Declarative Formalization of Strategies for Action Selection

Josefina Sierra-Santibáñez
Computer Science Department
Stanford University
Stanford, CA 94305
Phone: (650) 7234910
E-mail: jsierra@cs.stanford.edu

Abstract

We propose a representation scheme for the declarative formalization of strategies 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. It is demonstrated that circumscription solves the problem of projecting the strategies formalized in the paper, and that it allows us to derive useful conclusions about their computability, correctness, redundancy, inconsistency, and the quality of their solutions. Finally, an *advice taking* scenario is presented to illustrate how a program capable of reasoning non-monotonically about declarative formalizations of strategies can have interesting *reflective* behavior.

Introduction

Strategic knowledge has traditionally been specified using procedural programming languages or dynamic logic (Harel 1984) (Harmelen & Balder 1992). This paper proposes a representation scheme for the declarative formalization of *strategies for action selection* based on the *situation calculus* (McCarthy & Hayes 1969) and *circumscription* (McCarthy 1980) (McCarthy 1986).

The idea of representing *strategies* as sets of action selection rules (Genesereth & Hsu 1989) 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 move it to the table than anywhere else; (3) If a block is in final position, 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*[2]. A *consistent* set of action selection rules defines a particular *strategy.*

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

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

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

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

In addition to sets of action selection rules describing particular strategies, we need some axioms that allow a program to understand the predicates *good, bad* and *better* in terms of action selection. The predicate *Better*

[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}.

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

establishes a partial order among a set of actions with respect to a given goal and a particular situation. The following axiom says 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[3].

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

An important issue in reasoning about strategies is to foresee their consequences on action selection. We will call the problem of determining what sequences of actions can be selected according to a given strategy the *problem of projecting the strategy*[4]. The projection problem for strategies addresses the issue of characterizing the behavior of a particular strategy (or a class of strategies) when applied to solve a specific problem (or a class of problems). We talk about the *projection problem of a strategy* to distinguish it from the *projection problem* of the theory of action the strategy is about, which addresses the issue of characterizing the effects of the actions as opposed to the effects of the strategy on action selection.

The following axiom addresses this issue by defining a fluent called *Selectable*. 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.

$$\begin{aligned} (6) \qquad & Selectable(s) \leftrightarrow (s = S_0 \vee \\ & \exists s_1 a (Selectable(s_1) \wedge \neg Achieved(S_g, s_1) \wedge \\ & Prec(a, s_1) \wedge s = Result(a, s_1) \wedge (Good(a, S_g, s_1) \vee \\ & (\neg \exists b Good(b, S_g, s_1) \wedge \neg Bad(a, S_g, s_1))))) \end{aligned}$$

The introduction of axiom 6 describing the behavior of the fluent *Selectable* allows us to apply the nonmonotonic machinery developed for reasoning about action to the problem of reasoning about declarative formalizations of strategies for action selection. In the same way circumscription can be used to jump to the conclusion that a fluent does not change unless stated otherwise (i.e., to solve the *frame* problem), it can be used to assume that an action is “not good” or “not bad” unless it can be deduced from the set of axioms describing a strategy that it is so. This 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 in such a way that the conclusions about the selectability of different situations adapt automatically to the addition of consistent heuristics which may become available later on.

[3]The idea here is to select always the best possible action.

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

[5]We define $Achieved(s_g, s)$ and $Prec(a, s)$ formally later on. In general, 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.

Blocks World Example

In this section and the next, we show how the ideas outlined above can be applied to formalize and reason about heuristics for moving blocks in order to solve planning problems in the blocks world. The strategies formalized in the paper describe different algorithms for solving planning problems in the *elementary blocks world* domain (Gupta & Nau 1991). First, we summarize a very elegant and simple formalization of STRIPS in the *situation calculus*, and its application to reasoning about action in the blocks world, described in (McCarthy 1985). Then, we propose a *nested abnormality theory (NAT)* (Lifschitz 1995) that solves the problem of projecting the strategy described by axioms 1 to 4. In section 3, this theory is generalized into a class of *NAT's* which apply circumscription in a particular way that is useful for studying the projections of declarative formalizations of strategies of the sort described in this paper.

In (McCarthy 1985), John McCarthy proposes a very simple formalization of STRIPS (Fikes & Nilsson 1971) 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.

There are variables of the following sorts: for situations (s, s_1, …), for actions (a, a_1, …), and for propositional fluents (p, p_1, …)[6]. Associated with each sit-

[6]In the paper, we use the expression “propositional flu-

uation is a database of propositions, 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) *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

$$(7)\quad \begin{aligned} DB(p, Result(a, s)) \leftrightarrow \\ (Prec(a, s) \wedge (Add(p, a, s) \vee \\ (DB(p, s) \wedge \neg Delete(p, a, s)))) \vee \\ (\neg Prec(a, s) \wedge DB(p, s)) \end{aligned}$$

In (McCarthy 1985), an example of how to use this formalization of STRIPS to reason about action in the blocks world is given. The example is as follows (we have modified the initial conditions, and added a uniqueness of names axiom). The variables x, y and z range over blocks. The constant blocks used in the example are A, B, C, D, E, F, and T (for *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[7]. 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.

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

$$(9)\quad \begin{aligned} DB(p, S_0) \leftrightarrow \exists xy(p = On(x, y) \wedge \\ ((x = A \wedge y = B) \vee (x = B \wedge y = T) \vee \\ (x = C \wedge y = E) \vee (x = D \wedge y = T) \vee \\ (x = E \wedge y = D) \vee (x = F \wedge y = T))) \end{aligned}$$

$$(10)\quad \begin{aligned} Prec(a, s) \leftrightarrow \exists xy(a = Move(x, y) \wedge \\ \forall z \neg DB(On(z, x), s) \wedge \\ (y \neq T \rightarrow \forall z \neg DB(On(z, y), s)) \wedge \\ \neg DB(On(x, y), s) \wedge x \neq T \wedge x \neq y) \end{aligned}$$

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

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

For example, from axioms 7 to 12 we can prove that[8]

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

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.

The databases in the formalization of the blocks world presented above contain only propositional fluents of the form $On(x, y)$ for x, $y \in \{A,B,C,D,E,F,T\}$. These propositional fluents are called *frame fluents* (McCarthy & Hayes 1969) (Lifschitz 1990), since any configuration of the blocks *A,B,C,D,E,* and F can be described by combinations of their values[9].

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

The following axiom states 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 (Reiter 1993) from s by executing a nonempty sequence

ent" and the word "proposition" to refer to the corresponding reified formula.

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

[8]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).

[9]The symbols c_1 and c_2 are syntactical variables ranging over block constants.

of actions ($s \leq s_1$ is an abbreviation for $s < s_1 \vee s = s_1$) [10]. We include domain closure axioms for blocks, actions and situations. Axiom 16 restricts the domain of situations to those that can be reached from the initial situation S_0 or from the goal situation S_g. Finally, we introduce an axiom of induction for situations 17, which allows us to prove that a property holds for all the situations that can be reached from a given situation.

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

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

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

In addition to frame fluents, we use a number of *derived fluents*, such as *clear, final, above, tower-deadlock, terminal,* and *achieved*, which are partially[11] defined in terms of the frame fluents.

$$Holds(Clear(x), s) \leftrightarrow x = T \vee \neg \exists y Holds(On(y, x), s) \tag{18}$$

$$Holds(Final(x, S_g), s) \leftrightarrow (Holds(On(x, T), s) \wedge Holds(On(x, T), S_g)) \vee \exists y(Holds(Final(y, S_g), s) \wedge Holds(On(x, y), s) \wedge Holds(On(x, y), S_g)) \tag{19}$$

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

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

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

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

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 a class of problems.

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

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

[10] Notice the distinction between $Achieved(S_g, s)$ (axiom 23) and *reachable* $\leq$ (axiom 15). It 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.

[11] 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)* that can be derived from axioms 7 to 17. (Davis 1990) 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.

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 later on.

The *nested abnormality theory* described below 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 (and, therefore, what sequences of actions) can be selected according to the strategy. Let Σ be the conjunction of the universal closures of the formulas 6 to 24.

$$\Sigma,\ \{Better,\ min\ Bad: 5,\ \{min\ Good:\ 1,\dots,4\}\} \tag{25}$$

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 describing the strategy of the program (axioms 1 to 4). The predicate *Bad* is circumscribed with respect to the result of the 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 assumed for the blocks world (axioms 7 to 23), the specific problem reasoned about (axioms 9 and 24), and the mechanism for action selection (axiom 6).

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.

$$\begin{aligned} Good(Move(x,y),S_g,s) \leftrightarrow\ & (Holds(Tower\text{-}deadlock(x,S_g),s) \wedge y = T) \vee \\ & (\neg Holds(Final(x,S_g),s) \wedge \\ & Holds(Final(x,S_g),Result(Move(x,y),s))) \end{aligned} \tag{26}$$

$$\begin{aligned} Bad(Move(x,y),S_g,s) \leftrightarrow\ & Holds(Final(x,S_g),s) \vee \\ & (y \neq T \wedge \neg Holds(Final(x,S_g),s) \wedge \\ & \neg\exists z Holds(Final(x,S_g),Result(Move(x,z),s))) \end{aligned} \tag{27}$$

Proof The characterization of the semantics of NAT's in terms of second order logic theories including circumscription formulas and proposition 1 in (Lifschitz 1995) allow us to prove the equivalence between the following axiom sets[12].

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

Now, we use several rules for computing circumscription described in (Lifschitz 1993). The first equivalence below can be proved using formula (19) and proposition 2 in (Lifschitz 1993). The second equivalence uses formula (19) and proposition 3 in that paper.

$$\begin{aligned} \Sigma,\ CIRC(5', CIRC(1',\dots,4';\ Good);\ Bad;\ Better) &\equiv \\ \Sigma,\ CIRC(5',3',2',26';\ Bad;\ Better) &\equiv \\ \Sigma,\ 26',\ CIRC(3',\ \exists better(5' \wedge 2');\ Bad) & \end{aligned}$$

Using the equivalence (27) in section 3.2 of (Lifschitz 1993), we can prove that $\exists better(5' \wedge 2')(better)$ is equivalent to the following formula, which does not depend on *better*.

$$\begin{aligned} & y \neq T \wedge \neg Holds(Final(x,S_g),s) \wedge \\ & \neg\exists z Holds(Final(x,S_g),Result(Move(x,z),s)) \rightarrow \\ & Bad(Move(x,y),S_g,s) \end{aligned} \tag{28}$$

Finally, predicate completion (Lifschitz 1993) 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[13] 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,T)* is selectable at S_0 (i.e., $Selectable(Result(Move(A,T),S_0))$), but action *Move(C,T)* is not (i.e., $\neg Selectable(Result(Move(C,T),S_0))$).

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

[13] 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.

Advice Taking Scenario

The nested abnormality theory described by 25 not only solves the problem of projecting a specific strategy, it also describes a particular use of circumscription that allows reasoning about declarative formalizations of strategies of the sort proposed in this paper. The expression *25(strategy)* denotes the nested abnormality theory described by 25 parameterized for different strategy descriptions[14]. 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.

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

We describe a scenario in which a program uses *25(strategy)* to reason about declarative formalizations of strategies. The program starts with an empty strategy. As different heuristics are suggested by the adviser, the program 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 strategies can have interesting *reflective behavior* (McCarthy 1990b) (McCarthy 1995) (Steels 1996) (Sierra 1996). 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, accordingly, by adding and substituting action selection rules and axioms. It can avoid inconsistencies, which may cause it to have an arbitrary behavior. It can learn by *taking advice* (McCarthy 1959), 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. The projection of *Strategy-1* allows cyclic behaviors, such as the one described in fig. 1. The program concludes that *Strategy-1* is not *computable.*

We use the following formula to identify cycles in the projections of *state-based* strategies[15]. The expression $s \prec s_1$ means that there is a nonempty sequence of

[14] 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 for action selection (axiom 6).

[15] A strategy is *state-based* if whether an action is good or bad for a given goal at a particular situation depends only

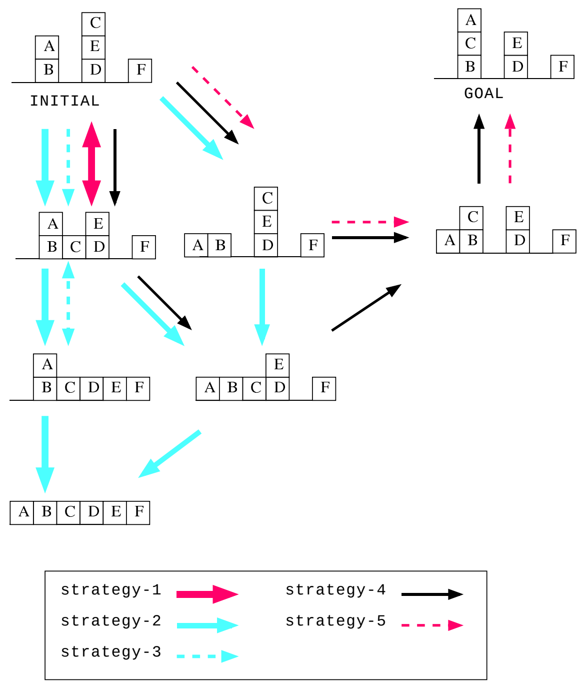

Figure 1: Behavior of different strategies for solving planning problems in the blocks world: (1) *Strategy-1* and *Strategy-3* are both uncomputable, since they allow cyclic behaviors; (2) *Strategy-2* describes a computable but incorrect strategy, its unique terminal situation is not a solution; (3) *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*.

selectable situations that leads from s to s_1. A *state-based* strategy contains a *cycle* (axiom 30) if there is a nonempty 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).

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

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

In particular, the program can prove that $\exists s s_1 Cycle(s, s_1)$ holds in the projection of *Strategy-1* by finding appropriate values for s and s_1.

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

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 [16]. The projection of *Strategy-2* is shown in fig. 1. *Strategy-2* is an *incorrect* strategy, because it allows terminal situations which do not satisfy the goal conditions.

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

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 allows cyclic behaviors, such as the one described in fig. 1, i.e. it satisfies $\exists s s_1 Cycle(s, s_1)$.

$$25(Strategy\text{-}3) \vdash Cycle(Result(Move(C,T), S_0), Result(\{Move(C,T), Move(E,T), Move(E,D)\}, S_0))$$

The advisor suggests a third heuristic: *If a block is in final position, do not move it.* The program constructs *Strategy-4* as the set of axioms 1 to 3. The set of situations that are selectable according to projection of *Strategy-4* (see fig. 1) is finite. 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.

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

The advisor suggests a fourth heuristic: *If there is a block that is above a block it ought to be above 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 projection of *Strategy-5* (see fig. 1) is finite. 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[17].

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

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.*

$$(31)\quad Holds(On(x,T), s) \wedge \neg Holds(Final(x, S_g), s) \rightarrow Bad(Move(z,x), S_g, s)$$

on what holds at that situation (i.e. the *state* associated with that situation), and not on what situations or actions have been selected so far. All the strategies considered in the paper are state-based.

[16] *Strategy-2* is defined, in fact, by two action selection rules. The first one is axiom 2, the second is as follows $Holds(Final(x, S_g), s) \wedge Holds(On(x,T), s) \rightarrow Bad(Move(x,y), S_g, s)$. This rule guarantees that the strategy 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.

[17] It compares the maximum length of the solutions generated by strategies 4 and 5.

The program constructs *Strategy-6* as the set of axioms 1 to 4 and 31. It can check that the projections of *Strategy-5* and *Strategy-6* are identical. This means that suggestion 31 is *redundant* with its current strategy. Therefore, including it in the database will not improve the program's 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.*

$$(32) \quad Holds(Tower\text{-}deadlock(x, S_g), s) \wedge \\ Holds(Clear(z), s) \wedge Holds(Final(z, S_g), s) \wedge \\ Holds(Clear(z), S_g) \wedge z \neq w \rightarrow \\ Better(Move(x, z), Move(x, w), S_g, s)$$

We use the following formula to detect *inconsistencies* in the projection of a strategy. The formulas 29, 30 and 33 are examples of verification axioms that can be added to Σ in order to reason about the behavior of different strategies.

$$(33) \quad Bad(a, S_g, s) \rightarrow \neg Good(a, S_g, s)$$

The program constructs *Strategy-7* as the set of axioms 1 to 4 and 32. Although axiom 32 describes a plausible heuristic, it is in contradiction with axiom 4. For example, axiom 4 implies that $Move(A,T)$ is good in the initial situation, whereas axioms 5 and 32 imply that $Move(A,T)$ is bad[18]. Using axiom 33, the program can prove that *Strategy-7* is *inconsistent.*

$$25(Strategy\text{-}7) \vdash Good(Move(A, T), S_g, S_0) \wedge \\ \neg Good(Move(A, T), S_g, S_0)$$

Therefore, it rejects suggestion 32, because it is *inconsistent* with its current strategy.

Conclusions

In this paper, we have considered a particular planning problem in which the states of both the initial and goal situations are completely determined. Interesting reasoning about strategic knowledge is concerned with classes of problems, instead of particular instances. The representation scheme and reasoning method proposed have however a broader scope. They can also be applied to entire classes of problems, provided these classes of problems are appropriately axiomatized.

The point of focusing on the study of a particular example was to illustrate potential applications of reasoning about declarative formalizations of strategies in a simple setting. We have seen, for example, 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 compositionality of the declarative formalization of strategies proposed, allow a program to reflect on its own behavior, and improve its problem solving strategy by simple additions or substitutions of sentences, much in the same way it happens in natural language. This is perhaps the best feature of the language, its *elaboration tolerance* (McCarthy 1988). The flexibility of adapting smoothly 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.

There are a number of interesting issues about the declarative formalization of strategies we have not considered. We have formalized only *state-based* strategies. An important number of strategies depend on chronological information, such as whether an action has been selected at a particular situation. The related problem of formalizing control information in the situation calculus is addressed by (Lin 1997). Derivations in logic programming are identified with situations, and a fluent *accessible* is defined in order to characterize those situations which correspond to derivations of Prolog programs including *cut.* Our work differs from (Lin 1997) in two aspects: (1) the emphasis on *representation*, in particular, on proposing a representation scheme for the *declarative formalization of strategies*; and (2) the use of *non-monotonic reasoning* to achieve *elaboration tolerance* and *reflection.*

Planning is one of the most challenging problems. Humans are sometimes able to come up with *heuristics* such as those formalized here. A deeper understanding of a domain may allow programs to come up with them as well. Issues such as the *safeness* and *postponability* (McCarthy 1990a) of certain actions and situations, with respect to the achievement of certain goals, underly the design of the heuristics formalized for the blocks world. Our hypothesis is that these issues may

[18]Notice that axiom 32 implies *Better(Move(A,F),Move(A,T),*S_g*,*S_0*).*

play a crucial role in the problem of automating the design of heuristics, which we have only begun to investigate.

Acknowledgments

This paper was inspired by a challenge problem proposed in (McCarthy 1997). The formalization of STRIPS in the *situation calculus* and its application to reason about action in the blocks world are fully developed in (McCarthy 1985). 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

Davis, E. 1990. *Representations of Commonsense Knowledge*. Morgan Kaufmann Publishers, Inc. San Mateo, California.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Genesereth, M. R., and Hsu, J. 1989. Partial programs. Technical Report Logic Group 89-20, Department of Computer Science, Stanford University.

Gupta, N., and Nau, D. 1991. Complexity results for blocks-world planning. In *Proceedings of AAAI-91*.

Harel, D. 1984. Dynamic logic. In *Handbook of Philosophical Logic*, volume II: Extensions of Classical Logic, D. Gabbay and F. Guenthner, Ed., 497–604. Dordrecht, the Netherlands: Reidel Publishing Company.

Harmelen, F., and Balder, J. 1992. *(ML)*^{2}: a formal language for KADS models of expertise. *Knowledge Acquisition* 4(1):127–161. Special issue: *The KADS approach to knowledge engineering*.

Lifschitz, V. 1990. Frames in the space of situations. *Artificial Intelligence* 46:365–376.

Lifschitz, V. 1993. Circumscription. In *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.

Lifschitz, V. 1995. Nested abnormality theories. *Artificial Intelligence* 74:1262–1277.

Lin, F. 1997. Applications of the situation calculus to formalizing control and strategic information: The prolog cut operator. In *Proceedings of IJCAI-97*, 1412–1418.

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence* 4:463–502.

McCarthy, J. 1959. Programs with common sense. In *Mechanization of Thought Processes, Proceedings of the Symposium of the National Physics Laboratory*, 77–84.

McCarthy, J. 1980. Circumscription -a form of non-monotonic reasoning. *Artificial Intelligence* 13:27–39.

McCarthy, J. 1985. Formalization of STRIPS in situation calculus. Technical Report Formal Reasoning Group, Department of Computer Science, Stanford University.

McCarthy, J. 1986. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence* 28:89–116.

McCarthy, J. 1988. Mathematical logic in artificial intelligence. *Daedalus* 117:297–311.

McCarthy, J. 1990a. Coloring maps and the Kowalski doctrine. In *Formalizing common sense: papers by John McCarthy*. Edited by Vladimir Lifschitz, Ablex, Norwood, NJ.

McCarthy, J. 1990b. *Formalizing common sense: papers by John McCarthy*. Edited by Vladimir Lifschitz, Ablex, Norwood, NJ.

McCarthy, J. 1995. Making robots conscious of their mental states. Technical Report Formal Reasoning Group, Department of Computer Science, Stanford University.

McCarthy, J. 1997. Course on formalization of common sense - non-monotonic reasoning. Lecture notes of CS323, Department of Computer Science, Stanford University.

Reiter, R. 1993. Proving properties of states in the situation calculus. *Artificial Intelligence* 64:337–351.

Sierra, J. 1996. *Software agents require formal knowledge level models*. Ph.D. Dissertation, Free University of Brussels.

Steels, L. 1996. The spontaneous self-organization of an adaptive language. *Machine Intelligence* 15.