (move-by-truck ?obj ?to))
               (exists (?obj) (in ?obj ?t)
                 (unload-from-truck ?obj ?to))))))

(def-adl-operator (fly ?p ?from ?to)
  (pre (?p)    (airplane ?p)
       (?from) (at ?p ?from)
       (?to)   (airport ?to)
       (and (not (= ?from ?to))
            (not (exists (?obj) (at ?obj ?from)
                   (move-by-plane ?obj ?from)))
            (not (exists (?obj) (in ?obj ?p)
                   (unload-from-plane ?obj ?from)))
            (or (goal (at ?p ?from))
                (exists (?obj) (at ?obj ?to)
                  (move-by-plane ?obj ?to))
                (exists (?obj) (in ?obj ?p)
                  (unload-from-plane ?obj ?to))))))
```

Figure 2: Modified Preconditions for the Logistics World (changes in **bold**).

an irrelevant location, and we don't load or unload an object unless we need to.

3 Experimental Results

To test the effectiveness of precondition control we attempted to solve the 30 test problems that were created for the AIPS-98 planning competition [AIP98]. The planners in the competition did not utilize domain specific control, and as a result found these test problems very difficult—they were only able to solve between 3 and 5 problems out of the 30. Table 1 shows the results we obtained. In the table the rows labeled "Pre" are the times obtained with precondition control, and those labeled with "TL" indicate the times obtained with the equivalent temporal logic control (measured in seconds of CPU usage). We see that precondition control in this domain gives tremendous speedups. For example, problem 27 ran almost 400 times longer using temporal logic control and was still unable to find a solution in that time. It should be noted that this is the first time (to our knowledge) that this suite of problems has been solved. In contrast during the planning competition, the fastest times reported were L1 in 0.8 sec., L2 in 4.3 sec., L5 in 2.4 sec., L7 in 788.9 sec., and L11 in 6.5 sec. No other problems were solved during the competition.[8] This illustrates the effectiveness of providing domain specific control information.

Another experiment we ran was one in the Blocks world. Again we took the control information provided by Bacchus and Kabanza, converted it to precondition control, and then compared the performance of precondition control with their temporal logic control. Specifically, Bacchus and Kabanza define the following "goodtower" predicate:

```
(def-predicate (goodtower ?x)
  (and (clear ?x)
       (goodtowerbelow ?x)))

(def-predicate (goodtowerbelow ?x)
 (or
  (and (ontable ?x)
       (not (exists (?y) (goal (on ?x ?y)))))
  (exists (?y) (on ?x ?y)
    (and
      (not (goal (ontable ?x)))
      (not (goal (clear ?y)))
      (forall (?z) (goal (on ?x ?z)) (= ?z ?y))
      (forall (?z) (goal (on ?z ?y)) (= ?z ?x))
      (goodtowerbelow ?y)))))
```

A block satisfies the predicate `(goodtower ?x)` if it is on top of a tower, i.e., it is clear, and the tower below it does not violate any goal conditions. The various tests for the violation of a goal condition in the tower below are given in the definition of `goodtowerbelow`. If `?x` is on the table, the goal cannot require that it be on another block `?y`. On the other hand, if `?x` is on another block `?y`, then `?x` should not be required to be on the table, and nor should `?y` be required to be clear. Any block that is required to be below `?x` should be `?y`, any block that is required to be on `?y` should be `?x`, and finally the tower below `?y` cannot violate any goal conditions. The key idea behind `goodtower` is that any block satisfying it need never be moved to achieve the goal—neither it nor any block below it needs to be moved.

With this predicate we can implement precondition control

[8] Our timings were obtained on a 400MHz Pentium II with 1GB of RAM, while during the competition a 266MHz Pentium II with 128MB of RAM was used. Nevertheless, the hardware difference cannot fully account for the difference in performance. It is possible, however, that some of the other problems could have been solved in the competition with more memory.

Prob #	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10
Pre	0.046	0.144	0.681	2.061	0.044	1.545	0.259	1.121	2.811	4.588
TL	0.376	1.761	15.924	44.825	0.267	69.849	4.622	79.186	177.876	138.873

Prob #	L11	L12	L13	L14	L15	L16	L17	L18	L19	L20
Pre	0.356	2.402	4.862	5.248	0.849	2.498	1.429	22.678	12.165	18.733
TL	4.278	236.382	890.176	653.132	19.875	142.494	75.488	3655.668	2306.783	**2519.612**

Prob #	L21	L22	L23	L24	L25	L26	L27	L28	L29	L30
Pre	9.358	148.724	2.837	3.257	54.369	30.707	21.881	196.988	300.786	30.499
TL	1781.584	**9589.805**	100.939	582.485	**2284.022**	**5108.930**	**8602.591**	**12548.064**	**15752.647**	**3256.335**

Table 1: Search Control in Logistics World. Time to solve the problem in CPU seconds. **Bold face** entries indicate problem not solved at end of this time.

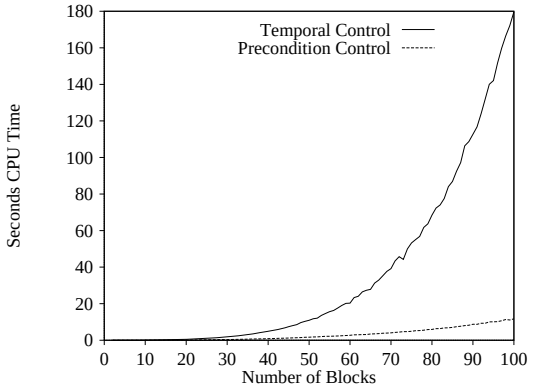

Figure 3: Temporal Logic Control vs. Precondition Control in the Blocks world

by adding the following extra preconditions:

1. `(pickup ?x)` only if `(exists (?y) (goal (on ?x ?y)) (goodtower ?y))`. That is, only if there is a block `?y` that `?x` should be placed on, and that block is in its final position.
2. `(putdown ?x)` only if `(not (exists (?y) (goal (on ?x ?y)) (goodtower ?y)))`. That is, only when `?x`'s final location is not ready. (Otherwise we should use `stack` instead of `putdown`).
3. `(stack ?x ?y)` only if `(and (goal (on ?x ?y)) (goodtower ?y))`. That is, only if `?x` is intended to be put on `?y`, and `?y` is in its final position.
4. `(unstack ?x ?y)` only if `(not (goodtower ?x))`. That is, don't move `?x` if it is already in its final position.

Figure 3 shows the relative performance in this domain of the original temporal logic control and the new precondition control. Each data point in the graph represents the average CPU time required to solve a suite of 10 randomly generated blocks world problems of that size. One hundred data points were collected (representing 1000 tests) for each plot.[9]

[9]This data was generated on a 200MHz Pentium Pro with 128MB RAM.

Again we achieve a significant speedup using precondition control. Precondition control solves the random blocks world problems involving 100 blocks in an average of 11.7 sec., while the temporal logic control requires about 179.6 sec. In Bacchus and Kabanza's experiments [BK98] the fastest domain independent planners cannot solve any of these problems when their size becomes greater than 12. This once again demonstrates the effectiveness of providing domain specific control information.

Space precludes discussing further examples, but in our experiments we have implemented precondition control for some other test domains including the classic monkeys and bananas domain, the towers of Hanoi, and the briefcase domain.

4 Applicability of Precondition Control

As pointed out in our introduction, expressing domain control information by simply adding preconditions has a number of potential advantages. As the previous section illustrates, information so expressed can be quite simple to understand, and in our experiments we have found that precondition control is applicable in quite a wide variety of planning domains.

In examining the range of control information employed by Bacchus and Kabanza in [BK98] we have, however, found that some control information expressible in their temporal logic is not so easily expressed as extra preconditions. When the control information tells us what not to do in the next state it generally can be easily encoded as extra preconditions. However, in some of their examples, the control knowledge forces the planner to do something next. Telling the planner what to do next in certain situations is more difficult to express with operator preconditions. Preconditions seem more naturally suited to prohibiting actions, not forcing them. There are other more complex examples of this phenomenon when one has control information that forces the planner to achieve a set of conditions in a particular order. It remains an open problem as to whether or not a systematic mechanism can be developed for encoding a wider range of temporal logic control as precondition control. The advantage of developing such a mechanism, as illustrated in the previous section, is the increase in planning efficiency it maybe able to provide.

Even without the ability to express all forms of temporal

logic control as precondition control, however, we have observed that converting even part of the temporal logic control into precondition control (yielding an amalgam of both types of control) can generate useful performance improvements.

Besides capturing control information in a more understandable form, the other allure of precondition control is that it could potentially be applied to any planning system. However, there are two difficulties that need to be addressed. First, is the expressiveness we require in the operator descriptions. In the examples above the logistics domain required including disjunctive and existential conditions in the preconditions, and the blocks world required a recursive precondition. As noted above the other planners that have implemented ADL operators have not allowed for this generality in operator preconditions.

Nevertheless, it is possible to encode some of this information by doing things like adding new primitive predicates to the domain. By modifying the adds and deletes of the operators it is often possible to update such predicates so that they faithfully represent, e.g., existential conditions. Similarly one could represent information about the goal by adding a new set of "goal" predicates to the initial state. It is an interesting area for future work to develop ways of encoding richer preconditions into such planners.

The second difficulty in utilizing precondition control in other planning systems is that is not clear if extra preconditions will help reduce the search space explored by such planners. In a forward chaining planner there is a clear correlation between extra preconditions and a reduction in the number of successors of the current state. That is, extra preconditions cut down the branching factor in the forward direction. Partial order planners, on the other hand, are more heavily influenced by the backwards, or regressive, branching factor. So it is not clear if extra preconditions will help them. The question is more ambiguous for GraphPlan-style planners [BF97]. Such planners build up a reachability graph, so extra preconditions may well serve to limit this graph and thus help in the overall performance of the planner.

In future work we plan to investigate further the range of applicability of precondition control. For now, however, we have been able to demonstrate the form of such control information, show how effective it can be, and point out the some of the requirements it imposes on the operator representation. The performance we obtain when precondition control is applicable suggests that the direction of implementing richer operator representations, pioneered in classical planning by the work on UCPOP [PW92] is worth further effort.

References

[AIP98] AIPS98. Artificial Intelligence Planning Systems 1998 planning competition. http://ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html, 1998.

[Bac98] Fahiem Bacchus. Tlplan planning system. Software available from http://www.lpaig.uwaterloo.ca/~fbacchus/tlplan.html, 1998.

[BF97] Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

[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.

[BK98] Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. Under review, currently available at http://www.lpaig.uwaterloo.ca/~fbacchus/online.html, 1998.

[CBE+92] J.G. Carbonell, J. Blythe, O. Etzioni, Y. Gill, R. Joseph, D. Khan, C. Knoblock, S. Minton, A. Pérez, S. Reilly, M. Veloso, and X. Wang. Prodigy 4.0: The manual and turorial. Technical Report CMU–CS–92–150, School of Computer Science, Carnegie Mellon University, 1992.

[ENS92] K. Erol, D.S. Nau, and V.S. Subrahmanian. On the complexity of domain-independent planning. In *Proceedings of the AAAI National Conference*, pages 381–386, 1992.

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

[KNHD97] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In *European Conference on Planning*, pages 273–285, 1997. (System available at http://www.informatik.uni-freiburg.de/~koehler/ipp.html).

[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.

[Ped89] E. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pages 324–332, 1989.

[PW92] J.S. Penberthy and D. Weld. UCPOP: A sound, complete, partial order planner for adl. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pages 103–114, 1992.

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

[Wil88] David Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann, San Mateo, California, 1988.