Heuristic Planning: a Declarative Forward Chaining Approach.

Josefina Sierra-Santibáñez
Universidad Autónoma de Madrid
Josefina.Sierra@ii.uam.es

Abstract

This paper introduces the notion of *heuristic planning*, and describes a particular approach to heuristic planning based on a *declarative formalization of strategies for action selection*. This approach is compared with some heuristic planning systems proposed in the literature. The heuristic information and declarative formalisms for the representation of heuristic knowledge used by these systems are compared in terms of their capacity of controlling the search process and their effectiveness for solving some planning problems.

1 Introduction

This paper introduces the notion of heuristic planning, and describes a particular approach to heuristic planning based on a declarative formalization of strategies for action selection proposed in [Sierra, 1998]. This approach is compared with the proposals of [Bacchus and Kabanza, 1996b], [Reiter, 1999] and [Doherty and Kvarnstrom, 2000]. The heuristic information and declarative formalisms for the representation of heuristic knowledge used by these systems are compared in terms of their capacity of controlling the search process and their effectiveness for solving some planning problems.

Heuristic planning is a particular case of the concept of heuristic search used in problem solving. Planning domains are traditionally represented in the planning literature by a set of STRIPS [Fikes and Nilsson, 1971] or ADL [Pednault, 1989] operators describing a set of available actions. Heuristic planning requires a representation of planning domains which includes as well heuristic information about the subregions of the search space in which a successful plan solution may be found. In heuristic search, this information is usually described by a heuristic function which estimates the distance of a given state to a goal state in the search space. Heuristic planning approaches, such as [Bacchus and Kabanza, 1996b], [Sierra, 1998] and [Reiter, 1999], use different representation formalisms to express that information. These formalisms tend to be declarative [McCarthy, 1959] and make use of logical languages such as the *situation calculus* [McCarthy and Hayes, 1969] or the language of *linear temporal logic*. Some of these approaches [Bacchus and Kabanza, 1996b] and [Reiter, 1999] focus on the pruning aspect of the heuristic information by detecting those states from which the search process should not proceed. Others, such as [Lin, 1997] and [Sierra, 1998], allow the representation of information that can also be used for leading the search process in promising directions and for establishing a partial order among the states from which the search can proceed.

Heuristic planning differs from the approaches to *planning as heuristic search* described in [Bonet and Geffner, 1999] and [Nguyen and Kambhampati, 2000] in the fact that the heuristic information is provided by the user, rather than automatically generated by planner from the description of a planning problem. The relation between the domain dependent approach to planning, represented by heuristic planning, and the domain independent approach, represented by planning as heuristic search, is very interesting in itself and its study could bring new ideas for both areas of research. Unfortunately, it falls out of the scope of this paper.

2 Heuristic Planning

We describe a heuristic planning system based on the proposals of [Sierra, 1998] for the declarative formalization of strategies for action selection. The *heuristic forward chaining planner* (HFCP) has been implemented in Prolog and uses the representation scheme for planning domains described in that paper.

A *planning domain* is specified by a set of available actions and a strategy for action selection. A *strategy for action selection* is a consistent set of action selection rules. An *action selection rule* [Genesereth and Hsu, 1989] is an implication whose antecedent is a formula of the situation calculus [McCarthy and Hayes, 1969], and whose consequent can take one of the following forms: $Good(a, s)$, $Bad(a, s)$ or $Better(a, b, s)$. The intuitive interpretation of these predicates is that performing action a at situation s is *good, bad* or *better* than performing action b.

The heuristic forward chaining planner HFCP explores the space of situations that can be reached from the initial situation performing executable sequences of actions using a heuristic search strategy, which is guided by the strategy for action selection supplied by the user as follows.

The predicate *Good* is used to characterize *optimal* actions. These are actions whose execution will always lead to finding an optimal plan solution[1]. This can be observed by examining the heuristics for the blocks world described below. Actions of the type *good* are used then to direct the search in the right direction, because they guarantee that an optimal solution will be found in that direction. They are used, therefore, to restrict the search to the subtree generated by applying a good action to the current situation.

The predicate *Bad* is used to characterize those actions which cannot be part of an optimal plan solution. Therefore, every subplan containing a bad action should not be considered by the planning system. Actions of the type *bad* are used, then, to prune the search tree considered by the planner at each situation. If an action is bad for the current situation, then the planner knows that it needs not generate or consider the situation resulting from applying that action to the current situation, nor any of the descendants of that situation. They are used, therefore, to detect and avoid the generation of subtrees which cannot contain an optimal plan solution.

The predicate *better* is used to establish a partial order among actions which are not known to be good or bad for a particular situation. In some aspects, it acts as a heuristic function in a heuristic search algorithm, because it can be used to order the set of possible actions in the current situation. The predicate better is used, then, to control the backtracking process of the search strategy, determining at each situation the action that should be tried out of the remaining untried actions. [Lin, 1997] formalizes a partial order relation on actions as well, and uses it to provide a formal semantics for the Prolog *cut* operator.

In summary, the action selection mechanism of the heuristic planner works as follows. If there is a good action for the current situation, it explores only the subtree of situations that can be generated from the situation resulting of applying this action to the current situation. If there is no good action for the current situation, it explores the subtrees of situations that can be generated by applying non bad actions to the current situation using the partial order established by the predicate better to guide its search process.

2.1 An Example of Strategy for Action Selection

We present an example of strategy for action selection that can be used for planning in the blocks world. The strategy consists of a set of action selection rules (represented as Prolog rules) which describe some *heuristics* for moving blocks in order to solve planning problems in the blocks world: *(1) If a block can be moved to final position, this should be done right away; (2) If a block is above another block it ought to be above but it is not in final position (tower-deadlock), put it on the table*; (3) It is better to move a block that is not in final position from another block that is in final position and should have third block on it in the goal configuration, than moving a block from another block that is not in final position or should not have any block on it in the goal configuration; (4) it is better to move a block from another block that, if cleared up, can be put immediately in final position, than moving a block from another block that could not be put immediately in final position; (5) If a block is in final position, do not move it; (6) If a block is neither on the table nor in final position and cannot be moved to final position, do not move it to any place different from the table.

These heuristics use three concepts which are not included in the usual descriptions of the blocks world. Therefore, their definitions must be part of the specification of the strategy for action selection. A block is in *final* position if it is on the table and it should be on the table in the goal configuration, or if it is on a block it should be on in the goal configuration and that block is in final position. A block is in *tower-deadlock* position if it is not in final position and it is above a block it should be above in the goal configuration. If a block is in tower-deadlock position, then it must be moved somewhere else (for example, to the table) before it can be put in final position. A clear block can be put in final position in the next situation (*final-next*) if it should be on the table or if it should be on another block that it is currently clear and in final position. The constants t and sg used in the program denote, respectively, the table of the blocks world and the situation sg which is used to describe the goal configuration of a planning problem.

```
/* DEFINITIONS */

above(X,Y,S) :- on(X,Y,S).
above(X,Y,S) :- on(X,Z,S), above(Z,Y,S).

final(X,S) :- on(X,t,sg), on(X,t,S).
final(X,S) :- on(X,Y,sg), not(Y=t),
              on(X,Y,S), final(Y,S).

tower_d(X,S) :- above(X,Y,sg), not(Y=t),
  above(X,Y,S), not(final(X,S)).

next_final(X,S) :-
     on(X,Y,sg), not(final(X,S)),
     (Y=t ; (clear(Y,S), final(Y,S))).

/* ACTION SELECTION RULES */

good(move(X,Y,Z),S) :- on(X,Y,S),
           clear(X,S), not(final(X,S)),
           on(X,Z,sg), (not(Z=t) ;
           (clear(Z,S), final(Z,S))).
good(move(X,Y,t),S) :- on(X,Y,S),
              clear(X,S), tower_d(X,S).

better(move(_,Y,t),move(_,W,t),S) :-
    final(Y), on(_,Y,sg),
    (not(on(_,W,sg)) ; not(final(W,S))).
better(move(_,Y,t),move(_,V,t),S) :-
  next_final(Y,S), not(next_final(V,S)).

bad(move(X,_,_),S) :- final(X,S).
bad(move(X,_,Z),S) :- on(X,W,sg),
             not(Z=t), (not(clear(W,S)) ;
             not(final(W,S))).
```

[1]In this paper, we assume that an optimal plan solution is a plan with a minimum number of actions. This notion of optimality is different from those proposed in [Lin, 1998].

3 Pruning Approaches

3.1 TLPlan

In [Bacchus and Kabanza, 1996b] and [Bacchus and Kabanza, 2000], a planning system called TLPlan, which uses first order linear temporal logic to represent search control knowledge, is described. This logic is interpreted over sequences of worlds. In particular, the *goal* and *temporal modalities* ($\bigcup$ until, $\Box$ always, $\Diamond$ eventually, and $\bigcirc$ next) are used to assert properties of world sequences. A *search control formula* describing the search control strategy to be used by the planner is specified in this logic. This formula describes properties the sequences of worlds generated by applying successful plans to the initial situation should satisfy.

The planner uses a *progression algorithm* which serves as the basis for an incremental mechanism that allows checking whether a plan prefix, generated by forward chaining, could lead to a plan that satisfies the search control formula. Whenever a plan prefix does not satisfy the search control formula, all the sequences of actions starting with that prefix are pruned from the search tree of executable sequences of actions. This mechanism allows reducing the search space by avoiding the exploration of sequences of actions which can never be extended to successful plan solutions.

Interesting experiments in which TLPlan outperforms state of the art planners, such as BlackBox [Kautz and Selman, 1998] and IPP [Koehler *et al.*, 1997] in various test domains using search control formulas are described [Bacchus and Kabanza, 2000]. Blackbox and IPP are both state of the art planning systems. They were the best performers in the AIPS'98 planning competition [AIPS, 1998]. TLPlan did not compete however in [AIPS, 2000]. The best performer in the domain dependent competition of [AIPS, 2000] was TALPlan [Doherty and Kvarnstrom, 2000] and [Kvarnstrom and Doherty, 2000]. TALPlan uses an approach for the representation and use of search control knowledge similar to that of TLPlan. We describe some specific features of TALPlan at the end of this section.

TLPlan is an interesting example of a heuristic forward chaining planner, in which search control knowledge is expressed in terms of properties the sequences of worlds generated by *successful plans* must satisfy. This heuristic information is used for pruning the search tree during the searching process, but it is not exploited for directing the search in promising directions or for establishing a partial order on the set of plan prefixes which may be extended to successful plans solutions. The search control formula used for the blocks world in [Bacchus and Kabanza, 2000] is described below.

$$goodtower(x) \equiv clear(x) \wedge \neg Goal(holding(x)) \wedge goodtowerbelow(x)$$

$$\begin{aligned} goodtowerbelow(x) \equiv\ & (ontable(x) \wedge \neg\exists[y : Goal(on(x,y))]) \vee \\ & \exists[y : on(x,y)] \neg Goal(ontable(x)) \wedge \neg Goal(holding(y)) \wedge \\ & \neg Goal(clear(y)) \wedge \forall[z : Goal(on(x,z))] z = y \wedge \\ & \forall[z : Goal(on(z,y))] z = x \wedge goodtowerbelow(y) \end{aligned}$$

$$badtower(x) \equiv clear(x) \wedge \neg goodtower(x)$$

$$\begin{aligned} \Box\, (& (\forall[x : clear(x)] goodtower(x) \rightarrow \bigcirc(clear(x) \vee \\ & \exists[y : on(y,x)] goodtower(y))) \wedge \\ & (badtower(x) \rightarrow \bigcirc \neg\exists[y : on(y,x)]) \wedge \\ & (ontable(x) \wedge \exists[y : Goal(on(x,y))] \neg goodtower(y) \rightarrow \\ & \bigcirc \neg holding(x))) \end{aligned}$$

It is easy to observe that the concepts defined by the predicates *goodtowerbelow(x)* and *final(x,s)* (used in the strategy for action selection of section 2.1) are equivalent, although the definition of *final(x,s)* is considerably simpler. The search control formula described above rules out any sequence of actions that disassembles a good tower, places a block on top of a bad tower, or picks up a block on the table unless that block can be moved to its final position. The two action selection rules which define the predicate *bad(a,s)* in the strategy for action selection of section 2.1 prune the same sequences of actions as this search control formula. Because, they forbid actions which move blocks that are already in final position (i.e., which disassemble good towers), and actions which move blocks that cannot be moved to final position to places different from the table.

The search control formula used by TLPlan does not recognize however some other sequences of actions that cannot be extended to construct optimal plans for the blocks world. These correspond to the action selection rules which characterize good actions in the strategy for action selection described in section 2. In order to take these heuristics into account in the representation formalism used by TLPlan, it would be necessary to rule out any sequence of actions which does not include a good action when that action can be performed in the current state. However, this approach would not direct the search in the right direction as the predicate good does. It will have to generate each successor sequence of actions, test it to determine whether it falsifies the search control formula, and prune the subtree of sequences of actions that can be generated from it in that case.

The formalism used by TLPlan has also been applied to generate plans that satisfy *temporally extended goals* [Bacchus and Kabanza, 1996a], which generalize the *safety* and *maintenance* goals mentioned in [Weld and Etzioni, 1994].

TALPlan [Doherty and Kvarnstrom, 2000] is a forward chaining planner which utilizes domain dependent knowledge to control search in the state space generated by action invocation. The domain dependent control knowledge, background knowledge, plans and goals are all represented using formulas in a temporal logic called TAL. TAL has been developed independently as a formalism for specifying agent narratives and reasoning about them. The domain control knowledge used by TALPlan in the blocks world is a translation of the search control formula used by TLPlan to the temporal logic TAL. Search control formulas are used for pruning the search space in a similar way in both systems, with the exception that TLPlan uses formula progression and the efficient version of TALPlan uses formula evaluation combined with some optimization techniques. These techniques allow moving part of the search control information from the search control formula to the preconditions of the operators. This improves performance significantly, since control rule violations can be detected before the planner even attempts to invoke an operator. It should be observed that this form of *precondition control* information is equivalent to the information encoded in action selection rules of the form $F(s) \rightarrow Bad(a, s)$. These rules are evaluated before operator invocation as well, and are therefore as efficient as precondition control information. Finally, and in addition to its impressive performance, TALplanner has the advantage that it has been extended to deal with concurrent actions, actions with arbitrary integer time duration and time dependent effects, and allocation of resources [Kvarnstrom and Doherty, 2000].

3.2 The Predicate BadSituation

In [Reiter, 1999], a forward chaining planner, which uses a regression based theorem prover and an iterative deepening search strategy, is proposed. The planner requires the following types of information from the user: (1) a predicate $goal(s)$, which is true if situation s satisfies the conditions of the goal for which a plan is sought; (2) a set of *action precondition and successor state axioms* [Reiter, 1993] for the primitive actions of the domain; and (3) a predicate *badSituation*(s), which is true if situation s is considered to be a bad situation for the planner to explore. The planner is implemented in GOLOG [Levesque *et al.*, 1997], and it has been extended to deal with incomplete initial situations [Finzi *et al.*, 2000].

The representation formalism used by the heuristic forward chaining planner described in section 2 is more expressive than that used in [Reiter, 1999], in the sense that it allows the representation of *positive heuristics* (the predicate *good* tells the planner what to do), and heuristics that establish preferences among actions (the predicate *better* establishes a partial order among actions). The predicate *badSituation(s)* allows pruning the search space by characterizing those situations from which a successful plan cannot be reached, but it does not allow guiding the search in promising directions or controlling the backtracking process establishing a partial order on successor situations, as the predicates *good* and *better* do.

The heuristics for the blocks world used in [Reiter, 1999] are described below. They are Prolog rules which define the predicate *badSituation(s)* that is used for pruning the search tree of reachable situations explored by the planner.

```
badSituation(do(move(X,Y),S)) :-
 not goodTower(X,do(move(X,Y),S)).

badSituation(do(moveToTable(X),S)) :-
 goodTower(X,S).

badSituation(do(moveToTable(X),S)) :-
 not goodTower(X,do(moveToTable(X),S)),
 existsActionThatCreatesGoodTower(S).

existsActionThatCreatesGoodTower(S) :-
      (A=move(Y,X) ; A=moveToTable(Y)),
       poss(A,S), goodTower(Y,do(A,S)).
```

These heuristics are very similar to the heuristics used by TLPlan. They characterize those situations from which a successful plan cannot be reached. However, as situations are equivalent to sequences of actions in the axiomatization of the situation calculus used by [Reiter, 1999], one can safely say that the heuristics characterize, in fact, bad sequences of actions as the heuristics used by TLPlan do. The main difference is that TLPlan uses linear temporal logic formulas to characterize legal sequences of actions, pruning away those sequences of actions which do not satisfy the search control formula; and [Reiter, 1999] uses situation calculus formulas to characterize bad sequences of actions, pruning away those sequences of actions that satisfy the predicate bad situation.

The heuristics used in [Reiter, 1999] prune however more situations than those used by TLPlan, because, in addition to prohibit building up bad towers and disassembling good towers, they also recognize the fact that whenever there is an action that builds a good tower (i.e., a good action) the situation resulting from performing any action which does not build a good tower should be considered as a bad situation. This amounts to forbidding any non good actions whenever there is a good action for the current situation, as the mechanism for action selection of the heuristic forward chaining planner described in section 2 does. This domain independent heuristic could be incorporated into the planner described in [Reiter, 1999], by adding the following rule to the definition of the predicate *badSituation*.

```
badSituation(do(A,S)) :-
                  poss(B,S), good(B,S),
                  not(good(A,S)).
```

The addition of this rule would allow the planner to use the information provided by action selection rules whose consequent is of the form *Good(a,s)* for pruning the search space, but it would not allow it to use them for directing its search process as the heuristic forward chaining planner described in section 2 does.

The heuristics in [Reiter, 1999] do not consider however an important concept which allows guiding the search process as well. This is the concept of a block being in *tower-deadlock* position. Therefore, they cannot discriminate between actions that move arbitrary blocks to the table (which are not necessarily optimal and can be postponed), and actions that move blocks in tower deadlock position to the table (which are necessary and should be executed right away). This is the meaning of the second action selection rule in the definition of the predicate *good* in the action selection strategy of section 2.1.

The programming language Golog and its interpreter have been extended to deal with uncertainty and general reward functions as well. DTGolog [Boutilier *et al.*, 2000] is based on the synthesis of Markov Decision Processes (MPDs) with the Golog programming language. MDPs are a conceptually and computational useful model for decision theoretic planning, but their solution is often intractable. DTGolog provides means to constrain the search for (ideally optimal) policies with a Golog program. The decision theoretic Golog interpreter solves the underlying MDP by adopting a policy that is consistent with the execution of the program and maximizes its expected utility. Action selection strategies, such as the one described in section 2.1, can be seen as MDP policies in the sense that they recommend different actions for particular types of situations with various degrees of confidence (i.e. *good* for optimal actions, and *better* for establishing a partial order on non optimal actions). An interesting question is whether approaches such as DTGolog can be applied to generate action selection strategies, such as the one used in this paper, rather than situation specific policies which can only be used to solve a particular instance of a problem.

4 Empirical Results

The *heuristic forward chaining planner* HFCP described in section 2 has been applied to solve some blocks world problems using the strategy for action selection described in section 2.1. The first problem set (shown in table 1) consists of 10 randomly generated blocks world problems of 25 blocks. The second problem set (shown in table 2) consists of 6 blocks world problems of different sizes. The sizes of the problems are specified in the first column of table 2. For each problem, we have computed the number of blocks that are initially in final and tower deadlock positions (columns *Final* and *TD*). The numbers in the columns *Steps*, *Nodes,* and *Time* correspond to the number of steps of the plans found by the HFCP planner, the number of situations (nodes) explored, and the time in milliseconds spent on planning.

Table 1. Problems of 25 blocks.

Prob	Final	TD	Steps	Nodes	Time
1	1	2	26	26	0
2	0	11	$\boxed{36}$	36	20
3	3	1	$\boxed{23}$	23	0
4	7	0	$\boxed{18}$	18	10
5	7	2	20	20	10
6	1	4	$\boxed{28}$	28	10
7	1	6	$\boxed{30}$	30	20
8	1	13	37	37	20
9	1	3	$\boxed{27}$	27	20
10	1	7	$\boxed{31}$	31	20

Table 2. Problems of different sizes.

Size	Final	TD	Steps	Nodes	Time
5	2	0	$\boxed{4}$	4	0
13	1	3	15	15	0
15	2	0	$\boxed{14}$	14	0
19	2	0	$\boxed{18}$	18	0
25	5	1	22	22	10
50	24	0	26	26	0

We have compared these results with those obtained from running the same problems in TLPlan. The results obtained from running the problems in the first problem set in TLPlan are shown in the table 3, and those obtained from running the problems in the second problem set in table 4.

In order to make a fair comparison we have divided by two the numbers of steps of the plans found by TLPlan. This is due to the fact that the action *move(x,y,z)*, used in the action selection strategy of section 2.1, corresponds to two actions of the form *pickup(x), putdown(x), stack(x,y)* or *unstack(x,y)* used by TLPlan. Therefore, the plans obtained by TLPlan should be twice as long as the plans obtained by the heuristic forward chaining planner. The same transformation has been applied to the number of nodes actually explored by TLPlan.

The numbers in the columns *Steps TLPlan*, *Nodes TLPlan,* and *Time TLPlan* correspond to the number of steps of the plans found by TLPlan, the number of situations (nodes) actually explored, and the time in milliseconds taken by TLPlan.

Table 3. Problems of 25 blocks.

Prob	Final	TD	Steps TLPlan	Nodes TLPlan	Time TLPlan
1	1	2	26	26	58
2	0	11	38	38	91
3	3	1	25	25	58
4	7	0	20	20	52
5	7	2	20	20	46
6	1	4	30	30	68
7	1	6	37	37	91
8	1	13	37	37	85
9	1	3	29	29	68
10	1	7	32	32	84

Table 4. Problems of different sizes.

Size	Final	TD	Steps TLPlan	Nodes TLPlan	Time TLPlan
5	2	0	5	5	4
13	1	3	15	15	19
15	2	0	18	18	26
19	2	0	25	25	47
25	5	1	22	22	51
50	24	0	26	26	158

Comparing the numbers in the columns *Steps* and *Steps TLPlan*, it can be observed that TLPlan cannot find optimal plans (i.e., plans with a minimum number of steps) for 10 of the 16 problems posed. The heuristic forward chaining planner obtains optimal plans for the 16 problems. As far as planning time is concerned, the heuristic forward chaining planner is faster than TLPlan.

5 Conclusions

We have introduced the notion of heuristic planning, and described a particular approach to heuristic planning based on a declarative formalization of strategies for action selection proposed in [Sierra, 1998]. This approach has been compared with the proposals of [Bacchus and Kabanza, 1996b] and [Reiter, 1999]. The heuristic information and declarative formalisms for the representation of heuristic knowledge used by these systems have been analyzed and compared in terms of their capacity of controlling the search process and their effectiveness for solving some planning problems. The following are some conclusions that can be drawn from this comparative analysis.

1. The mechanism for action selection and the declarative formalization of strategies for action selection used by the heuristic planner described in this paper allow controlling the search process in more sophisticated ways than the pruning approaches used by TLPlan and the planning system described in [Reiter, 1999]. In particular, they allow directing the search in promising directions, and controlling the backtracking process in such a way that actions are tried in order of their quality, which is established by the predicate better.
2. The quality of the heuristic information used by the heuristic planner for the domain of the blocks world is slightly better than that used by TLPlan or the planning system described in [Reiter, 1999]. In particular, it uses the concept of *tower deadlock* position, which is not considered in the heuristics used by the other systems.
3. The regression theorem prover used by the planner described in [Reiter, 1999] can be applied to the resolution of planning problems with incomplete initial situations [Finzi *et al.*, 2000], whereas the state update approaches used by TLPlan and the heuristic planner described in this paper cannot be applied to solve these problems. DTGolog has extended this approach as well, allowing it to deal with uncertainty and general reward functions [Boutilier *et al.*, 2000]
4. The representation formalism used by TLPlan has been applied to generate plans that satisfy *temporally extended goals* [Bacchus and Kabanza, 1996a]. In the temporal logic approach to planning, TALplanner has contributed to improve performance of heuristic planners and to extend them to deal with concurrent actions, actions with arbitrary integer time duration and time dependent effects, and allocation of resources [Kvarnstrom and Doherty, 2000].

Finally, we have described some experiments which show that the heuristic forward chaining planner described in section 2 improves the performance of TLPlan in the domain of the blocks world. The improvement of performance allows constructing plans of better quality without sacrificing the amount of search required or the time spent on planning.

Future work will address the study of meta reasoning techniques for declarative formalizations of strategies for action selection, and the automatic generation of heuristics for action selection. Issues such as the safety and postponability of certain actions underlie the design of the domain dependent heuristics used in this paper, and may also be useful for generating such heuristics automatically from the description of a planning problem.

Acknowledgment

The author would like to thank John McCarthy for interesting her on the declarative formalization of heuristics, and providing the intellectual and financial support necessary for the realization of this work. Vladimir Lifschitz contributed with valuable comments.

References

[AIPS, 1998] AIPS-1998, Artificial Intelligence Planning Systems: 1998 Planning Competition. http://ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html, 1998.

[AIPS, 2000] AIPS-2000, Artificial Intelligence Planning Systems: 2000 Planning Competition. http://www.cs.toronto.edu/aips2000, 2000.

[Bacchus and Kabanza, 1996a] F. Bacchus and F. Kabanza. Planning for Temporally Extended Goals. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1215–1222, 1996. American Association for Artificial Intelligence.

[Bacchus and Kabanza, 1996b] F. Bacchus and F. Kabanza. Using Temporal Logic to Control Search in a Forward Chaining Planner. In M. Ghallab and A. Milani, eds. *New Directions in AI Planning*, pages 141–153, IOS Press, Amsterdam, 1996.

[Bacchus and Kabanza, 2000] F. Bacchus and F. Kabanza. Using Temporal Logics to Express Search Control Knowledge for Planning. *Artificial Intelligence*, 116:123–191, 2000.

[Bonet and Geffner, 1999] B. Bonet and Hector Geffner. Planning as Heuristic Search: New Results. In *Proceedings of the European Conference on Planning*, 1999.

[Boutilier *et al.*, 2000] Craig Boutilier, Ray Reiter, Mikhail Soutchanski and Sebastian Thrun. Decision-Theoretic, High-level Agent Programming in the Situation Calculus. In *Proceedings of the National Conference on Artificial Intelligence*, pages 355–362 2000. American Association for Artificial Intelligence.

[Doherty and Kvarnstrom, 2000] Patrick Doherty and J. Kvarnstrom. TALplanner: A Temporal Logic Based Planner. *AI Magazine*. Accepted for publication.

[Fikes and Nilsson, 1971] Richard Fikes and Nils Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2:189–208, 1971.

[Finzi *et al.*, 2000] Alberto Finzi, Fiora Pirri and Ray Reiter. Open World Planning in the Situation Calculus. In *Proceedings of the National Conference on Artificial Intelligence*, 2000. American Association for Artificial Intelligence.

[Genesereth and Hsu, 1989] Michael R. Genesereth and J. Hsu. Partial Programs. TR-89-20, Comp. Science, Stanford University, 1989.

[Kautz and Selman, 1998] Henry Kautz and Bart Selman. Blackbox: a New Approach to the Application of Theorem Proving to Problem Solving. http://www.research.att.com/kautz, 1998.

[Koehler *et al.*, 1997] J. Koehler, B. Nebel, J. Hoffmann and Y. Dimopoulos. Extending Planning Graphs to an ADL Subset. In *Proceedings of the European Conference on Planning*, 1997, pages 273–285, 1997.

[Kvarnstrom and Doherty, 2000] J. Kvarnstrom and Patrick Doherty. TALplanner: A Temporal Logic Based Forward Chaining Planner. *Annals of Mathematics and Artificial Intelligence*. Accepted for publication.

[Levesque *et al.*, 1997] Hector Levesque, Ray Reiter, Y. Lespérance, F. Lin and R. Scherls. GOLOG: a Logic Programming Language for Dynamic Domains. *Journal of Logic Programming*, Special Issue on Actions, 31(1-3):59–83, 1997.

[Lifschitz, 1993] Vladimir Lifschitz. Circumscription. *Handbook of Logic in Artificial Intelligence and Logic Programming*, D. Gabbay and C.J. Hogger, Ed., volume 3: Non-monotonic Reasoning and Uncertain Reasoning. Oxford University Press, 1993.

[Lifschitz, 1995] Vladimir Lifschitz. Nested Abnormality Theories. *Artificial Intelligence*, 74:1262–1277, 1995.

[Lin, 1997] Fangzhen Lin. Applications of the Situation Calculus to Formalizing Control and Strategic Information: the Prolog Cut Operator. In *Proceedings of the Fifteen International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1412–1418, 1997.

[Lin, 1998] Fangzhen Lin. On Measuring Plan Quality (A Preliminary Report). In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning*, pages 224–232, 1998.

[McCarthy and Hayes, 1969] John McCarthy and Pat Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. *Machine Intelligence*, 4:463–502, 1969.

[McCarthy, 1959] John McCarthy. Programs with Common Sense. In *Mechanization of Thought Processes, Proceedings of the Symposium of the National Physics Laboratory*, pages 77–84, 1959.

[McCarthy, 1980] John McCarthy. Circumscription -a Form of Non-monotonic Reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[McCarthy, 1986] John McCarthy. Applications of Circumscription to Formalizing Common Sense Knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[McCarthy, 1997] Lecture Notes of CS323 Course on Formalization of Common Sense – Non-monotonic Reasoning. http://www-formal.stanford.edu/jmc/cs323/96/lecture-notes.ps

[Nguyen and Kambhampati, 2000] XuanLong Nguyen and Subbarao Kambhanpati. Extracting Effective and Admissible State-space Heuristics from the Planning Graph. In *Proceedings of the National Conference on Artificial Intelligence*, 2000. American Association for Artificial Intelligence.

[Pednault, 1989] E. Pednault. ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus. In *Proceeding of the International Conference on Principles of Knowledge Representation and Reasoning*, pages 324–332, 1989.

[Reiter, 1991] Ray Reiter. The Frame Problem in the Situation Calculus: a Simple Solution (Sometimes) and a Completeness Result for Goal Regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy,* pages 418–420, Academic Press, San Diego, CA, 1991.

[Reiter, 1993] Ray Reiter. Proving Properties of States in Situation Calculus. *Artificial Intelligence* 64:337–351, 1993.

[Reiter, 1999] Ray Reiter. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems.* In preparation. Draft available at http://www.cs.toronto.edu/cogrobo, 1999.

[Sierra, 1998] Josefina Sierra. Declarative Formalization of Strategies for Action Selection. In *Proceedings of the Seventh International Workshop on Non-monotonic Reasoning*, pages 21–29, 1998.

[Sierra, 2000a] Josefina Sierra. Declarative Formalization of Reasoning Strategies: a case study on heuristic nonlinear planning. *Annals of Mathematics and Artificial Intelligence*, to appear.

[Sierra, 2000b] Josefina Sierra. Declarative Formalization of Strategies for Action Selection: Applications to Planning. In *Proceedings of the European Workshop on Logics for Artificial Intelligence*, pages 133–147, 2000.

[Weld and Etzioni, 1994] Daniel Weld and Oren Etzioni. The First Law of Robotics (a Call to Arms). In *Proceedings of the National Conference on Artificial Intelligence*, pages 1042–1047, 1994. American Association for Artificial Intelligence.