Planning with Declarative Formalizations of Heuristics for Action Selection

Josefina Sierra-Santibáñez
Escuela Técnica Superior de Ingeniería Informática
Universidad Autónoma de Madrid
Email: Josefina.Sierra@iuam.es

Abstract

We propose a representation scheme for the declarative formalization of heuristics for action selection based on the situation calculus and circumscription. The formalism is applied to represent a number of heuristics for moving blocks in order to solve planning problems in the blocks world. An advice-taking scenario is presented to illustrate how the action selection strategy of a program can be refined by simple additions of better heuristics. Some examples are given in which the formalism is used to reason about the behaviors that may be caused by a number of strategies for action selection. Experiments showing how the use of declarative representations of heuristics for action selection allows improving the performance of state of the art planning systems are described.

1 Introduction

This paper proposes a representation scheme for the declarative formalization of heuristics for action selection based on the situation calculus [13] and circumscription [15].

The idea of representing strategies for action selection as sets of action selection rules is explored. An action selection rule [6] is an implication whose antecedent is a formula of the situation calculus, 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 following action selection rules 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 not in final position and cannot be moved to final position, it is better to move it to the table than anywhere else; (3) If a block is in final position, do not move it; (4) If a block is another block it ought to be above but it is not in final position (tower-deadlock), put it on the table:

\[-\text{Final}(x, s) \land \text{Final}(x, \text{Result}(\text{Move}(x, y), s)) \rightarrow \text{Good}(\text{Move}(x, y), s) \quad (1)\]
\[-\text{Final}(x, s) \land \neg \exists z \text{Final}(x, \text{Result}(\text{Move}(x, z), s)) \land y \neq T \rightarrow \text{Better}(\text{Move}(x, T), \text{Move}(x, y), s) \quad (2)\]
\[
\text{Final}(x, s) \rightarrow \text{Bad}(\text{Move}(x, y), s) \quad (3)
\]
\[
\text{Tower-d}(x, s) \rightarrow \text{Good}(\text{Move}(x, T), s) \quad (4)
\]

A consistent set of action selection rules (such as \( T_{S_4} = \{1, 2, 3, 4\} \)) defines a strategy for action selection.

\(^1\)The expressions Final(x, s) and Tower-d(x, s) are abbreviations for the formulas Holds(Final(x), s) and Holds(Tower-d(x), s). A block is in final position if it is on the table and it should be on the table, or if it is on a block it should be on and that block is in final position. See section 3 for a formal definition of the concepts of final position and tower-deadlock.
2 Nonmonotonic Interpretation

The formal semantics of a strategy for action selection $T_S$ is given by $\text{INT}(T_S)$ [20], the nested abnormality theory specified on the right hand side of formula 6. Nested abnormality theories [12] extend simple abnormality theories [17] by allowing the specification of nested applications of the circumscription operator [15]. $\text{INT}(T_S)$ characterizes the conditions under which an action is good or bad for a particular situation, by jumping to the conclusions that: (1) an action is "not good" unless the action selection rules in $T_S$ imply that it is good; and (2) an action is "not bad" unless the action selection rules in $T_S$, together with axiom 5, imply that it is bad. Axiom 5 asserts that an action is bad for a particular situation if there exists a better action for the same situation.

$$\text{Better}(a_1, a_2, s) \rightarrow \text{Bad}(a_0, s)$$

(5)

$$\text{INT}(T_S) \equiv \{ \text{Better, min Bad : 5, \{min Good : T_S\}} \}$$

(6)

Formally, this is achieved as follows. First, the predicate Good is circumscribed with respect to the conjunction of the universal closures of the axioms in $T_S$. Then, the predicate Bad is circumscribed with respect to the result of the circumscription of Good in $T_S$ and the universal closure of axiom 5. Better is allowed to vary because minimizing the extension of Bad may affect through axiom 5 the extension of Better.

This nonmonotonic interpretation of action selection strategies has both representation and computational advantages. It allows describing strategies: (1) succinctly, since it is not necessary to specify negative information (i.e. which actions are not good, not bad, or not better than others); (2) according to a least commitment policy, in which it is not necessary to assert that an action is good, bad, or better than other unless it is known for sure; and (3) incrementally, since it is possible to refine an action selection strategy by simple additions of better heuristics (i.e. consistent action selection rules that may become available later on). In these three cases, circumscription takes care of appropriately adapting its consequences to the lack of information or the availability of new relevant facts.

The following formal result\(^2\) establishes some conditions under which the interpretation $\text{INT}(T_S)$ of a strategy for action selection $T_S$ can be computed by predicate completion.

**Proposition 1** If every axiom of $T_S$ is a first order action selection rule such that its antecedent does not contain the predicates Good, Bad or Better, then $\text{INT}(T_S)$ is equivalent to the conjunction of the first order sentences 7 and 8 resulting from the application of a variant of Clark's completion algorithm [3] described below to $T_S$.

$$\forall a \forall s (\text{Good}(a, s) \leftrightarrow A_3^{\text{good}}(a, s))$$

(7)

$$\forall a \forall s (\text{Bad}(a, s) \leftrightarrow A_3^{\text{bad}}(a, s) \vee \exists a_1 a_2 (a = a_2 \wedge A_3^{\text{better}}(a_1, a_2, s)))$$

(8)

**Completion Algorithm** Let $T_S$ be a declarative formalization of a strategy for action selection. The axioms of $T_S$ are all of the form $a \rightarrow P(t_a, t_s)$, where $a$ is a first order formula which does not contain the predicates Good, Bad or Better, $t_a$ is a tuple of terms of the sort action, $t_s$ is a term of the sort situation, and $P$ is one of the predicates Good, Bad or Better.

**Step 1** Replace each rule of the form $a \rightarrow P(t_a, t_s)$ in $T_S$ by $a \wedge \bar{a} = t_a \wedge s = t_s \rightarrow P(\bar{a}, s)$, where $\bar{a}$ is a tuple of new variables of the sort action, and $s$ is a new variable of the sort situation.

**Step 2** Replace each rule $A_1(\bar{a}, s) \rightarrow P(\bar{a}, s)$ obtained in the previous step by $\exists \bar{x} A_1(\bar{a}, s) \rightarrow P(\bar{a}, s)$, where $\bar{x}$ are the free variables in the original rule.

**Step 3** For each $P$, replace all the rules of the form $A_2(\bar{a}, s) \rightarrow P(\bar{a}, s)$ obtained in step 2 by a single rule of the form $\forall \bar{x} A_2(\bar{a}, s) \rightarrow P(\bar{a}, s)$.

**Step 4** Replace the rule $A_3^{\text{good}}(a, s) \rightarrow \text{Good}(a, s)$ obtained in step 3 by $\forall a \forall s (\text{Good}(a, s) \leftrightarrow A_3^{\text{good}}(a, s))$.

**Step 5** Replace the rules $A_3^{\text{bad}}(a, s) \rightarrow \text{Bad}(a, s)$ and $A_3^{\text{better}}(a_1, a_2, s) \rightarrow \text{Better}(a_1, a_2, s)$ obtained in step 3 by a single rule\(^3\) of the form $\forall a \forall s (\text{Bad}(a, s) \leftrightarrow A_3^{\text{bad}}(a, s) \vee \exists a_1 a_2 (a = a_2 \wedge A_3^{\text{better}}(a_1, a_2, s)))$.

\(^2\)See [20] for a formal proof of proposition 1.

\(^3\)We assume the variables $a$, $a_1$ and $a_2$ of the sort action are distinct from each other.
For example, the nonmonotonic interpretation \( INT(T_{s_1}) \) of the action selection strategy \( T_{s_1} \), described by the four action selection rules presented in section 1, can be computed by the completion algorithm. We show only the result of the last step of the algorithm.

\[
\forall a \left( \text{Good}(a, s) \leftrightarrow \exists y \left( \text{Final}(x, \text{Result}(\text{Move}(x, y), s)) \land \neg \text{Final}(x, s) \land a = \text{Move}(x, y) \right) \lor \\
\exists t \left( \text{Tower-}a(t, x, s) \land a = \text{Move}(x, T) \right) \right) \\
\forall a \left( \text{Bad}(a, s) \leftrightarrow \exists y \left( \text{Final}(x, s) \land a = \text{Move}(x, y) \right) \lor \exists a_1 a_2 (a = a_2 \land \exists y (y \neq T \land \\
\neg \exists z \left( \text{Final}(x, \text{Result}(\text{Move}(x, z), s)) \land \neg \text{Final}(x, s) \land a_1 = \text{Move}(x, T) \land a_2 = \text{Move}(x, y) \right) ) \right) \\
\right)
\]

These two formulas characterize the conditions under which a move is good or bad for a particular situation according to the strategy for action selection \( T_{s_1} \).

3 Reasoning about Strategies

The interpretation of a strategy for action selection gives us a characterization of the conditions under which an action is good or bad for a particular situation. Let us assume that an agent is capable to determine whether these conditions hold for a particular situation. Then, it can infer from the interpretation of an action selection strategy what actions are good or bad for that particular situation, and use that information to decide what to do. For example, it can try to find a good action for each situation, and pick up a non-bad action if there were no good actions for that situation.

The action selection mechanism of such an agent can be described by the following axiom, that characterizes the set of selectable actions for a particular situation. \( \text{Prec} \) is a predicate that holds for an action \( a \) and a situation \( s \) if action \( a \) can be performed at situation \( s \).

\[
\text{Sel}(a, s) \leftrightarrow \text{Prec}(a, s) \land (\text{Good}(a, s) \lor (\neg \exists b \text{Good}(b, s) \land \neg \text{Bad}(a, s))) \\
\]

However, an agent may also want to reason about a strategy for action selection in order to decide whether it should be applied to solve a given problem. This would allow it to save time and resources by detecting incorrect strategies before putting them to practice.

The following axiom characterizes the situations that can be selected by applying a strategy for action selection to solve a given problem. \( S_0 \) denotes the initial situation of the problem. \( \text{Achieved} \) is a predicate that holds for a situation that satisfies the goal conditions.

\[
\text{Select}(s) \leftrightarrow s = S_0 \lor \exists s_1 \left( \text{Select}(s_1) \land \neg \text{Achieved}(s_1) \land \text{Sel}(a, s_1) \land s = \text{Result}(a, s_1) \right) \\
\]

These two axioms, together with the foundational axioms [18], give us a characterization of the actions and situations that are selectable according to a strategy for action selection in terms of a set of situation calculus formulas. We have assumed that the agent is capable to determine whether these formulas hold for a particular situation. One way in which it could do that is by using a theory of action. We present a simple theory of action that can be used for this purpose. But it should be clear that the ideas presented in this paper would work for other theories of action as well.

The theory of action used in this paper is based on a formalization of STRIPS [4] in the situation calculus described in [16]. Associated with each situation is a database of propositions describing the state associated with that situation. The predicate \( DB(p, s) \) asserts that propositional fluent \( p \) is in the database associated with situation \( s \). Each action is described by a precondition, an add list and a delete list, which are formally characterized by the following predicates: (1) \( \text{Prec}(a, s) \) is true provided action \( a \) can be performed in situation \( s \); (2) \( \text{Del}(p, a, s) \) is true if proposition \( p \) becomes false when action \( a \) is performed in situation \( s \); (3) \( \text{Add}(p, a, s) \) is true if proposition \( p \) becomes true when action \( a \) is performed in situation \( s \). The function \( \text{Result} \) maps a situation \( s \) and an action \( a \) into the situation that results when action \( a \) is performed in situation \( s \). When an action is considered, it is first determined whether its precondition is satisfied. If the precondition is met, then the sentences on the delete list are deleted from the database, and the sentences on the add list are added to it.

\[
DB(p, \text{Result}(a, s)) \leftrightarrow \text{Prec}(a, s) \land (\text{Add}(p, a, s) \lor DB(p, s) \land \neg \text{Del}(p, a, s)) \\
\]
In order to interpret action selection rules, such as axioms 1 to 4, in terms of the theory of action described above, we need to establish a connection between what holds at a situation and what is in the database associated with that situation. In this paper, we assume that the state associated with any situation can be described in terms of the truth values of a finite set of frame fluents [13] [10]. The rest of the fluents, called defined fluents, are described in terms of the frame fluents. The database associated with a situation determines the truth values of the frame fluents as follows: a frame fluent holds at a particular situation if and only if it is in the database associated with that situation.

\[ Frame(p) \rightarrow (\text{Holds}(p, s) \iff DB(p, s)) \] (12)

We assume uniqueness of names for every function symbol, and every pair of distinct function symbols. The expression \( s < s_1 \) means that \( s_1 \) can be reached from \( s \) by performing a nonempty sequence of executable actions; \( s \prec s_1 \) means that \( s_1 \) can be reached from \( s \) by performing a nonempty sequence of selectable actions. We introduce an axiom of induction for situations that allows us to prove that a property holds for all the situations, and constrains the domain of situations to those that can be reached from the initial situation by performing executable sequences of actions [18]. A situation is terminal with respect to an action selection strategy and a problem if it is selectable, but no action can be selected at that situation. Finally, the predicate achieved characterizes the situations that satisfy the goal conditions of a given problem.

\[
\begin{align*}
&\forall i \exists j (h(x) = h(y) \rightarrow x = y), \forall i \exists j (h(x) \neq g(y)) \\
&\forall s (s < s_1) \land \forall s, s_1 (s < \text{Result}(a, s_1) \iff \text{Prec}(a, s_1) \land s \leq s_1) \\
&\forall P(P(S_0) \land \forall s, a (P(s) \land \text{Prec}(a, s) \rightarrow P(\text{Result}(a, s)))) \rightarrow \forall s P(s) \\
&\text{Terminal}(s) \iff \text{Select}(s) \land \exists a \text{Sel}(a, s) \\
&\text{Achieved}(s) \iff \forall P(\text{Goal}(p) \rightarrow \text{Hold}(p, s))
\end{align*}
\] (13) (14) (15) (16) (17) (18)

For example, the problem shown in fig. 1 can be formalized as follows. The variables \( x, y \) and \( z \) range over blocks. The constants \( A, B, C, D, E, F \) and \( T \) (for Table) are of the sort block. The function symbol \( On \) maps a pair of blocks \( x \) and \( y \) into the propositional fluent \( On(x, y) \) describing the fact that block \( x \) is on block \( y \). The function symbol \( Move \) maps a pair of blocks \( x \) and \( y \) into the action \( Move(x, y) \) denoting the act of moving block \( x \) on top of block \( y \). We include domain closure axioms for blocks and actions. The initial and goal configurations are described by axioms 20 and 21. The precondition, delete and add lists of \( Move(x, y) \) are characterized by axioms 22, 23 and 24.

\[
\begin{align*}
&\forall x (x = A \lor x = B \lor x = C \lor x = D \lor x = E \lor x = F \lor x = T) \land \forall a \exists y (a = \text{Move}(x, y)) \\
&\text{DB}(p, s_0) \iff \exists y (\text{On}(x, y) \land ((x = A \land y = B) \lor (x = B \land y = T)) \\
&(x = C \land y = E) \lor (x = D \land y = T) \lor (x = E \land y = D) \lor (x = F \land y = T)) \\
&\text{Goal}(p) \iff \exists y (\text{On}(x, y) \land ((x = A \land y = C) \lor (x = B \land y = T) \\
&(x = C \land y = B) \lor (x = D \land y = T) \lor (x = E \land y = D) \lor (x = F \land y = T)) \\
&\text{Prec}(\text{Move}(x, y), s) \iff \neg \text{DB}(\text{On}(x, y), s) \land x \neq T \land x \neq y \land \forall z \neg \text{DB}(\text{On}(z, x), s) \\
&\neg y \neq T \rightarrow \forall z \neg \text{DB}(\text{On}(z, y), s) \\
&\text{Del}(p, \text{Move}(x, y), s) \iff \exists z (\text{On}(x, z) \land \text{DB}(p, s) \land z \neq y) \\
&\text{Add}(p, \text{Move}(x, y), s) \iff p = \text{On}(x, y)
\end{align*}
\] (19) (20) (21) (22) (23) (24)

Frame fluents are of the form \( On(x, y) \). We use a number of defined fluents, such as clear, final,

---

4The symbols \( h \) and \( g \) are meta-variables ranging over distinct function symbols; \( \vec{x} \) and \( \vec{y} \) denote tuples of variables.

5The predicate \( \text{Goal}(p) \) asserts that propositional fluent \( p \) is one of the goal conditions of the planning problem.
Figure 1: Behaviors associated with different strategies. Strategy-1 and strategy-2 may cause cyclic behaviors. Strategy-3 and strategy-4 are both correct. They may cause the behaviors indicated by their respective arrows. Notice that strategy-4 is better than strategy-3.

above\(^6\), and tower-deadlock

\[ Frame(p) \leftrightarrow \exists xy(p = On(x, y)) \] \hspace{1cm} (25)

\[ Holds(Clear(x), s) \leftrightarrow x = T \lor \neg \exists y Holds(On(y, x), s) \] \hspace{1cm} (26)

\[ Holds(Final(x), s) \leftrightarrow (Holds(On(x, T), s) \land Goal(On(x, T)) \lor \exists y(Holds(Final(y), s) \land Holds(On(x, y), s))) \] \hspace{1cm} (27)

\[ Holds(Above(x, y), s) \leftrightarrow Holds(On(x, y), s) \lor \exists z(Holds(On(x, z), s) \land Holds(Above(z, y), s)) \] \hspace{1cm} (28)

\[ Goal-Above(x, y) \leftrightarrow Goal(On(x, y)) \lor \exists z(Goal(On(x, z)) \land Goal-Above(z, y)) \] \hspace{1cm} (29)

\[ Holds(Tower-d(x), s) \leftrightarrow \neg Holds(Final(x), s) \land \exists y(y \neq T \land Holds(Above(x, y), s) \land Goal-Above(x, y)) \] \hspace{1cm} (30)

4 Advice Taking Scenario

We describe a scenario in which a program uses the nonmonotonic interpretation \( INT(T_S) \) of a strategy for action selection \( T_S \) (see formula 6) and the set of axioms \( T = \{0, \ldots, 30\} \) to reason about different strategies for action selection. The program starts with an empty strategy. An advisor suggests different heuristics [14] it could use to solve the problem in fig. 1.

\(^6\)If we assume uniqueness of names, a complete characterization of the extensions of the formulas \( Holds(On(x, y), s) \) and \( Goal(On(x, y)) \), an axiom of induction for situations, and that there is only a finite number of blocks [as we do], the definitions of \( Holds(Final(x), s) \), \( Holds(Above(x, y), s) \) and the predicate \( Goal-Above(x, y) \) provided here allow us to characterize the extensions of these formulas. The same holds for the definition of \( Select(s) \) presented before.
Initially, the advisor suggests the following heuristic: *If a block can be moved to final position, this should be done right away.* The program constructs strategy-1, \( T_{S1} = \{1\} \), which consists of a single action selection rule (axiom 1). Using \( INT(T_{S1}) \), the program can prove that if it applies strategy-1, its execution may not terminate by entering into a cycle (see fig. 1). In order to do so, it can use the following fact: a state-based\(^7\) strategy may cause a cycle (axiom 31) if there is a pair of selectable situations \( s_1, s_2 \) with the same associated state and such that \( s_1 < s_2 \) (i.e. \( s_2 \) can be reached from \( s_1 \) by a nonempty sequence of selectable actions).

\[
\text{Cycle}(s_1, s_2) \leftrightarrow \text{Select}(s_1) \land \text{Select}(s_2) \land \forall p (\text{Holds}(p, s_1) \leftrightarrow \text{Holds}(p, s_2)) \land s_1 < s_2 \quad (31)
\]

In particular, the program can prove the following.

\[
T, 31, INT(T_{S1}) \vdash \text{Cycle}(S_0, \text{Result}(\{\text{Move}(C, T), \text{Move}(C, E)\}, S_0))
\]

The program asks for more advice, instead of applying strategy-1 to solve the problem. The advisor proposes a second heuristic: *if a block is not in final position and cannot be moved to final position, it is better to move it to the table than anywhere else.* The program constructs strategy-2, \( T_{S2} = \{1, 2\} \). Using \( INT(T_{S2}) \), it can prove that strategy-2 may cause a cycle (see fig. 1).

\[
T, 31, INT(T_{S2}) \vdash \text{Cycle}(\text{Result}(\{\text{Move}(C, T), \text{Result}(\{\text{Move}(C, T), \text{Move}(E, T), \text{Move}(E, D)\}), S_0)\}
\]

The advisor suggests a third heuristic: *If a block is in final position, do not move it.* The program constructs strategy-3, \( T_{S3} = \{1, \ldots, 3\} \). The set of situations that are selectable according to strategy-3 is finite (see fig. 1). The program can prove that \( T_{S3} \) is correct, because its terminal\(^8\) situations achieve the goal conditions\(^9\).

\[
T, INT(T_{S3}) \vdash \forall s (\text{Terminal}(s) \rightarrow \text{Achieved}(s)) \land \forall s (\text{Terminal}(s) \leftrightarrow \text{Result}(\{\text{Move}(C, T), \text{Move}(A, T), \text{Move}(C, B), \text{Move}(A, C)\}, S_0) \land
\]

\[
s = \text{Result}(\{\text{Move}(A, T), \text{Move}(C, B), \text{Move}(A, C)\}, S_0))
\]

The advisor suggests a fourth heuristic: *If there is a block that is above a block it ought to be above, but it is not in final position (tower-deadlock), put it on the table.* The program constructs strategy-4, \( T_{S4} = \{1, \ldots, 4\} \). Notice that the set of selectable situations gets smaller as a strategy is refined (see fig. 1). The program can prove that strategy-4 is correct, and that it is better than strategy-3, since it always solves the problem using a smaller or equal number of actions.

\[
T, INT(T_{S4}) \vdash \forall s (\text{Terminal}(s) \rightarrow \text{Achieved}(s)) \land \forall s (\text{Terminal}(s) \leftrightarrow s = \text{Result}(\{\text{Move}(A, T), \text{Move}(C, B), \text{Move}(A, C)\}, S_0))
\]

The advisor suggests a fifth heuristic: *If a block is on the table but not in final position, do not move anything on that block.* \( On(x, T, s) \) abbreviates \( \text{Holds}(On(x, T), s) \).

\[
On(x, T, s) \land \neg \text{Final}(x, s) \rightarrow \text{Bad}(\text{Move}(z, x), s)
\]

\[
(32)
\]

The program constructs \( T_{S5} = \{1, \ldots, 4, 32\} \). It can prove that the sets of selectable situations according to \( T_{S4} \) and \( T_{S5} \) are identical. Therefore, axiom 32 is redundant with its current strategy \( T_{S4} \).

---

\(^7\)A strategy is state-based if whether an action is good or bad for a particular situation depends only on what holds at that situation (i.e. the state associated with that situation). All the strategies considered in the paper are state-based.

\(^8\)A situation is terminal with respect to a strategy if it is selectable, but no action can be selected at that situation.

\(^9\)Axiom 36 of induction for situations is needed here to prove that \( \neg \text{Terminal}(s) \) holds for all the situations not mentioned in the theorem.
The advisor suggests a sixth heuristic: If a block is in tower-deadlock position, it is better to move it on top of a clear block that is in final position and should be clear on the goal configuration than anywhere else.

\[
\text{Holds}(\text{Tower-}\text{d}(x, s) \wedge \text{Holds}(\text{Clear}(z), s) \wedge \text{Holds}(\text{Final}(z), s) \wedge \\
\neg \exists y \text{Goal}(\text{On}(y, z)) \wedge z \neq w \rightarrow \text{Better}(\text{Move}(x, z), \text{Move}(x, w), s)
\]  

(33)

We use axiom 34 to detect inconsistencies in the behaviors that may be caused by a strategy. Formulas 31 and 34 are examples of verification axioms that can be added to \( T \) in order to reason about strategies.

\[
\text{Bad}(a, s) \rightarrow \neg \text{Good}(a, s)
\]  

(34)

The program constructs \( T_{S6} = \{1, \ldots, 4, 33\} \). Using axiom 34, \( T \) and \( \text{INT}(T_{S6}) \), the program can prove the following contradiction\(^{10}\). This means that axiom 33 is inconsistent with its current strategy \( T_{S4} \).

\[
T, 34, \text{INT}(T_{S6}) \vdash \text{Good}(\text{Move}(A, T), S_i) \land \neg \text{Good}(\text{Move}(A, T), S_i)
\]

5 Related Work

In [19], 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 \( \text{goal}(s) \), which is true if situation \( s \) satisfies the conditions of the goal for which a plan is sought; (2) a set of \( \text{action precondition and successor state axioms} \) for the primitive actions of the domain; and (3) a predicate \( \text{badSituation}(s) \), which is true if situation \( s \) is considered to be a bad situation for the planner to consider. The planner is implemented in GOL G [9], and it has been extended to deal with concurrent actions and incomplete initial situations [5].

The representation scheme proposed in this paper is more expressive than that used in [19], in the sense that it allows the representation of positive heuristics (the predicate \( \text{good} \) tells a system what to do), and heuristics that establish preferences among actions (the predicate \( \text{better} \) establishes a partial order among actions). The predicate \( \text{badSituation}(s) \) 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 as the predicate \( \text{good} \) does in our formalization.

The heuristics for the blocks world used in [5] prune approximately the same set of situations as action selection rules 1 to 3. In particular, the definition of \( \text{good-tower} \) is equivalent to our concept of \( \text{final position} \). However, the heuristics in [5] do not consider the concept of \( \text{tower-deadlock position} \), and therefore they cannot be used to 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 action selection rule 4.

In [2], 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 \( \text{goal} \) and \( \text{temporal modalities} \) (until, always, eventually, and next) are used to assert properties of world sequences. A search control formula describing the search control strategy to be used by the program 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. Interesting experiments in which TLPLAN outperforms state of the art planners, such as BlackBox [7] and IPP [8]

\(^{10}\)Axiom 4 implies that \( \text{Move}(A, T) \) is \( \text{good} \) in the initial situation \( A \) is in tower-deadlock position. Axioms 5 and 33 imply that \( \text{Move}(A, T) \) is \( \text{bad} \) for the same situation.
in various test domains using search control formulas are described. Blackbox and IPP are both state of
the art planning systems. They were the best performers in the AIPS'98 planning competition [1].

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 selectable plans
(rather than actions) must satisfy. The last search control formula used for the blocks world in [2] prunes
approximately the same set of situations as the first three action selection rules of $T_{S_1}$ (the action selection
strategy proposed in section 1 of this paper). In particular, their definition of good-tower is equivalent to
our concept of final position.

An advantage of our proposal is the availability of a formal model of the planner which allows limited
forms of meta-reasoning, such as determining the correctness, redundancy, inconsistency or quality of
different strategies for action selection. This is an important feature that allows the planner to reject
incorrect strategies, and to provide its users with feed back on how to improve their strategies. This is
not possible in TLPLAN, because it does not have a formal description of its own mechanism for action
selection, which allows it to reason about the consequences of adopting a particular strategy.

6 Experiments

We have implemented a heuristic forward chaining planner which can use declarative representations of
planning domains and strategies for action selection in Prolog. The action selection strategy used by
the planner can be improved by simple additions of better heuristics, as illustrated in the advice taking
scenario of section 4. However, the meta-reasoning capabilities described in the scenario have not been
implemented yet.

The planner has been applied to solve some blocks world problems using $T_{S_1}$, the strategy for action
selection described in section 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 our planner, the number of situations (nodes) explored, and the time in
milliseconds spent on planning.

We have compared our results with those obtained from running the same problems in 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.

Comparing the numbers in the columns Steps and Steps TLPlan, it can be observed that TLPlan
cannot find optimal plans (i.e., with a minimum number of steps) for 10 of the 16 problems posed. Our
planner obtains optimal plans for the 16 problems. As far as planning time is concerned, our planner is
faster than TLPlan. The only exceptions are the problems of sizes 15 and 19. However, the numbers of
steps of the plans found by TLPlan are very far from optimality, 18 and 25 steps versus 14 and 18 steps
for the optimal plans. These two problems illustrate the necessity of looking ahead more than one step
in order to find optimal plans for blocks world problems. In fact, it is possible to find optimal solutions
for both problems without backtracking, if we add a new action selection rule to $T_{S_1}$ that prefers moving
a block to the table if the block under it can be moved to final position in the situation resulting from
moving it to the table.
Table 1. Problems of 25 blocks.

<table>
<thead>
<tr>
<th>Prob</th>
<th>Final</th>
<th>TD</th>
<th>Steps</th>
<th>Nodes</th>
<th>Time</th>
<th>Steps TLPlan</th>
<th>Nodes TLPlan</th>
<th>Time TLPlan</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>26</td>
<td>26</td>
<td>0</td>
<td>26</td>
<td>26</td>
<td>58</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>11</td>
<td>36</td>
<td>36</td>
<td>0</td>
<td>38</td>
<td>38</td>
<td>91</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>23</td>
<td>23</td>
<td>0</td>
<td>25</td>
<td>25</td>
<td>58</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>0</td>
<td>18</td>
<td>18</td>
<td>0</td>
<td>20</td>
<td>20</td>
<td>52</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>2</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>20</td>
<td>46</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>4</td>
<td>28</td>
<td>28</td>
<td>0</td>
<td>30</td>
<td>30</td>
<td>68</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>6</td>
<td>30</td>
<td>30</td>
<td>0</td>
<td>37</td>
<td>37</td>
<td>91</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>13</td>
<td>37</td>
<td>37</td>
<td>0</td>
<td>37</td>
<td>37</td>
<td>85</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>3</td>
<td>27</td>
<td>27</td>
<td>0</td>
<td>29</td>
<td>29</td>
<td>68</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>7</td>
<td>31</td>
<td>31</td>
<td>50</td>
<td>32</td>
<td>32</td>
<td>84</td>
</tr>
</tbody>
</table>

Table 2. Problems of different sizes.

<table>
<thead>
<tr>
<th>Size</th>
<th>Final</th>
<th>TD</th>
<th>Steps</th>
<th>Nodes</th>
<th>Time</th>
<th>Steps TLPlan</th>
<th>Nodes TLPlan</th>
<th>Time TLPlan</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>11</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>3</td>
<td>15</td>
<td>15</td>
<td>0</td>
<td>15</td>
<td>15</td>
<td>19</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>0</td>
<td>14</td>
<td>274</td>
<td>320</td>
<td>18</td>
<td>18</td>
<td>26</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td>0</td>
<td>18</td>
<td>3583</td>
<td>5610</td>
<td>25</td>
<td>25</td>
<td>47</td>
</tr>
<tr>
<td>25</td>
<td>5</td>
<td>1</td>
<td>22</td>
<td>29</td>
<td>50</td>
<td>22</td>
<td>22</td>
<td>51</td>
</tr>
<tr>
<td>50</td>
<td>24</td>
<td>0</td>
<td>26</td>
<td>26</td>
<td>0</td>
<td>26</td>
<td>26</td>
<td>158</td>
</tr>
</tbody>
</table>

7 Conclusions

We have proposed a representation scheme for the declarative formalization of heuristics for action selection based on the situation calculus and circumcision. The formalism has been applied to represent a number of heuristics for moving blocks in order to solve planning problems in the blocks world. An advice taking scenario has been presented to illustrate how the action selection strategy of a program can be refined by simple additions of better heuristics. Some examples in which the formalism is used to reason about the behaviors that may be caused by a number of strategies for action selection have been described.

Finally, we have implemented a heuristic forward chaining planner that can take advice in the form of declarative formalizations of heuristics for action selection, and we have described some experiments showing how the use of declarative representations of strategies for action selection allows improving the performance of state-of-the-art planning systems.

Future work should address the development of meta-reasoning techniques that allow proving theorems about the behavior of strategies for action selection on classes of problems, rather than problem instances. The automatic generation of heuristics for action selection is a very interesting and difficult problem as well. Issues such as the safeness and postenable of certain actions may be useful for the development of heuristics, and can sometimes be extracted automatically from the description of a planning domain.

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


