Declarative Formalization of Heuristics

Josefina Sierra-Santibáñez
Computer Science Department
Stanford University
Stanford, CA 94305
E-mail: jsierra@cs.stanford.edu

Abstract

We propose a representation scheme for the declarative formalization of heuristics based on the situation calculus and circumscription. The formalism is applied to represent and reason about heuristics for the blocks world. It is demonstrated that the particular use of circumscription proposed allows reasoning about the behavior of declarative formalizations of heuristics of the sort described in this paper. Finally, an *advice taking* scenario is presented to illustrate *elaboration tolerance* and potential applications to implement interesting *reflective* behavior.

1 Introduction

Strategic knowledge has traditionally been specified using procedural programming languages or dynamic logic [5] [8]. This paper proposes a representation scheme for the declarative formalization of heuristics based on the *situation calculus* [22] and *circumscription* [14] [16].

The idea of representing *heuristics* as sets of action selection rules [4] is explored. An *action selection rule* is an implication whose antecedent is a formula of the situation calculus, and whose consequent may take one of the following forms: $Good(a, s_g, s)$, $Bad(a, s_g, s)$ or $Better(a, a_2, s_g, s)$. Action selection rules are interpreted as follows: if the conditions of the antecedent hold, then performing action a at situation s is *good, bad* or *better* than performing action a_2 for the purpose of achieving the goal described by situation s_g [1].

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 put it on the table than anywhere else; (3) If a block is in final position*[2]*, do not move it; (4) If there is a block that is above a block it ought to be above in the goal configuration but it is not in final position (tower-deadlock), put it on the table.* A *consistent* set of action selection rules defines a particular *strategy.*

(1) $\neg Holds(Final(x,S_g),s) \wedge Holds(Final(x,S_g), Result(Move(x,y),s)) \rightarrow Good(Move(x,y),S_g,s)$

(2) $\neg Holds(Final(x,S_g),s) \wedge \neg\exists z Holds(Final(x,S_g), Result(Move(x,z),s)) \wedge y \neq Table \rightarrow$
$Better(Move(x,Table), Move(x,y), S_g, s)$

(3) $Holds(Final(x,S_g),s) \rightarrow Bad(Move(x,y),S_g,s)$

(4) $Holds(Tower\text{-}deadlock(x,S_g),s) \rightarrow Good(Move(x,Table),S_g,s)$

Axioms 5 and 6 specify how a set of action selection rules describing a particular strategy should be interpreted in terms of action selection. The predicate *Better* establishes a partial order among a set of actions with respect to a given goal and a particular situation. In order to select always the best possible action, axiom 5 assumes that an action is bad for a given goal and a particular situation if there is a better action for the same goal and situation.

(5) $Better(a_1, a_2, S_g, s) \rightarrow Bad(a_2, S_g, s)$

[1]The situational argument s_g in the predicates *Good, Bad* and *Better* allows reasoning about multiple goals. For example, by substituting the variable s_g by two different constants S_{g_1} and S_{g_2} we can express the fact that performing action a at situation s is good for the purpose of achieving the goal described by situation S_{g_1} but bad for achieving the goal described by situation S_{g_2}. In the rest of the paper, we assume that there is a single goal situation represented by the constant S_g.

[2]The concepts of *final* position and *tower-deadlock* will be defined formally later on.

Axiom 6 characterizes the situations that can be selected according to a particular strategy (such as that described by axioms 1 to 4). The extension of the fluent[3] *selectable* characterizes the behavior of a program that uses such a strategy by describing the sequences of actions that can be taken by that program[4].

$$(6)\quad Selectable(s) \leftrightarrow s = S_0 \vee \exists s_1 a(Selectable(s_1) \wedge \neg Achieved(S_g, s_1) \wedge Prec(a, s_1) \wedge s = Result(a, s_1) \wedge (Good(a, S_g, s_1) \vee (\neg \exists b Good(b, S_g, s_1) \wedge \neg Bad(a, S_g, s_1))))$$

A situation is *selectable* iff: (1) it is the initial situation; or (2) it is the result of performing a good action in a selectable situation at which the goal has not been achieved[5]; or (3) it is the result of performing a non-bad action in a selectable situation at which the goal has not been achieved and for which there are no good actions.

The declarative formalization of a strategy as a set of action selection rules is interpreted non-monotonically. Circumscription is applied to minimize the extensions of the predicates *good* and *bad*, i.e., to jump to the conclusion that an action is *"not good"* or *"not bad"* unless it can be deduced from the set of axioms describing the strategy that it is so. The extension of the fluent selectable (defined by axiom 6) depends on conclusions that need[6] to be derived from such a minimization, namely the facts that an action can be proved to be "not good" or "not bad" for a given goal and situation. Therefore, the *projection of a strategy* (i.e., the set of situations that can be selected according to it) depends non-monotonically on the declarative formalization of such a strategy.

This particular use of circumscription has some representational advantages. It allows us to describe strategies: (1) *succinctly*, since negative information (i.e., which actions are not good, not bad or not better than others) need not be specified; (2) according to a *least commitment* strategy, in which it is not necessary to state that an action is good, bad or better than another unless it is known for sure; and (3) *incrementally*, because the application of circumscription is designed so in such a way that the conclusions about the selectability of different situations adapt automatically to the addition of consistent heuristics that may become available later on.

In the following sections, we show how the ideas outlined above can be applied to formalize and reason about heuristics for the *blocks world*. The strategies formalized in the paper describe different algorithms for solving planning problems in the *elementary blocks world* domain [3]. Section 2 summarizes a very elegant and simple solution to the problem of reasoning about action described in [15]. Section 3 proposes a *nested abnormality theory* [11] that characterizes the behavior of the strategy described by axioms 1 to 4. This theory is generalized into a class of nested abnormality theories that apply circumscription in a particular way that allow reasoning about the behavior of declarative formalizations of strategies of the sort described in this paper. Finally, an *advice taking* [13] scenario is presented in order to illustrate potential applications.

2 A Simple Theory of Action

In [15], John McCarthy proposes a very simple formalization of STRIPS [2] in the *situation calculus*. The formalization is as follows. STRIPS is a planning system that uses a database of logical formulas to represent information about a state. Each action has a *precondition*, an *add list*, and a *delete list*. 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.

We will use variables of the following sorts: for situations (s, s_1, ...), for actions (a, a_1, ...), and for propositional fluents (p, p_1, ...). Associated with each situation is a database of propositions[7], and this gives us the wff *DB(p,s)* asserting that p is in the database associated with s. The function *Result* maps a situation s and an action a into the situation that results when action a is performed in situation s.

STRIPS is characterized by three predicates: (1) *Prec(a,s)* is true provided action a can be performed in situation s; (2) *Delete(p,a,s)* is true if proposition p is to be deleted when action a is performed in situation s;

[3]A *fluent* is a function defined on situations [22]. $Selectable(s)$ is a *propositional fluent*, that is, a fluent whose possible values are *true* or *false*.

[4]If we assume the existence of an initial situation S_0, the problem of determining the sequences of actions that can be selected according to a particular strategy is equivalent to the problem of determining the situations that are selectable for that strategy.

[5]We define $Achieved(s_g, s)$ and $Prec(a, s)$ formally later on. A situation s *achieves* the goal described by another situation s_g if all the conditions (propositional fluents) that *hold* at s_g *hold* at s as well. $Prec(a, s)$ is true if action a can be performed at situation s.

[6]Notice that action selection rules do not allow negations of the predicates *good*, *bad* and *better* on their consequents. Therefore, it is not possible to infer negative information about these predicates by classical deduction.

[7]In the paper, we use the expression *"propositional fluent"* and the word *"proposition"* in order to refer to the corresponding reified formula.

(3) *Add(p,a,s)* is true if proposition p is to be added when action a is performed in situation s. STRIPS has the single axiom

$$(7)\quad DB(p, Result(a,s)) \leftrightarrow (Prec(a,s) \wedge (Add(p,a,s) \vee (DB(p,s) \wedge \neg Delete(p,a,s)))) \vee (\neg Prec(a,s) \wedge DB(p,s))$$

The following example [15] illustrates how this formalization of STRIPS can be used to reason about action in the blocks world. The variables x, y and z range over blocks. The constants for blocks are A, B, C, D, E, F, and *Table*. The one kind of propositional fluent is *On(x,y)* describing the fact that block x is on block y. The one kind of action is *Move(x,y)* denoting the act of moving block x on top of block y. We assume uniqueness of names for every function symbol, and every pair of distinct function symbols[8]. The initial situation S_0 is described by axiom 9. The *precondition, delete* and *add lists* of *Move(x,y)* are characterized by axioms 10, 11 and 12, respectively.

$$(8)\quad h(\vec{x}) \neq g(\vec{y}) \wedge (h(\vec{x}) = h(\vec{y}) \rightarrow \vec{x} = \vec{y})$$

$$(9)\quad DB(p,S_0) \leftrightarrow \exists xy(p = On(x,y) \wedge ((x = A \wedge y = B) \vee (x = B \wedge y = Table) \vee (x = C \wedge y = E) \vee (x = D \wedge y = Table) \vee (x = E \wedge y = D) \vee (x = F \wedge y = Table)))$$

$$(10)\quad Prec(a,s) \leftrightarrow \exists xy(a = Move(x,y) \wedge \neg DB(On(x,y),s) \wedge x \neq Table \wedge x \neq y \wedge (y \neq Table \rightarrow \forall z \neg DB(On(z,y),s)) \wedge \forall z \neg DB(On(z,x),s))$$

$$(11)\quad Delete(p,a,s) \leftrightarrow \exists xyz(p = On(x,z) \wedge a = Move(x,y) \wedge z \neq y)$$

$$(12)\quad Add(p,a,s) \leftrightarrow \exists xy(p = On(x,y) \wedge a = Move(x,y))$$

For example, using axioms 7 to 12 we can prove that[9].

$$DB(p, Result(\{Move(A,Table), Move(C,B), Move(A,C)\}, S_0)) \leftrightarrow \exists xy(p = On(x,y) \wedge ((x = A \wedge y = C) \vee (x = B \wedge y = Table) \vee (x = C \wedge y = B) \vee (x = D \wedge y = Table) \vee (x = E \wedge y = D) \vee (x = F \wedge y = Table)))$$

The atomic formula $Holds(p,s)$ asserts that the value of p at situation s is true. The propositional fluents that are included in the database of associated with a situation s (i.e., the fluents of the form $On(x,y)$ for $x,y \in \{A,B,C,D,E,F,Table\}$) play the role of a *coordinate frame,* in the sense that specific configurations of blocks[10] can be described by combinations of values of these *frame fluents* [22] [9].

$$(13)\quad Frame(p) \leftrightarrow \exists wz(p = On(w,z) \wedge (\bigvee_{c_1,c_2 \in \{A,B,C,D,E,F,Table\}} (w = c_1 \wedge z = c_2)))$$

In order to interpret action selection rules, such as axioms 1 to 4, in terms of the theory of action described by axioms 7 to 13, we need to establish a connection between what *holds* at a situation, and what is in the *database* associated with that situation. The following axiom does so by asserting that a frame fluent holds at a particular situation, if and only if it is in the database associated with that situation.

$$(14)\quad Frame(p) \rightarrow (Holds(p,s) \leftrightarrow DB(p,s))$$

The expression $s < s_1$ means that s_1 can be reached [23] from s by executing a non-empty sequence of actions ($s \leq s_1$ is an abbreviation for $s < s_1 \vee s = s_1$) [11]. Axiom 16 introduces domain closure assumptions for blocks, actions and situations. It restricts the domain of situations to those that can be reached from the initial

[8]The symbols h and g are meta-variables ranging over distinct function symbols; the expressions $\vec{x}$ and $\vec{y}$ represent tuples of variables.

[9]We use the following notation to abbreviate the description of situations $Result(\{\},s) = s$, and $Result(\{a|l\},s) = Result(l, Result(a,s))$, where l is a sequence of actions (i.e., sequences of actions are applied from left to right).

[10]The symbols c_1 and c_2 are meta-variables ranging over block constants.

[11]The distinction between $Achieved(S_g,s)$ (axiom 23) and *reachable* $\leq$ (axiom 15) is crucial for understanding the role of the goal situation S_g in the formalization. The fact that axiom 15 implies that the goal situation S_g is not *reachable* from the initial situation S_0 does not imply that the planning problem is not solvable. When the program selects an action (see axiom 6 defining the fluent *selectable*), it tries to find a situation *reachable* from the initial situation at which the goal is *achieved.* That is, a situation that satisfies the conditions imposed on the goal situation. In this sense, we could say that the role of the goal situation S_g is purely descriptive, as far as this paper is concerned.

situation S_0 or from the goal situation S_g. Finally, axiom 17 of induction allows us to prove that a property holds for all the situations that can be reached from a given situation.

(15) $$\forall s(\neg s < S_0 \wedge \neg s < S_g) \wedge \forall a s s_1(s < Result(a, s_1) \leftrightarrow Prec(a, s_1) \wedge s \leq s_1)$$

(16) $$\forall x(x = A \vee x = B \vee x = C \vee x = D \vee x = E \vee x = F \vee x = Table) \wedge \\ \forall a(\bigvee_{c_1, c_2 \in \{A,B,C,D,E,F,Table\}} a = Move(c_1, c_2)) \wedge \forall s(S_0 \leq s \vee S_g \leq s)$$

(17) $$\forall P(P(s) \wedge \forall s_1 a(s \leq s_1 \wedge P(s_1) \wedge Prec(a, s_1) \rightarrow P(Result(a, s_1))) \rightarrow \forall s_2(s \leq s_2 \rightarrow P(s_2)))$$

We also use a number of *derived fluents*, such as *clear, final, above, tower-deadlock, terminal,* and *achieved*, which are partially[12] defined in terms of the frame fluents.

(18) $$Holds(Clear(x), s) \leftrightarrow x = Table \vee \neg \exists y Holds(On(y, x), s)$$

(19) $$Holds(Final(x, S_g), s) \leftrightarrow (Holds(On(x, Table), s) \wedge Holds(On(x, Table), S_g)) \vee \\ \exists y(Holds(Final(y, S_g), s) \wedge Holds(On(x, y), s) \wedge Holds(On(x, y), S_g))$$

(20) $$Holds(Above(x, y), s) \leftrightarrow Holds(On(x, y), s) \vee \exists z(Holds(On(x, z), s) \wedge Holds(Above(z, y), s))$$

(21) $$Holds(Tower\text{-}deadlock(x, S_g), s) \leftrightarrow \neg Holds(Final(x, S_g), s) \wedge \\ \exists y(y \neq Table \wedge Holds(Above(x, y), s) \wedge Holds(Above(x, y), S_g))$$

(22) $$Terminal(s) \leftrightarrow Selectable(s) \wedge \neg \exists a Selectable(Result(a, s))$$

(23) $$Achieved(S_g, s) \leftrightarrow \forall p(DB(p, s) \leftrightarrow DB(p, S_g))$$

Axiom 24 describes the configuration of the goal situation S_g. In general, a problem will be described by a set of conditions on some initial and goal situations. The particular problem described by axioms 9 and 24 characterizes completely the state of the initial and goal situations, but this needs not be the case. Specifying a set of constrains on initial and goal situations allows defining classes of problems, instead of particular instances. Such constraints can be used then to reason about the behavior of different strategies on classes of problems.

(24) $$DB(p, S_g) \leftrightarrow \exists xy(p = On(x, y) \wedge ((x = A \wedge y = C) \vee (x = B \wedge y = Table) \vee \\ (x = C \wedge y = B) \vee (x = D \wedge y = Table) \vee (x = E \wedge y = D) \vee (x = F \wedge y = Table)))$$

3 Reasoning about Heuristics

The formulas presented so far allow us to prove that some actions are *good, bad* or *better* than others for a given goal and a particular situation. But we aren't still able to decide which situations are selectable according to a particular strategy. Action selection rules do not give us complete information. They don't tell us which actions are "not good", "not bad" or "not better" than others. In order to interpret them in terms of action selection (using axiom 6), we need to be able to jump to the conclusion that an action is "not good" or "not bad" unless the heuristics known so far (axioms 1 to 4) imply that it is so. This incompleteness of our formalization is also one of its main advantages, because it allows us to refine the problem solving strategy of a program by simple additions of better heuristics. We illustrate this idea with an *advice taking* scenario in section 4.

Formula 25 describes a *nested abnormality theory* that characterizes the behavior of the strategy described by axioms 1 to 4 when it is applied to solve the problem described by axioms 9 and 24. That is, it allows us to determine what situations can be selected according to the strategy. Let Σ be the conjunction of the universal closures of the formulas 6 to 24.

(25) $\Sigma,\ \{Better,\ min\ Bad:\ 5,\ \{min\ Good:\ 1, \ldots, 4\}\}$

The expression 25 describes a nested abnormality theory in which circumscription is applied in the following way. The predicate *Good* is circumscribed with respect to the universal closures of the axioms defining the strategy of the program (axioms 1 to 4). The predicate *Bad* is circumscribed with respect to the result of the

[12] Axioms 19 and 20 are not explicit definitions of *final* and *above*, because these symbols occur both on the left and right hand sides. But these formulas are strong enough for deriving both positive and negative ground instances of *Holds(Above(x,y),s)* and *Holds(Final(x,S_g),s)* from the positive and negative ground instances of *Holds(On(x,y),s)* we can derive from axioms 7 to 17. [1] points out that it is possible to define *on* in terms of *beneath* ($beneath(y, x) \equiv above(x, y)$), but it is not possible to fully define *beneath* in terms of *on* in a first order theory.

circumscription described above and the universal closure of axiom 5, which contributes to the definition of *Bad* with positive instances of *Better*. We need to let *Better* vary, because minimizing the extension of *Bad* may affect (through axiom 5) the extension of *Better*. The latter circumscription, which characterizes the extensions of the predicates *Good* and *Bad*, is conjoined with Σ, which describes the theory of action (axioms 7 to 23), the specific problem reasoned about (axioms 9 and 24), and the mechanism of action selection (axiom 6).

Mathematically, circumscription is defined as a syntactic transformation of logical formulas. It transforms an axiom set A into a stronger axiom set A^*, such that the models of A^* are precisely the minimal models of A. Circumscription is in fact a family of syntactic transformations, because several minimality conditions can be used in conjunction with the same axiom set A. The expression $CIRC(A; P; Z)$ is defined by the following second order formula.

$$A(P, Z) \wedge \neg\exists pz(A(p, z) \wedge p < P)$$

The models of $CIRC(A; P; Z)$ are the models of A in which the extent of P cannot be made smaller at the price of changing the interpretation of the constants Z. *Nested abnormality theories* (NAT's) are theories in which the circumscription operator can be applied to a subset of the axioms, and whose axioms may have a nested structure with each level corresponding to a further application of the circumscription operator.

In general, it is difficult to use the definition of circumscription directly for proving properties about the circumscribed predicates, because it requires theorem proving in second order logic. Sometimes, computational methods can be applied to simplify circumscription formulas (see [10]). Theorem 1 demonstrates that this is the case for the nested abnormality theory described by formula 25. In particular, it shows that we can rewrite the result of the circumscription formulas in 25 as a conjunction of two first order logic formulas which have a particularly simple structure, each of them is an explicit definition of the circumscribed predicates *good* and *bad*.

Theorem 1 The nested abnormality theory described by 25 is equivalent to the second order logic theory whose axioms are Σ, plus the universal closures of formulas 26 and 27.

$$(26)\quad Good(Move(x,y), S_g, s) \leftrightarrow (Holds(Tower\text{-}deadlock(x, S_g), s) \wedge y = Table) \vee \\ (\neg Holds(Final(x, S_g), s) \wedge Holds(Final(x, S_g), Result(Move(x,y), s)))$$

$$(27)\quad Bad(Move(x,y), S_g, s) \leftrightarrow Holds(Final(x, S_g), s) \vee (y \neq Table \wedge \\ \neg Holds(Final(x, S_g), s) \wedge \neg\exists z Holds(Final(x, S_g), Result(Move(x,z), s)))$$

Proof The characterization of the semantics of NAT's in terms of second order logic theories including circumscription formulas and proposition 1 in [11] allow us to prove the equivalence between the following axiom sets[13].

$$\Sigma,\ \{Better,\ min\ Bad:\ 5,\ \{min\ Good:\ 1, \ldots, 4\}\} \equiv \\ \Sigma,\ CIRC(5',\ CIRC(1', \ldots, 4';\ Good);\ Bad;\ Better)$$

Now, we use several rules for computing circumscription described in [10]. The first equivalence below can be proved using formula (19) and proposition 2 in [10]. The second equivalence uses formula (19) and proposition 3 in that paper.

$$\Sigma,\ CIRC(5',\ CIRC(1', \ldots, 4';\ Good);\ Bad;\ Better) \equiv \\ \Sigma,\ CIRC(5', 3', 2', 26';\ Bad;\ Better) \equiv \\ \Sigma,\ 26',\ CIRC(3',\ \exists better(5' \wedge 2');\ Bad)$$

Using the equivalence (27) in section 3.2 of [10], we can prove that $\exists better(5' \wedge 2')(better)$ is equivalent to the following formula, which does not depend on *better*.

$$(28)\quad \neg Holds(Final(x, S_g), s) \wedge \neg\exists z Holds(Final(x, S_g), Result(Move(x,z), s)) \wedge y \neq Table \rightarrow \\ Bad(Move(x,y), S_g, s)$$

[13] In the following, we denote the universal closure of a formula A by A'.

Finally, predicate completion [10] give us the result of the theorem.

$$\Sigma,\ 26',\ CIRC(3',\ 28';\ Bad)\ \equiv\ \Sigma,\ 26',\ 27' \qquad \diamond$$

Theorem 1 shows that the nested abnormality theory described by 25 is equivalent to the second order theory whose axioms are Σ, 26 and 27. This means that using theorem proving methods for first order logic[14] we can decide the selectability of any situation with respect to the strategy described by axioms 1 to 4. For example, we can prove that action *Move(A,Table)* is selectable at S_0 (i.e., $Selectable(Result(Move(A,Table),S_0)))$, but action *Move(C,Table)* is not (i.e., $\neg Selectable(Result(Move(C,Table),S_0)))$.

The nested abnormality theory described by 25 not only allow us to characterize the behavior of a specific strategy, it also describes a particular use of circumscription that permits reasoning about declarative formalizations of heuristics of the sort proposed in this paper. The expression *25(strategy)* denotes the nested abnormality theory described by 25 parameterized for different strategy descriptions[15]. The idea is to replace axioms 1 to 4 by other sets of axioms describing different strategies, so that we can reason about the behavior of different strategies.

$$25(strategy) \equiv \Sigma,\ \{Better,\ min\ Bad:\ 5,\ \{min\ Good:\ strategy\}\}$$

In particular, formula *25(strategy)* allows us to analyze the following properties of a strategy.

Computability Formula 30 permits identifying *cycles* in the projections of *state-based* strategies[16]. The expression $s \prec s_1$ means that there is a non-empty sequence of *selectable* situations that leads from s to s_1. A *state-based* strategy contains a *cycle* (axiom 30) if there is a non-empty sequence of selectable situations that leads from a situation s to a different situation s_1 with the same associated state (i.e., whose associated database contains the same formulas as the database associated with s).

(29) $\forall s(\neg s \prec S_0 \wedge \neg s \prec S_g) \wedge \forall a s s_1(s \prec Result(a,s_1) \leftrightarrow Selectable(Result(a,s_1)) \wedge s \preceq s_1)$

(30) $Cycle(s,s_1) \leftrightarrow s \prec s_1 \wedge \forall p(DB(p,s) \leftrightarrow DB(p,s_1))$

A state-based strategy that contains a cycle (i.e., $25(strategy) \vdash \exists s s_1 Cycle(s,s_1)$) is *not computable*, because the action selection mechanism may enter into that cycle and iterate indefinitely. On the other hand, a *strategy* such that $25(strategy)$ allows us to prove that the set of *selectable* situations is *finite* (see examples later on) is clearly *computable*.

Correctness A strategy is *correct* iff the goal is achieved in all its terminal situations.

$$25(strategy) \vdash \forall s(Terminal(s) \rightarrow Achieved(S_g,s))$$

Quality of the Solutions The following formula allows us to compute the *maximum cost* n associated with a strategy[17].

(31) $Max\text{-}cost = n \leftrightarrow \exists s(Terminal(s) \wedge Length(s) = n) \wedge \forall s(Terminal(s) \rightarrow Length(s) \leq n)$

Given two correct and computable strategies, we say that the first is *better* than the second if the maximum cost of the first strategy is smaller than the maximum cost of the second. The intuition here is that the first strategy always solves the problem performing a smaller or equal number of actions.

Redundancy Two strategies are *redundant* iff they produce the same behaviors. That is, the axiom sets $25(strategy\text{-}1)$ and $25(strategy\text{-}2)$ are logically equivalent[18].

[14]Notice that the only second order axiom in Σ is an axiom of induction for situations, which we do not need to decide the selectability of a single situation.

[15]The expression *25(strategy)* can also be parameterized with respect to the problem description (axioms 9 and 24), the theory of action (axioms 7 to 23), or the mechanism of action selection (axiom 6).

[16]A strategy is *state-based* if whether an action is good or bad for a given goal at a particular situation –selectable– depends only on what holds at that situation (i.e. the *state* associated with the situation), and not on what actions have been selected at previous situations. All the strategies considered in the paper are state-based.

[17]The length of a situation that can be reached from the initial situation S_0 is defined as follows. $Length(Result(\{\},S_0)) = 0$ and $Length(Result(\{a|l\},S_0)) = 1 + Length(Result(\{l\},S_0))$, where l is a sequence of actions.

[18]We apply this definition to compare a strategy with the strategy resulting of adding to it several heuristics. If both strategies produce the same behaviors, we say that the heuristics added are *redundant* with respect to the original strategy.

Inconsistency Formula 32 can be used to detect inconsistencies in the projection of a strategy. The formulas 29 to 32 are examples of *verification axioms* that we add to Σ in order to reason about the behavior of different strategies.

(32) $Bad(a, S_g, s) \rightarrow \neg Good(a, S_g, s)$

A strategy is *inconsistent* iff there is a formula A such that $25(strategy) \vdash A \wedge \neg A$.

4 Taking Advice

We describe a scenario in which a program uses circumscription as described in *25(strategy)* to reason about declarative formalizations of heuristics. The program starts with an empty strategy. As different heuristics are suggested by the adviser, it considers how they may affect its problem solving behavior, and reacts accordingly.

The scenario tries to illustrate the idea that a program capable of reasoning non-monotonically about declarative formalizations of heuristics can have interesting *reflective behavior* [18] [20] [25] [24]. For example, it can save computational resources by detecting uncomputable or incorrect strategies. It can determine which of the heuristics is told improve, are redundant, partially redundant, or inconsistent with its current strategy. It can improve its problem solving strategy by adding action selection rules and axioms describing heuristics that will improve its behavior, and ignore those that are redundant with its current strategy. It can avoid inconsistencies, which may cause it to have an arbitrary behavior. It can learn by *taking advice* [13], and reflecting on it.

Initially, the advisor suggests to use the following heuristic: *If a block can be moved to final position, this should be done right away.* The program constructs *Strategy-1*, which is described by axiom 1. It can prove that the projection of *Strategy-1* contains the cycle described in fig. 1. Therefore, it concludes that *Strategy-1* is *not computable.*

$$25(Strategy\text{-}1) \vdash Cycle(S_0, Result(\{Move(C, Table), Move(C, E)\}, S_0))$$

The program asks for more advice, instead of trying to apply *Strategy-1* to solve the problem. The advisor proposes a different 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*, which is described by axiom 2 [19]. The projection of *Strategy-2* is shown in fig. 1. *Strategy-2* is *incorrect*, because some of its terminal situations do not satisfy the goal conditions.

$$\begin{aligned}25(Strategy\text{-}2) \vdash\ & Terminal(Result(\{Move(A, Table), Move(C, Table), Move(E, Table)\}, S_0)) \wedge \\ & \neg Achieved(S_g, Result(\{Move(A, Table), Move(C, Table), Move(E, Table)\}, S_0))\end{aligned}$$

The program still needs more advice. The advisor suggests now to consider both heuristics together. The program constructs *Strategy-3*, which is described by axioms 1 and 2. The projection of *Strategy-3* still contains a cycle, described in fig. 1.

$$\begin{aligned}25(Strategy\text{-}3) \vdash\ & Cycle(Result(Move(C, Table), S_0), \\ & Result(\{Move(C, Table), Move(E, Table), Move(E, D)\}, S_0))\end{aligned}$$

The advisor suggests a third heuristic: *If a block is in final position, do not move it.* The program constructs *Strategy-4* as the set of axioms 1 to 3. The set of situations that are selectable according to *Strategy-4* is finite (see fig. 1). The program knows the strategy is *correct*, since all its terminal situations happen to be solutions. The second order axiom of induction for situations (17) is needed here in order to prove that $\neg Terminal(s)$ holds for all the situations not mentioned in the theorem below.

$$\begin{aligned}25(Strategy\text{-}4) \vdash\ & (Terminal(s) \rightarrow Achieved(S_g, s)) \wedge (Terminal(s) \leftrightarrow \\ & s = Result(\{Move(A, Table), Move(C, B), Move(A, C)\}, S_0) \vee \\ & s = Result(\{Move(C, Table), Move(A, Table), Move(C, B), Move(A, C)\}, S_0))\end{aligned}$$

[19] *Strategy-2* is defined, in fact, by two action selection rules. The first one is axiom 2, the second is as follows $Holds(Final(x, S_g), s) \wedge Holds(On(x, Table), s) \rightarrow Bad(Move(x, y), S_g, s)$. This rule guarantees that *Strategy-2* terminates as shown in fig. 1. The rest of the strategies that terminate do not need this rule, because it is subsumed by axiom 3.

The advisor suggests a fourth heuristic: *If there is a block that is above a block it ought to be above in the goal configuration but it is not in final position (tower-deadlock), put it on the table.* The program constructs *Strategy-5* as the set of axioms 1 to 4. The set of situations that are selectable according to *Strategy-5* is finite (see fig. 1). The program can prove that *Strategy-5* is *correct.* It can also conclude that *Strategy-5* is *better* than *Strategy-4*, since it always solves the problem performing a smaller or equal number of actions[20].

$$25(\mathit{Strategy}\text{-}5) \vdash (\mathit{Terminal}(s) \rightarrow \mathit{Achieved}(S_g, s)) \land (\mathit{Terminal}(s) \leftrightarrow$$
$$s = \mathit{Result}(\{\mathit{Move}(A, \mathit{Table}), \mathit{Move}(C, B), \mathit{Move}(A, C)\}, S_0))$$

The advisor still suggests a fifth heuristic: *If a block is on the table but not in final position, do not move anything on that block.*

$$(33)\quad \mathit{Holds}(\mathit{On}(x, \mathit{Table}), s) \land \neg \mathit{Holds}(\mathit{Final}(x, S_g), s) \rightarrow \mathit{Bad}(\mathit{Move}(z, x), S_g, s)$$

The program constructs *Strategy-6* as the set of axioms 1 to 4 and 33. It can check that the axiom sets *25(Strategy-5)* and *25(Strategy-6)* are logically equivalent. This means that suggestion 33 is *redundant* with its current strategy. Therefore, including axiom 33 in its database will not improve its behavior.

The advisor finally suggests a sixth heuristic: *If there is a block that is above a block it ought to be above in the goal configuration but it is not in final position (tower-deadlock), 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.*

$$(34)\quad \mathit{Holds}(\mathit{Tower}\text{-}\mathit{deadlock}(x, S_g), s) \land \mathit{Holds}(\mathit{Clear}(z), s) \land \mathit{Holds}(\mathit{Final}(z, S_g), s) \land$$
$$\mathit{Holds}(\mathit{Clear}(z), S_g) \land z \neq w \rightarrow \mathit{Better}(\mathit{Move}(x, z), \mathit{Move}(x, w), S_g, s)$$

The program constructs *Strategy-7* as the set of axioms 1 to 4 and 34. Although axiom 34 describes a plausible heuristic, it is in contradiction with axiom 4. For example, axiom 4 implies that $\mathit{Move}(A, \mathit{Table})$ is good in the initial situation, whereas axioms 5 and 34 imply that $\mathit{Move}(A, \mathit{Table})$ is bad[21]. Using axiom 32, the program can prove that *Strategy-7* is *inconsistent.*

$$25(\mathit{Strategy}\text{-}7) \vdash \mathit{Good}(\mathit{Move}(A, \mathit{Table}), S_g, S_0) \land \neg \mathit{Good}(\mathit{Move}(A, \mathit{Table}), S_g, S_0)$$

Therefore, it rejects suggestion 34, because it is *inconsistent* with its current strategy.

5 Conclusions

We have proposed a representation scheme for the declarative formalization of heuristics based on the situation calculus and circumscription. The formalism has been applied to represent and reason about heuristics for the blocks world. It has been demonstrated that the particular use of circumscription proposed allows reasoning about the behavior of declarative formalizations of heuristics of the sort described in this paper. In particular, we have seen that the techniques presented are useful to determine the computability and correctness of a particular strategy (or a class of strategies) with respect to a given problem (or a class of problems). We have also considered issues involved in updating and composing strategic knowledge from different sources, such as determining whether a set of heuristics improve, are inconsistent or redundant with a particular strategy (or a class of strategies). The possibility of reasoning about these issues together with the natural composition of the declarative formalization of heuristics proposed allow a program to *reflect* on its own behavior and improve its problem solving strategy by simple additions or substitutions of sentences, as it has been shown in the *advice taking* scenario of section 4. This is, perhaps, the best feature of the language, its *elaboration tolerance* [17]. The flexibility of adapting to conceptual changes in the specification of a problem or its solution is a very important feature that procedural or dynamic logic languages do not have.

[20]The maximum cost of *strategy-5* is smaller than the maximum cost of *strategy-4* (i.e., $25(\mathit{Strategy}4) \vdash \mathit{Maxcost} = 4$ and $25(\mathit{Strategy}5) \vdash \mathit{Maxcost} = 3$).

[21]Notice that axiom 34 implies *Better(Move(A,F),Move(A,Table),S_g,S_0).*

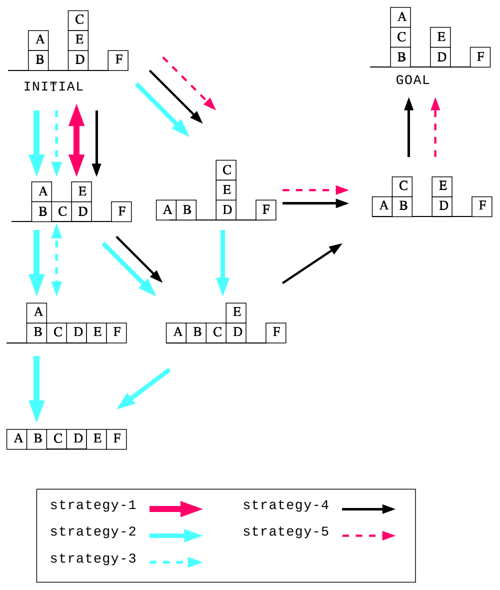

Figure 1: Behavior of different strategies for solving planning problems in the blocks world: (1) *Strategy-1* and *Strategy-3* are both uncomputable, since they allow cyclic behaviors; (2) *Strategy-2* describes a computable but incorrect strategy, its unique terminal situation is not a solution; (3) *Strategy-4* and *Strategy-5* describe two computable and correct strategies, together with their projections for a particular blocks world problem. It can be easily observed that *Strategy-5* is better than *Strategy-4*. Remember that the criterion for action selection is that, in absence of good actions, any action that is not implied to be bad can be selected.

Acknowledgments

This paper was inspired by a challenge problem proposed in [21]. The formalization of STRIPS in the *situation calculus* and its application to reason about action in the blocks world are fully developed in [15]. In addition to this, I'm particularly indebted to John McCarthy for encouraging, supporting, and enriching the writing of this work with many interesting ideas, some of which I have not been able to develop yet. Luc Steels introduced me to most of the problems discussed in section three, provided valuable ideas, and encouraged me all the way through. Vladimir Lifschitz helped me to understand the incompleteness of my original formalization, and taught me how to make it simpler and stronger. Mike Genesereth, Eyal Amir, Tom Costello, Winton Davies and Sasa Buvac read previous versions of the paper and contributed with useful comments. This work was supported partly by the Navy Grant N00014-94-1-0775, and partly by the Air Force Grant F49620-97-1-0207.

References

[1] Davis, E. (1990) Representations of Commonsense Knowledge. Morgan Kaufmann Publishers, Inc. San Mateo, California.

[2] Fikes, R.E., and Nilsson, N.J.. (1971) STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, 2:189-208.

[3] Gupta, N., and Nau, D.S. (1991) Complexity results for blocks-world planning. In Proceedings of AAAI-91.

[4] Genesereth, M. R., and Hsu, J.Y. (1989) Partial Programs. Logic Group Technical Report 89-20. Computer Science Department, Stanford University.

[5] Harel, D. (1984) Dynamic logic. In D. Gabbay and F. Guenthner, Eds. *Handbook of Philosophical Logic, Vol. II: extensions of Classical Logic,* pp 497-604. Dordrecht, the Netherlands: Reidel Publishing Company.

[6] van Harmelen, F. *Meta-level Inference Systems.* Research Notes in AI. London, San Mateo California, Pitman, Morgan Kaufmann, 1991.

[7] Harmelen, F et al. Knowledge-level Reflection, in *Enhancing the Knowledge Engineering Process*, contributions from Esprit, L. Steels and B. Lepape, editors, 1992.

[8] Van Harmelen F. and Balder J.R. $(ML)^2$*: a formal language for KADS models of expertise.* Knowledge Acquisition, Vol 4, nr 1, March 1992, pp. 127-161. Special issue: *The KADS approach to knowledge engineering.*

[9] Lifschitz, V. (1990) Frames in the space of situations. Artificial Intelligence 46, 365-376.

[10] Lifschitz, V. (1993) Circumscription. In D.M. Gabbay, C.J. Hogger, J., editors. Handbook of Logic in Artificial Intelligence and Logic Programming. Volume 3: Non-monotonic Reasoning and Uncertain Reasoning. Oxford University Press.

[11] Lifschitz, V. (1995) Nested abnormality theories. Artificial Intelligence 74, 1262-1277.

[12] Lin, F. (1997) Applications of the Situation Calculus to Formalizing Control and Strategic Information: The Prolog Cut Operator. In Proc. of IJCAI-97, 1412-1418.

[13] McCarthy, J. (1959) Programs with Common Sense. In Mechanization of Thought Processes, Proceedings of the Symposium of the National Physics Laboratory, pages 77-84, London, U.K., 1959. Her Majesty's Stationary Office.

[14] McCarthy, J. (1980) Circumscription -a form of non-monotonic reasoning. Artificial Intelligence, 13 (1,2): 27-39, 171-172.

[15] McCarthy, J. (1985) Formalization of STRIPS in Situation Calculus. Technical Report, Computer Science Department, Stanford University.

[16] McCarthy, J. (1986) Applications of Circumscription to Formalizing Common Sense Knowledge. Artificial Intelligence, 28 (1), 89-116.

[17] McCarthy, J. (1988) Mathematical logic in artificial intelligence. Daedalus, 117(1): 297-311.

[18] McCarthy, J. (1990) Formalizing common sense: papers by John McCarthy. Ablex, Norwood, NJ, 1990.

[19] McCarthy, J. (1990) Coloring Maps and the Kowalski Doctrine. Formalizing Common Sense.

[20] McCarthy, J. (1995) Making Robots Conscious of their Mental States. Computer Science Department, Stanford University. Machine Intelligence, 1995.

[21] McCarthy, J. (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

[22] McCarthy, J., and Hayes, P. (1969) Some philosophical problems from the standpoint of artificial intelligence. Machine Intelligence 4, 463-502.

[23] Reiter, R. (1993) Proving properties of states in the situation calculus. Artificial Intelligence 64, 337-351.

[24] Sierra, J. (1996) Software agents require formal knowledge level models. Ph.D. thesis. Free University of Brussels.

[25] Steels, L. (1996) The Spontaneous Self-organization of an Adaptive Language. Machine Intelligence 15. http://arti.vub.ac.be/www/steels/mi15.ps