From: AAAI Technical Report WS-96-07. Compilation copyright © 1996, AAAI (www.aaai.org). All rights reserved.

How to Execute a Conditional?*

Richard B. Scherl†
Department of Computer and Information Science
New Jersey Institute of Technology
Newark, New Jersey 07102
scherl@vienna.njit.edu

Yves Lespérance
Department of Computer Science
Glendon College
York University
2275 Bayview Ave.
Toronto, ON, Canada M4N 3M6
lesperan@cs.toronto.edu

Hector J. Levesque
Fangzhen Lin
Ray Reiter
Department of Computer Science
University of Toronto
Toronto, Ontario
Canada M5S 1A4
{hector fl reiter}@cs.toronto.edu

Abstract

The execution of a plan containing conditionals by an agent with incomplete knowledge poses some difficult problems. In order for the conditional to be meaningful, the agent must know whether or not the condition is true at execution time. This paper proposes one solution to this problem by integrating sensing actions into GOLOG, a high-level robot programming language. At run time, the interpreter performs a small amount of planning to ensure that the agent will know whether or not a condition is true prior to the point where the test for the truth of the condition needs to be made.

Introduction

Artificial agents, be they robots or software agents, need to be designed to achieve their goals in a world about which the agents have incomplete knowledge. Our approach to the design of such agents is to develop a high level language, called GOLOG (Levesque *et al.* 1996; Lespérance *et al.* 1995; 1994), to specify such agents. Programs written in GOLOG can be seen as schematic plans with the details automatically filled in at execution time. GOLOG programs are composed in a way similar to conventional high-level computer programs, however it has a semantics grounded in the situation calculus (Reiter 1991). The output of the GOLOG interpreter is a sequence of primitive actions expressed in the situation calculus.

Unlike conventional computer programs, GOLOG programs frequently need to work under incomplete knowledge. Consider the conditional "if C then α". Clearly, to execute the command, the agent needs to know whether C is true. This presents no conceptual problem if the agent has complete knowledge, which is the assumption made by classical planners and compilers for traditional computer programs. Relaxing this assumption exposes many difficult problems, some of which have been discussed by Etzioni *et al* (1992). For instance, should the agent ask its sensors first or should it check its knowledge base first? In addition to the sensory and mental actions, what other actions is the agent is allowed to do? Levesque(1996) discusses the limitations of the classical definition of planning and generalizes the definition to cover cases where the agent has incomplete knowledge of the initial situation and can execute sensing actions.

In this paper we adopt a version of the situation calculus with a representation of knowledge and knowledge-producing actions(Scherl and Levesque 1993) as the semantic foundation for GOLOG. Given the conditional "if C then α" to execute in state s, the agent strives to achieve a state in which it knows whether C is true in s. Taken as a planning problem, this differs from classical planning in the following ways[1]:

1. The goal is epistemic. (To have the knowledge of whether something is true.)
2. It involves more than one state. (To achieve a state where the truth value of something in an earlier state comes to light.)

Now the output of the GOLOG interpreter is a sequence of basic situation calculus actions that may include sensing actions. There may be many ways to achieve the goal of knowing whether C is true. The achieving of the goal of knowing the truth value of

*This research received financial support from the Information Technology Research Center (Ontario, Canada), the Institute for Robotics and Intelligent Systems (Canada), and the Natural Science and Engineering Research Council (Canada)

†Most of the work was performed while a National Sciences and Engineering Research Council of Canada International Postdoctoral Fellow

[1]It is interesting to note here that because of the second feature, classical planning formalisms such as STRIPS are no longer expressive enough, and we have to take seriously formalisms that represent states explicitly.

a condition is an additional element of the high-level schematic nature of GOLOG plans that are filled in with the details at execution time, e.g., the particular sequence of actions (including sensing actions) needed to ensure that the agent will know whether or not the condition is true at the time the test needs to be made.

The following problem (based on an example due originally to Savage and then modified by Poole) will be used to illustrate the approach taken here[2]:

> The problem is to make a 3 egg omelette from a set of eggs some of which may be bad. None of the eggs in the omelette should be bad. We have two bowls; we can only see if an egg is bad if it is in a bowl. We can throw out the whole bowl.
>
> We can assume a limited number of eggs (say 5), and add the statement that there are at least 3 good eggs. Furthermore, the agent has two methods of determining whether or not the egg is bad – visual and olfactory.

The following are the actions available to the agent:

- Break an egg into a bowl.
- Pour the contents of one bowl into another.
- Throw out the contents of one bowl.
- Visually inspect a bowl to see if there are any bad eggs in it.
- Sniff a bowl to see if there are any bad eggs in it.

The goal is to:

> Have three eggs in a bowl that are not bad.

In the next two sections, the situation calculus background and then the GOLOG programming language are discussed. The addition of knowledge-producing actions to the situation calculus and the needed revisions to the GOLOG interpreter are covered in the following two sections.

The Situation Calculus: A Language for Specifying Dynamics

The situation calculus (following the presentation in (Reiter 1991)) is a first-order language for representing dynamically changing worlds in which all of the changes are the result of named *actions* performed by some agent. For example:

$$\text{BREAK\_INTO}(\textit{bowl}),\ \text{FETCH}(\textit{container}),\ \text{POUR}(\textit{bowl}1, \textit{bowl}2),\ \text{THROW\_OUT}(\textit{bowl})$$

[2]For a full axiomatization of the omelette problem see (Scherl 1996b).

Terms are used to represent states of the world–i.e. *situations*. If α is an action and s a situation, the result of performing α in s is represented by $do(\alpha, s)$. The constant S_0 is used to denote the initial situation. Relations whose truth values vary from situation to situation, called *fluents*, are denoted by predicate symbols taking a situation term as the last argument. For example, $\text{BROKEN}(x, s)$ means that object x is broken in situation s. Functions whose denotations vary from situation to situation are called *functional fluents*. They are denoted by function symbols with an extra argument taking a situation term, as in $\text{NUMBER\_EGGS}(\textit{bowl}, s)$, i.e., the number of eggs in *bowl* in s.

In the omelette example, the following fluents are needed:

$$\text{IN}(\textit{egg}, \textit{bowl}, s),\ \text{BAD}(\textit{egg}, s),\ \text{BROKEN}(\textit{egg}, s),\ \text{HOLDING}(\textit{egg}, s),\ \text{NUMBER\_EGGS}(\textit{bowl}, s)$$

The following non-fluents are needed:

$$\text{EGG}(x),\ \text{SMALL\_BOWL},\ \text{LARGE\_BOWL},\ \text{BASKET}$$

It is assumed that the axiomatizer has provided for each action $\alpha(\vec{x})$, an *action precondition axiom* of the form given in 1, where $\pi_\alpha(s)$ is a formula specifying the preconditions for action $\alpha(\vec{x})$.

Action Precondition Axiom

$$\text{POSS}(\alpha(\vec{x}), s) \equiv \pi_\alpha(\vec{x}, s) \tag{1}$$

An action precondition axiom for the action BREAK_INTO is given below.

$$\begin{gathered} \text{POSS}(\text{BREAK\_INTO}(\textit{bowl}), s) \\ \equiv \\ \exists\, \textit{egg}\, \neg\text{BROKEN}(\textit{egg}, s) \land \text{HOLDING}(\textit{egg}, s) \end{gathered} \tag{2}$$

The predicate POSS allows us to define situations reachable by an executable sequence of actions. Intuitively, $s \leq s'$ holds if and only if there is a sequence of zero or more *executable* actions which lead from situation s to s'. An action is executable if the action's preconditions are true in the situation in which the action is to be performed. We need the following axioms[3]:

$$\neg s < S_0 \tag{3}$$

$$s < do(a, s') \equiv (\text{POSS}(a, s') \land s \leq s') \tag{4}$$

where $s \leq s'$ is shorthand for $s < s' \lor s = s'$.

[3]The full set of foundational axioms for the situation calculus can be found in (Lin and Reiter 1994). These are extended to cover the situation calculus with knowledge in (Scherl 1996a).

Furthermore, it is assumed that the axiomatizer has provided for each fluent F, two *general effect axioms* of the form given in 5 and 6.

General Positive Effect Axiom for Fluent F

$$\text{Poss}(a,s) \wedge \gamma_F^+(\vec{x},a,s) \rightarrow F(do(\vec{x},a,s)) \quad (5)$$

General Negative Effect Axiom for Fluent F

$$\text{Poss}(a,s) \wedge \gamma_F^-(\vec{x},a,s) \rightarrow \neg F(do(\vec{x},a,s)) \quad (6)$$

Here $\gamma_F^+(a,s)$ is a formula describing under what conditions doing the action a in situation s leads the fluent F to become true in the successor situation $do(a,s)$ and similarly $\gamma_F^-(a,s)$ is a formula describing the conditions under which performing action a in situation s results in the fluent F becoming false in situation $do(a,s)$. Effect axioms provide the "causal laws" for the domain of application.

Reiter(1991) shows how to derive a set of *successor state axioms* of the form given in 7 from the axioms (positive and negative effect) and a completeness assumption[4].

Successor State Axiom

$$\begin{aligned}&\text{Poss}(a,s) \rightarrow [F(\vec{x},do(a,s)) \equiv \\ &\gamma_F^+(\vec{x},a,s) \vee (F(\vec{x},s) \wedge \neg\gamma_F^-(\vec{x},a,s))]\end{aligned} \quad (7)$$

Similar successor state axioms may be written for functional fluents. A successor state axiom is needed for each fluent F, and an action precondition axiom is needed for each action a.

The following are successor state axioms for the fluents Broken and In:

$$\begin{aligned}&\text{Poss}(a,s) \rightarrow [\text{Broken}(e,do(a,s)) \equiv \\ &\quad (\text{Holding}(e,s) \wedge \exists b\, a = \text{break\_into}(b)) \vee \\ &\quad\quad \text{Broken}(e,s)]\end{aligned} \quad (8)$$

$$\begin{aligned}&\text{Poss}(a,s) \rightarrow [\text{In}(e,b_1,do(a,s)) \equiv \\ &\quad (\text{Holding}(e,s) \wedge a = \text{break\_into}(b_1)) \vee \\ &\quad (\exists b_2\; a = \text{pour}(b2,b) \wedge \text{In}(e,b_2,s) \vee \\ &\quad \text{In}(e,b_1,s) \wedge \neg((a = \text{throw\_out}(b) \vee \\ &\quad\quad \exists b2\, a = \text{pour}(b,b2))]\end{aligned} \quad (9)$$

The axioms specify completely all possible ways that the truth value of the fluents can change in moving from situation s to situation $do(a,s)$.

[4]This then amounts to a solution to the frame problem under the assumption there are no ramifications, i.e., indirect effects of actions. In (Lin and Reiter 1994), the approach discussed in this section is extended to work with state constraints (ramifications) by compiling the effects of the state constraints into the successor state axioms. Reiter(1991) also discusses the need for unique name axioms for actions and situations.

GOLOG: Adding complex actions to the situation calculus

Actions in the situation calculus are primitive and determinate. They are like primitive computer instructions (e.g. assignment). We need complex actions for the same reason that we need programs. This set of complex action expressions forms a programming language that we call GOLOG (alGOl in LOGic).

Complex actions could be treated as first class entities, but since the tests that appear in forms like **if** ϕ **then** δ_1 **else** δ_2 involve formulas ϕ, this means that we must reify fluents and formulas. Moreover, it is necessary to axiomatize the correspondence between these reified formulas and the actual situation calculus formulas. This results in a much more complex theory.

Instead we treat complex action expressions as abbreviations for expressions in the situation calculus logical language. They may be thought of as macros that expand into the genuine logical expressions. A particular execution sequence of a complex action expression will be a sequence of situation calculus primitive actions. In this way the solution to the frame problem (for primitive actions) is extended to complex actions as well, since the complex actions are eliminated by macro expansion.

This is done by defining a predicate Do as in $Do(\delta,s,s')$ where δ is a complex action expression. $Do(\delta,s,s')$ is intended to mean that the agent's doing action δ in situation s leads to a (not necessarily unique) situation s'. The inductive definition of Do includes the following cases:

- $Do(a,s,s') \overset{\text{def}}{=} \text{Poss}(a,s) \wedge s' = do(a,s)$ — simple actions
- $Do(\phi?,s,s') \overset{\text{def}}{=} \phi[s] \wedge s = s'$ — tests
- $Do([\delta_1;\delta_2],s,s') \overset{\text{def}}{=} \exists s''(Do(\delta_1,s,s'') \wedge Do(\delta_2,s'',s'))$ — sequences
- $Do([\delta_1|\delta_2],s,s') \overset{\text{def}}{=} Do(\delta_1,s,s') \vee Do(\delta_2,s,s')$ — nondeterministic choice of actions
- $Do((\Pi x)\delta,s,s') \overset{\text{def}}{=} \exists x\, Do(\delta,s,s')$ — nondeterministic choice of parameters
- $Do($**if** ϕ **then** δ_1 **else** $\delta_2,s,s') \overset{\text{def}}{=}$

$$(\phi[s] \rightarrow \text{Do}(A,s,s')) \wedge (\neg\phi[s] \rightarrow \text{Do}(B,s,s'))$$

- $Do(\delta^*,s,s') \overset{\text{def}}{=}$ — nondeterministic iteration

$$\begin{aligned}&\forall P(\forall s_1\, P(s_1,s_1) \supset \\ &\quad \forall s_1,s_2,s_3[P(s_1,s_2) \wedge Do(\delta,s_2,s_3) \supset P(s_1,s_3)] \supset P(s,s')\end{aligned}$$

- $Do(\textbf{while } \phi \textbf{ do } \delta, s, s') \overset{\text{def}}{=}$

$$\begin{aligned}&\forall P(\\ &\qquad \forall s_1\, \neg\phi[s_1] \rightarrow P(s_1,s_1)\\ &\qquad \forall s_1,s_2,s_3\, (\phi[s_1] \wedge Do(A,s_1,s_2) \wedge P(s_2,s_3))\\ &\quad \rightarrow P(s_1,s_3)\\ &\qquad\qquad\qquad\qquad) \rightarrow P(s,s')\end{aligned}$$

Additionally, the notation $\phi[s]$ means that a situation argument is added to all fluents in ϕ, if one is missing. The definition of while loops could be simplified by utilizing the definition of nondeterministic iteration.

Additionally, there is the following abbreviation:

$$\text{ACHIEVE}(\phi) \overset{\text{def}}{=} [\mathcal{A}_1 \mid \mathcal{A}_2 \mid \ldots \mid \mathcal{A}_n]^*; \phi?$$

A possible GOLOG program for the omelette example[5] is as follows:

```
while ¬(NUMBER_EGGS(LARGE_BOWL) = 3)
    (Πe) ACHIEVE(HOLDING(e));
        BREAK_INTO(SMALL_BOWL);
        if BAD(SMALL_BOWL)
          thenTHROW_OUT(SMALL_BOWL);
          elsePOUR(SMALL_BOWL, BIG_BOWL);
```

The problem that is being addressed in this paper is how to ensure that the agent knows the truth value of BAD(SMALL_BOWL) each time that the condition needs to be evaluated.

Adding Knowledge and Perceptual Actions

To model the effects of perceptual actions, we must come up with a suitable formalization of knowledge. The approach we take is to adapt the standard possible-world model of knowledge to the situation calculus, as first done by Moore(1980). Informally, we think of there being a binary accessibility relation over situations, where a situation s' is understood as being accessible from a situation s if as far as the agent knows in situation s, he might be in situation s'. So something is known in s if it is true in every s' accessible from s, and conversely something is not known if it is false in some accessible situation.

To treat knowledge as a fluent, we introduce a binary relation $K(s', s)$, read as "s' is accessible from s" and treat it the same way we would any other fluent. In other words, from the point of view of the situation calculus, the last argument to K is the official situation argument (expressing what is known in situation s), and the first argument is just an auxiliary like the y in $\text{BROKEN}(y, s)$.[6]

We can now introduce the notation $\textbf{Knows}(P, s)$ (read as P is known in situation s) as an abbreviation for a formula that uses K. For example

$$\textbf{Knows}(\text{BROKEN}(y), s) \overset{\text{def}}{=} \forall s'\; K(s', s) \rightarrow \text{BROKEN}(y, s').$$

Note that this notation supplies the appropriate situation argument to the fluent on expansion (and other conventions are certainly possible). For the case of equality literals the convention is to supply the situation argument to each non-variable argument of the equality predicate. For example:

$$\begin{aligned}&\textbf{Knows}(\text{NUMBER}(\text{BILL}) = \text{NUMBER}(\text{MARY}), s) \overset{\text{def}}{=}\\ &\qquad \forall s'\; K(s', s) \rightarrow\\ &\qquad \text{NUMBER}(\text{BILL}, s') = \text{NUMBER}(\text{MARY}, s').\end{aligned}$$

This notation can be generalized inductively to arbitrary formulas.

Turning now to knowledge-producing actions, there are two sorts of actions to consider: actions whose effect is to make known the truth value of some formula, and actions to make known the value of some term. A discussion of the second case may be found in (Scherl and Levesque 1993). In the first case, we might imagine a SENSE_P action for a fluent P, such that after doing a SENSE_P, the truth value of P is known. We introduce the notation $\textbf{Kwhether}(\text{P}, s)$ as an abbreviation for a formula indicating that the truth value of a fluent P is known.

$$\textbf{Kwhether}(\text{P}, s) \overset{\text{def}}{=} \textbf{Knows}(\text{P}, s) \vee \textbf{Knows}(\neg\text{P}, s),$$

It will follow from our specification that $\textbf{Kwhether}(\text{P}, do(\text{SENSE}_P, s))$. The specifications of both INSPECT and SNIFF are similar to SENSE_P.

The approach being developed here rests on the specification of a successor state axiom for the K relation. For all situations $do(a, s)$, the K relation will be completely determined by the K relation at s and the action a.

For non-knowledge-producing actions (e.g. $\text{BREAK\_INTO}(p)$), the specification (based on Moore (1980; 1985)) is as follows:

$$\begin{aligned}&Poss(\text{BREAK\_INTO}(p), s) \rightarrow\\ &\quad [K(s'', do(\text{BREAK\_INTO}(p), s)) \equiv\\ &\qquad \exists s'\, (K(s', s) \wedge (s'' = do(\text{BREAK\_INTO}(p), s')))]\end{aligned} \qquad (10)$$

The idea here is that as far as the agent at world s knows, he could be in any of the worlds s' such

[5]Here ACHIEVE(HOLDING(e)) will be instantiated by the sequence of actions that enable the agent to pick up some available egg. These low-level actions have not been defined in this paper.

[6]Note that using this convention means that the arguments to K are reversed from their normal modal logic use.

that $K(s', s)$. At $do(\text{BREAK\_INTO}(p), s)$ as far as the agent knows, he can be in any of the worlds $do(\text{BREAK\_INTO}(p), s')$ for any s' such that $K(s', s)$. So the only change in knowledge that occurs in moving from s to $do(\text{BREAK\_INTO}(p), s)$ is the knowledge that the action BREAK_INTO has been performed.

Now consider the simple case of the knowledge-producing action INSPECT that determines whether or not the fluent BAD is true (following Moore (1980; 1985)).

$$\begin{aligned}&\text{Poss}(\text{INSPECT}(b), s) \rightarrow \\ &\quad [K(s'', do(\text{INSPECT}(b), s)) \equiv \exists s'(K(s', s) \wedge \\ &\quad\quad (s'' = do(\text{INSPECT}(b), s') \wedge \text{Poss}(\text{INSPECT}(b), s')) \\ &\quad\quad\quad \wedge (\text{BAD}(b, s) \equiv \text{BAD}(b, s')))]\end{aligned} \tag{11}$$

Again, as far as the agent at world s knows, he could be in any of the worlds s' such that $K(s', s)$. At $do(\text{INSPECT}(b), s)$ as far as the agent knows, he can be in any of the worlds $do(\text{INSPECT}(b), s')$ for all s' such that $K(s', s)$ and $\text{BAD}(s) \equiv \text{BAD}(s')$. The idea here is that in moving from s to $do(\text{INSPECT}(b), s)$, the agent not only knows that the action $\text{INSPECT}(b)$ has been performed (as above), but also the truth value of the predicate BAD. Observe that the successor state axiom for BAD guarantees that BAD is true at $do(\text{INSPECT}(b), s)$ iff BAD is true at s, and similarly for s' and $do(\text{INSPECT}(b), s')$. Therefore, BAD has the same truth value in all worlds s'' such that $K(s'', do(\text{INSPECT}(b), s))$, and so $\mathbf{Kwhether}(\text{BAD}, do(\text{INSPECT}(b), s))$ is true.

The axiomatization for $\text{SNIFF}(b)$ is exactly the same. The two actions would likely differ in their possibility conditions. For example, the axiomatization of $\text{Poss}(\text{INSPECT}(b))$ may require that the lighting be adequate, while the axiomatization of $\text{Poss}(\text{INSPECT}(b))$ may require that the agent not have a cold.

In the omelette problem, there are two knowledge-producing actions. In general, there may be many. Associated with each knowledge-producing action α_i is a formula $\varphi_i(s, s')$. The form of the successor state axiom for K is then as follows:

Successor State Axiom for K

$$\begin{aligned}&\forall s, s'',\ K(s'', do(a, s)) \equiv \\ &\quad [\exists s'\ (K(s', s) \wedge (s'' = do(a, s'))) \wedge \\ &\quad\quad ((a = \alpha_1) \rightarrow \varphi_1) \wedge \\ &\quad\quad\quad \vdots \\ &\quad\quad ((a = \alpha_n) \rightarrow \varphi_n)))]\end{aligned}$$

The relation K at a particular situation $do(a, s)$ is completely determined by the relation at s and the action a. In (Scherl and Levesque 1993) it is argued that this formulation provides a solution to the frame problem for the situation calculus with knowledge and knowledge-producing actions.

Achieving epistemic goals

Given the conditional[7]

```
if BAD(SMALL_BOWL)
  then THROW_OUT(SMALL_BOWL);
  else POUR(SMALL_BOWL, BIG_BOWL);
```

to execute in the state s, the agent strives to achieve a state s^* so that $\mathbf{Kwhether}(\text{C}(\text{S}), \text{S}^*)$ and $s \leq s^*$ holds. This can be ensured by having the interpreter insert a $\text{ACHIEVE}(\mathbf{Kwhether}(\text{BAD})$ complex action before the test. This amounts to performing planning to achieve the epistemic goal[8].

Given a background theory $\mathcal{D}$[9], a sequence of actions[10] $\alpha_1, ..., \alpha_n$ is *a plan* for the goal $\mathbf{Kwhether}(\varphi, s)$ iff the plan is executable:[11]

$$\mathcal{D} \models Poss([\alpha_1, ..., \alpha_n], s),$$

and after the plan is executed, the agent knows the truth value of φ:

$$\mathcal{D} \models \mathbf{Kwhether}(\varphi[s], do([\alpha_1, ..., \alpha_n], s^*)).$$

In the omelette problem, the GOLOG interpreter would insert a single sense action (either $\text{INSPECT}(b)$ or $\text{SNIFF}(b)$ depending on the physical conditions of both the location and the agent) prior to the test for $\text{BAD}(b)$. The result of executing the interpreter on $Do(\delta, s, s')$ where δ is the omelette plan given earlier

[7] Actually the test for $\neg(\text{NUMBER\_EGGS}(\text{LARGE\_BOWL}) = 3)$ poses a similar problem, but in this particular example it can be shown that no sensing is necessary, i.e. deduction suffices to determine the truth value of the fluent.

[8] We remark that here we are only allowing the agent to construct sequential plans to instantiate the ACHIEVE action. In general one may want to consider producing plans with conditionals and loops. For example, one way to find out whether ONTABLE(A) holds is to move one step; scan the surrounding; if see either the table or the block A, then sense whether ONTABLE(A), else continue move one step .... See (Levesque 1996) for a discussion of the general case from a different perspective.

[9] The theory must include unique names axioms for actions, successor state axioms, axioms about **Knows**, and axioms about the initial state

[10] In this section, the notation $[\alpha_1, ..., \alpha_n]$ is used without formal definition wherever a single action term may occur to represent the sequential application of each action term in the list; beginning with α_1 and ending with α_n.

[11] It may be argued that we should require the agent to have a knowledge of this fact:

$$\mathcal{D} \models \mathbf{Knows}(Poss([\alpha_1, ..., \alpha_n], s), s^*).$$

is a binding for s' — the name of a situation resulting from a successful execution of a sequence of primitive actions that instantiate δ. This sequence of primitive actions will have the sense actions spliced in at the appropriate points so that the agent will always know the truth value of $\textsc{Bad}(b)$ at the point where it would need to test the condition.

In general[12], the issue of what sorts of actions the agent may perform arises. Note that in order to satisfy the preconditions of the perceptual acts the agent may need to alter the state of the world. For some applications it may be necessary to ensure that the plan **A** leave the truth value of the condition C unchanged, i.e., $C(s) \equiv C(do(\mathbf{A}, s))$. For others, this requirement is unnecessary and therefore may preclude finding a plan.

A minimal requirement is that the truth value of $C(s)$ be recoverable. This is addressed by the following proposition:

Proposition 1 *A sequence $\alpha_1, ..., \alpha_n$ of actions is a plan for* $\mathbf{Kwhether}(\varphi(s), s^*)$ *iff*

1. *α_1 is executable in s: $\mathcal{D} \models Poss(\alpha_1, s)$.*
2. *There is a sentence $\varphi'(do(\alpha_1, s))$ that does not mention any state term except $do(\alpha_1, s)$ such that $\mathcal{D} \models \varphi(s) \equiv \varphi'(do(\alpha_1, s))$, and $\alpha_2, ..., \alpha_n$ is a plan for*

$$\mathbf{Kwhether}(\varphi'(do(\alpha_1, s)), s^*).$$

Related Work and Discussion

The epistemic goals considered here are related in many ways to the information goals (Etzioni *et al.* 1992)). For instance, the information goal "determine if the file *paper.tex* contains the word *theorem*" in (Etzioni *et al.* 1992) can be formalized as the following epistemic goal: $\mathbf{Kwhether}(\textsc{Contains}(\text{PAPER.TEX}, \text{THEOREM}, S_0), s)$. Etzioni *et al* address the problem of what the agent may change in achieving information goals.

The information goals in (Etzioni *et al.* 1992) are defined procedurally, and are in many cases stricter than they need to be. For instance, if the agent knows that *paper.tex* contains the string *theorem* iff *paper.tex** does, then to satisfy the goal $\mathbf{Kwhether}(\textsc{Contains}(\text{PAPER.TEX}, \text{THEOREM}, S_0), s)$, the agent is allowed to erase the file *paper.tex* since it knows that as long as the goal goes, it is just as well to use *paper.tex**. In contrast, the corresponding information goal in (Etzioni *et al.* 1992) will forbid the agent to delete *paper.tex*.

[12]This issue does not arise in the omelette program and the condition defined below is trivially satisfied.

We remark that a similar problem (or feature) exists for classical planning. For instance, to satisfy the goal $\textsc{On}(\text{A}, \text{B})$, the agent is allowed to paint the blocks, and not only put the block A on the block B, but also glue them together. There are some possible solutions. One is to require that plans be *justified* (Fink and Yang (1993)) in the sense that they do not contain any unnecessary actions. Another is to strengthen the goals, for instance, to conjoin $\mathbf{Kwhether}(\textsc{Contains}(\text{PAPER.TEX}, \text{THEOREM}, S_0), s)$ with:

$$(\forall s').(S_0 \leq s' \leq s \supset (\forall x).\textsc{Content}(\text{PAPER.TEX}, x, S_0) \\ \equiv \\ \textsc{Content}(\text{PAPER.TEX}, x, s'),$$

which is really a goal of maintenance: Maintain the content of the file PAPER.TEX.

References

Etzioni, Oren; Hanks, Steve; Weld, Daniel; Draper, Denise; Lesh, Neal; and Williamson, Mike 1992. An approach to planning with incomplete information. In Nebel, Bernhard; Rich, Charles; and Swartout, William, editors 1992, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, Cambridge, Massachusetts. 115–125.

Fink, E. and Yang, Q. 1993. Planning with primary effects. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, Cambridge, Massachusetts. 1374–1379.

Lespérance, Yves; Levesque, Hector; Lin, Fangzhen; Marcu, Daniel; Reiter, Ray; and Scherl, Richard 1994. A logical approach to high-level robot programming — a progress report. Appears in *Control of the Physical World by Intelligent Systems*, Working Notes of the 1994 AAAI Fall Symposium, New Orleans, LA.

Lespérance, Yves; Levesque, Hector J.; Lin, F.; Marcu, Daniel; Reiter, Raymond; and Scherl, Richard B. 1995. Foundations of a logical approach to agent programming. To appear in *Proceedings of the IJCAI-95 Workshop on Agent Theories, Architectures, and Languages.*

Levesque, Hector; Reiter, Raymond; Lespérance, Yves; Lin, Fangzhen; and Scherl, Richard B. 1996. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming.* to appear.

Levesque, Hector 1996. What is planning in the presence of sensing? In *AAAI-96, to appear.*

Lin, Fangzhen and Reiter, Raymond 1994. State constraints revisited. *Journal of Logic and Computation* 4(5):655–678.

Moore, R.C. 1980. Reasoning about knowledge and action. Technical Note 191, SRI International.

Moore, R.C. 1985. A formal theory of knowledge and action. In Hobbs, J.R. and Moore, R.C., editors 1985, *Formal Theories of the Commonsense World.* Ablex, Norwood, NJ. 319–358.

Reiter, Raymond 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, Vladimir, editor 1991, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy.* Academic Press, San Diego, CA. 359–380.

Scherl, Richard B. and Levesque, Hector J. 1993. The frame problem and knowledge producing actions. In *Proceedings, Eleventh National Conference on Artificial Intelligence.* 689–695.

Scherl, Richard 1996a. Foundational axioms for the situation calculus with knowledge. unpublished paper.

Scherl, Richard 1996b. Omelettes: Kosher and denver. unpublished paper.