On Knowledge-Based Programming with Sensing in the Situation Calculus

RAY REITER
University of Toronto

We consider a class of knowledge-based Golog programs with sense actions. These programs refer explicitly to an agent's knowledge, and are designed to execute on-line, and under a dynamic closed-world assumption on knowledge. On-line execution of sense actions dynamically updates the background axioms with sentences asserting knowledge of the sense actions' outcomes. We formalize what all this might mean, and show that under suitable assumptions the knowledge modality in such programs can be implemented by provability. This leads to an on-line Golog interpreter for such programs, which we demonstrate on a knowledge-based program with sensing for the blocks world.

Categories and Subject Descriptors: I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods—*representation languages*; *modal logic*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*modal logic*; *computational logic*

General Terms: Languages, Theory

Additional Key Words and Phrases: Situation calculus, situation calculus programming languages, sensing and knowledge, dynamic closed-world assumption, theorem-proving

1. INTRODUCTION

Our concern will be with *knowledge-based programs*, specifically, Golog programs [Levesque et al. 1997] that appeal to knowledge and actions, including sense actions. As an example, we consider the blocks world, and a program that explicitly refers to an agent's knowledge, and to the sense actions she can perform to gain information about the world. We imagine that initially the agent is positioned in front of a table of blocks, with no prior knowledge about which blocks are where. The only information she is given in advance is an enumeration of all the (finitely many) blocks. Of course, the agent needs to know that this is a blocks world, so we include suitable successor state and action precondition axioms. But the agent definitely knows nothing about the initial configuration of blocks. For that matter, neither do we, the program designers. Our program will allow the agent to gain the information she needs, and to carry out the actions required to place all the blocks onto the table:

Author's address: Department of Computer Science, University of Toronto, Toronto, Canada M5S 1A4; email: `reiter@cs.toronto.edu`.

$$
\begin{array}{l}
\textbf{proc } allToTable(b) \\
\quad \textbf{Knows}((\forall x) ontable(x))? \mid \\
\quad (\pi\ x).x \notin b? \ ; \\
\qquad \textbf{if } \neg\textbf{KWhether}(clear(x)) \ \textbf{ then } sense_{clear}(x) \ \textbf{ endIf} \ ; \\
\qquad \textbf{if } \textbf{Knows}(\neg clear(x)) \ \textbf{ then } allToTable(\{x\} \cup b) \\
\qquad \textbf{else if } \neg\textbf{KWhether}(ontable(x)) \ \textbf{ then } sense_{ontable}(x) \ \textbf{ endIf} \ ; \\
\qquad\quad \textbf{if } \textbf{Knows}(ontable(x)) \ \textbf{ then } allToTable(\{x\} \cup b) \\
\qquad\quad \textbf{else } moveToTable(x) \ ; \ allToTable(\{\ \}) \ \textbf{ endIf} \\
\qquad \textbf{endIf} \\
\textbf{endProc}
\end{array}
$$

This procedure appeals to Golog operators that should be intuitively clear, with the possible exception of that for nondeterministic choice, $(\pi\ x)\alpha(x)$, whose reading is "Nondeterministically choose an x, and for that choice, do the program $\alpha(x)$." The parameter b in $allToTable(b)$ stands for the set of those blocks that the agent has considered, and rejected, as candidates for moving to the table since her previous $moveToTable$ action. The test action $x \notin b?$ prevents her from ever reconsidering such a block. Thus, the initial call to this procedure is with $allToTable(\{\ \})$. The procedure assumes two knowledge-producing actions,

$sense_{ontable}(x)$ and $sense_{clear}(x)$,

whose action preconditions are always *true*. This program attempts to minimize the number of *sense* actions the agent performs by first checking that she doesn't currently know the truth value of the sensed fluent. For example, the program fragment **if** $\neg$**KWhether**$(clear(x))$ **then** $sense_{clear}(x)$ **endIf** instructs the agent to sense whether x is clear only when she does not already know whether x is clear.

While on the face of it this program seems perfectly intuitive, there are a number of technical problems lurking behind the scenes, particularly what it would take to implement it:

(1) **Knowledge and lack of knowledge.** The background axiomatization must characterize the agent's knowledge about the initial situation, and also her *lack of knowledge*. So, for example, her ignorance about which blocks are where can be represented by

$$(\forall x, y)\neg\textbf{KWhether}(on(x, y), S_0).$$

Representing lack of knowledge can be problematic when an agent has complex knowledge about the initial situation:

$$
\begin{array}{l}
\textbf{Knows}((\forall x).red(x) \supset ontable(x), S_0), \quad \textbf{Knows}(red(B) \vee red(C), S_0), \\
\textbf{Knows}(on(A, B) \vee on(A, C), S_0), \quad \textbf{Knows}((\exists x)on(x, C), S_0), \\
\textbf{Knows}((\forall x, y, z).on(y, x) \wedge on(z, x) \supset y = z, S_0).
\end{array}
$$

Assuming that the above characterizes all that the agent knows, what does the agent *not* know in this example? Whatever that might be, it must somehow be axiomatized because it does represent a truth about the agent's knowledge. This is a very commonly occurring problem. We begin with a collection $\mathcal{K}$ of axioms about what an agent *does* know, and we want to make a *closed-world assumption about knowledge* to the effect that $\mathcal{K}$ captures everything that the agent knows; any knowledge sentences not following logically from $\mathcal{K}$ are taken to be false. The problem here, of course, is to somehow capture this closed-world assumption in a way that relieves the designer from having to figure out the relevant lack of knowledge axioms when given what the agent does know.

(2) **On-line execution.** Because the program appeals to sense actions, it is designed for on-line execution. This means that it must never backtrack over a sense action; once such an action is performed, it cannot be undone. Knowledge-based programs that invoke information-gathering actions must be carefully designed to prevent execution traces that include sense actions but that eventually lead to dead-ends. The above program has this property.

(3) **Implementing sense actions.** What should be done when such a program encounters a sense action? The program is executing on-line, so that each action in an execution trace is meant to be performed as it is generated during a program run. Performing an ordinary action, like $moveToTable(A)$, is unproblematic; in a setting where the program is controlling a robot, the robot would simply perform the action, and the action term would be added to the situation history being constructed by the program interpreter. Performing a sense action on-line is a different matter. Consider the robot receiving the on-line program request $sense_{clear}(A)$ when the current action log is S. It will respond with one of "yes" or "no", depending on the outcome of its sense action. If "yes" then the robot now knows that A is clear: $\mathbf{Knows}(clear(A), do(sense_{clear}(A), S))$. If "no" then $\mathbf{Knows}(\neg clear(A), do(sense_{clear}(A), S))$. Normally, neither of these facts will be logical consequences of the underlying axioms; they provide new information about the robot's world. Therefore, to provide an on-line implementation of the sense action $sense_{clear}(A)$, dynamically *add* one of

$\mathbf{Knows}(clear(A), do(sense_{clear}(A), S))$,
$\mathbf{Knows}(\neg clear(A), do(sense_{clear}(A), S))$,

to the background axioms, depending on the sense action's actual outcome.

(4) **A knowledge-based Golog interpreter.** For knowledge-based programs like $allToTable$ above, we cannot depend on a conventional Golog interpreter [Levesque et al. 1997], which relies on a closed initial database, and which, in any case, would have to be augmented with the ability to reason about knowledge.

For the above reasons, it is problematic to directly execute knowledge-based programs like $allToTable$ using a closed-world Golog interpreter. The stance we shall take here is to view such programs as *specifications* for agent behaviours, and seek an alternative representation for them that can be executed under standard closed-world Golog.

2. FORMAL PRELIMINARIES

We rely on the description of the situation calculus of [Pirri and Reiter 1999], with specific reference to the successor state axioms and action precondition axioms of basic action theories, and we refer the reader to that paper for the details. We rely also on the approach of [Scherl and Levesque 1993] for representing knowledge and sensing actions in the situation calculus, and we assume that the reader is familiar with their approach. We remind the reader that, following [Moore 1980;

1985], Scherl and Levesque introduce an accessibility relation K into the situation calculus; the intended reading of $K(s', s)$ is that situation s' is accessible from situation s. With the accessibility relation K in hand, one then defines knowledge as an abbreviation:

$$\mathbf{Knows}(\phi, s) \stackrel{def}{=} (\forall s').K(s', s) \supset \phi[s'].$$

Here, ϕ is a situation-suppressed expression, and $\phi[s']$ is that situation calculus formula obtained from ϕ by restoring the situation argument s' to all predicate and function symbols of ϕ that take situation arguments.

2.1 Two Simplifying Assumptions

In the interest of simplifying the presentation of this paper, we shall make two notationally undesirable but otherwise inessential assumptions about the underlying language of the situation calculus:

(1) The language has no functional fluents, which are function symbols that take situation arguments. Non fluent function symbols are permitted. To represent a functional fluent, e.g., $numberOfBlocksOnTable(s)$, the axiomatizer should use a relational fluent, e.g., $numberOfBlocksOnTable(n, s)$, and should enforce, via its truth value in the initial database and via its successor state axiom, that n must always exist and be unique.

(2) Except for the equality predicate, $\sqsubset$ and $Poss$, the language has no non fluent predicate symbols. To represent such "eternal" relation, for example, $isPrimeNumber(n)$, the axiomatizer is required to use a relational fluent, e.g., $isPrimeNumber(n, s)$, and to assign it the successor state axiom

$$isPrimeNumber(n, do(a, s)) \equiv isPrimeNumber(n, s).$$

Moreover, any assertion about $isPrimeNumber(n)$ in the initial database must be made in terms of $isPrimeNumber(n, S_0)$.

2.2 Basic Action Theories with Knowledge and Sensing

Based on Scherl and Levesque's proposal, we can define a *basic action theory* taking the form

$$\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \cup \mathcal{K}_{Init}$$

where,

(1) Σ consists of the following foundational axioms for the situation calculus with knowledge:

Uniqueness of Situations

$$do(a_1, s_1) = do(a_2, s_2) \supset a_1 = a_2 \wedge s_1 = s_2$$

Subhistories

$$\neg s \sqsubset S_0$$

$$s \sqsubset do(a, s') \equiv s \sqsubseteq s'$$

Induction

$$(\forall P).(\forall s)[Init(s) \supset P(s)] \wedge (\forall a, s)[P(s) \supset P(do(a, s))] \supset (\forall s)P(s).$$

Here, we have introduced the abbreviation:

$$Init(s) \stackrel{def}{=} \neg(\exists a, s')s = do(a, s').$$

Any model of these axioms will consist of a forest of isomorphic trees, one rooted at S_0, the others rooted at the other initial situations in the model. All these roots can serve in specifying a K relation over initial situations.

Accessible Initial Situations

$$K(s, s') \supset [Init(s) \equiv Init(s')]. \tag{1}$$

(2) $\mathcal{D}_{ss}$ is a set of successor state axioms including the following axiom for the accessibility relation K:

$$\begin{aligned} &K(s', do(a, s)) \equiv \\ &\quad (\exists s^*).s' = do(a, s^*) \wedge K(s^*, s) \wedge \\ &\qquad (\forall \vec{x}_1)[a = sense_{\psi_1}(\vec{x}_1) \supset \psi_1(\vec{x}_1, s^*) \equiv \psi_1(\vec{x}_1, s)] \wedge \cdots \wedge \\ &\qquad (\forall \vec{x}_m)[a = sense_{\psi_m}(\vec{x}_m) \supset \psi_m(\vec{x}_m, s^*) \equiv \psi_m(\vec{x}_m, s)]. \end{aligned} \tag{2}$$

Here, each sense action $sense_{\psi_i}(\vec{x}_i)$, $i = 1, \ldots, m$, is associated with a condition $\psi_i(\vec{x}_i, s)$ whose truth value in situation s the action is designed to determine. Scherl and Levesque also treat *read* actions, whose purpose is to determine the denotations of functional fluents, but since we are assuming no functional fluents in our situation calculus language, we do not consider these.

It remains only to pin down the permissible syntactic forms that ψ may take in sense actions $sense_\psi(\vec{x})$.

DEFINITION 2.1. *(Objective situation-suppressed expressions).*

(a) If $F(\vec{t}, \sigma)$ is a relational fluent atom, then $F(\vec{t})$ is an objective situation-suppressed expression.

(b) If t_1 and t_2 are terms not of sort *situation*, then $t_1 = t_2$ is an objective situation-suppressed expression.

(c) If ϕ and ψ are objective situation-suppressed expressions, so are $\neg\phi$, $\phi \vee \psi$, and $(\exists v)\psi$, where v is not a *situation* variable.

Objective expressions are statements only about the world, not the agent's mental state; they do not involve expressions of the form **Knows**(ϕ). An *objective situation-suppressed sentence* is an objective situation-suppressed expression without any free variables. In what follows, we shall simply say "ϕ is objective" in place of the long-winded "ϕ is an objective situation-suppressed sentence."

We require that every sense action $sense_\psi(\vec{x})$ be such that $\psi(\vec{x})$ is an objective situation-suppressed expression. So the idea is that an agent can sense objective sentences—truths about the external world—but not, reasonably enough, truths about his own knowledge.

The No Side-Effects Assumption for Sense Actions. For fluents other than K, we suppose that they are provided with successor state axioms in the usual way. But in the presence of sense actions, there is always the possibility that such actions can affect these fluents, as in, for example

$$eyesOpen(do(a, s)) \equiv a = senseForObstacle \vee$$

$$eyesOpen(s) \wedge a \neq closeEyes.$$

However, we will not allow knowledge-producing actions to have such side-effects on ordinary fluents; in the formal story we develop here, such actions are only permitted to affect the K fluent. In other words, for each sense action $sense_\psi$, and each relational fluent R, we require that R's successor state axiom, together with the other background axioms, will entail

$$(\forall \vec{x}, \vec{y}, s) R(\vec{x}, do(sense_\psi(\vec{y}), s)) \equiv R(\vec{x}, s).$$

This no side-effects condition is needed to obtain the above successor state axiom for K. It also guarantees the intuitively necessary property that by virtue of performing a knowledge-producing action, an agent will come to know the outcome of that action. We shall make extensive use of this assumption in what follows.

One might argue that the no side-effects assumption is unreasonable, that sense actions often produce a change in the state of ordinary fluents, as in the above $senseForObstacle$ example. The counter-argument is that indeed certain preconditions (e.g., $eyesOpen$) may be necessary for sense actions to occur, but then separate actions—not sense actions—should be provided by the axioms to achieve these preconditions (e.g., $openEyes$). Then to perform a sense action, one must first perform the appropriate state-changing actions to establish that sense action's preconditions. This is the perspective we adopt here.

(3) $\mathcal{D}_{ap}$ is a set of action precondition axioms.

(4) $\mathcal{D}_{una}$ is the set of unique-names axioms for actions.

(5) $\mathcal{K}_{Init}$ is any set of initial accessibility axioms specifying the K relation in the initial situation; these must have the property that, by virtue of the successor state axiom for K, they will be true in all situations. In particular, using induction, this property can be shown to hold for the following standard accessibility relations in initial situations.

(a) Reflexive in initial situations:

$$(\forall s).Init(s) \supset K(s, s).$$

(b) Symmetric in initial situations:

$$(\forall s, s').Init(s) \wedge Init(s') \supset [K(s, s') \supset K(s', s)].$$

(c) Transitive in initial situations:

$$(\forall s, s', s'').Init(s) \wedge Init(s') \wedge Init(s'') \supset [K(s, s') \wedge K(s', s'') \supset K(s', s'').$$

(d) Euclidean in initial situations:

$$(\forall s, s', s'').Init(s) \wedge Init(s') \wedge Init(s'') \supset [K(s', s) \wedge K(s'', s) \supset K(s', s'')].$$

For example, with reference to the symmetry property, the following is a consequence of the foundational axioms and the [Scherl and Levesque 1993] solution to the frame problem for K:

$$(\forall s, s')[Init(s) \wedge Init(s') \supset [K(s, s') \supset K(s', s)]] \\ \supset (\forall s, s').K(s, s') \supset K(s', s).$$

(6) $\mathcal{D}_{S_0}$ is a set of first-order sentences describing the initial state of the world being axiomatized. To pin these down, we need a couple of definitions.

DEFINITION 2.2. *(Formulas about σ).* Let σ be a term of sort *situation*. The formulas about σ are defined inductively:

(a) A relational fluent atom $F(\vec{t}, \sigma)$ is a formula about σ.

(b) If t_1 and t_2 are terms not of sort *situation*, then $t_1 = t_2$ is a formula about σ.

(c) If ϕ is an admissible situation-suppressed expression (defined below), then $\mathbf{Knows}(\phi, \sigma)$ is a formula about σ.

(d) If ϕ and ψ are formulas about σ, so are $\neg\phi$, $\phi \vee \psi$, and $(\exists v)\psi$, where v is not a *situation* variable.

DEFINITION 2.3. *(Admissible situation-suppressed expressions).* These are inductively defined as follows:

(a) If $F(\vec{t}, \sigma)$ is a relational fluent atom, then $F(\vec{t})$ is an admissible situation-suppressed expression.

(b) If t_1 and t_2 are terms not of sort *situation*, then $t_1 = t_2$ is an admissible situation-suppressed expression.

(c) If ϕ and ψ are admissible situation-suppressed expressions, so are $\neg\phi$, $\phi \vee \psi$, $\mathbf{Knows}(\phi)$, and $(\exists v)\psi$, where v is not a *situation* variable.

Compare this with Definition 2.1 of the objective situation-suppressed expressions. The objective expressions are always admissible, but unlike the former, the latter are permitted to express properties of the agent's mental state, i.e., they may mention expressions of the form $\mathbf{Knows}(\phi)$.

We can now define $\mathcal{D}_{S_0}$ to be any set of sentences about S_0.

EXAMPLE 2.1. Here, we give the action precondition, and successor state axioms for the blocks world that underlies the *allToTable* knowledge-based program.

Action Precondition Axioms

$$Poss(move(x, y), s) \equiv clear(x, s) \wedge clear(y, s) \wedge x \neq y,$$

$$Poss(moveToTable(x), s) \equiv clear(x, s) \wedge \neg ontable(x, s).$$

Successor State Axioms

$$\begin{aligned}
&clear(x, do(a, s)) \equiv \\
&\quad (\exists y)\{[(\exists z)a = move(y, z) \vee a = moveToTable(y)] \wedge on(y, x, s)\} \vee \\
&\quad clear(x, s) \wedge \neg(\exists y)a = move(y, x),
\end{aligned}$$

$$\begin{aligned}
&on(x, y, do(a, s)) \equiv a = move(x, y) \vee \\
&\qquad on(x, y, s) \wedge a \neq moveToTable(x) \wedge \neg(\exists z)a = move(x, z),
\end{aligned}$$

$$\begin{aligned}
&ontable(x, do(a, s)) \equiv a = moveToTable(x) \vee \\
&\qquad\qquad ontable(x, s) \wedge \neg(\exists y)a = move(x, y).
\end{aligned}$$

2.3 Satisfiability of Basic Action Theories with Knowledge

The main result about a basic action theory $\mathcal{D}$ with knowledge is the following:

THEOREM 2.1. (Relative Satisfiability).

(1) $\mathcal{D}$ is satisfiable iff $\mathcal{D}_{S_0} \cup \mathcal{D}_{una} \cup \mathcal{K}_{Init} \cup \{(1)\}$ is satisfiable.

(2) *Moreover, whenever $\mathcal{K}_{Init}$ consists of any subset of the accessibility relations Reflexive, Symmetric, Transitive, Euclidean, $\mathcal{D}$ is satisfiable iff $\mathcal{D}_{S_0} \cup \mathcal{D}_{una} \cup \mathcal{K}_{Init}$ is satisfiable.*

PROOF. Very much like the proof of relative satisfiability for basic action theories without knowledge of [Pirri and Reiter 1999]. Essentially, one shows how a model of $\mathcal{D}_{S_0} \cup \mathcal{D}_{una} \cup \mathcal{K}_{Init}$ can be extended to a model of $\mathcal{D}$. We omit the lengthy details. □

3. REDUCING KNOWLEDGE TO PROVABILITY FOR THE INITIAL SITUATION

The starting point for our implemention of knowledge-based programs is the observation that, in a certain important special case, knowledge is reducible to provability. Here, we describe what we mean by this, and we give suitable conditions under which it will be true. Specifically, we shall be concerned with characterizing entailments of the form $\mathcal{D} \models \mathbf{Knows}(\phi, S_0)$ in terms of provability of knowledge-free sentences.

First we need a simple consequence of the Relative Satisfiability Theorem 2.1:

THEOREM 3.1. *Suppose that $\mathbf{Knows}(\phi, S_0)$ is a sentence about S_0. Then,*

(1) $\mathcal{D} \models \mathbf{Knows}(\phi, S_0)$ *iff* $\mathcal{D}_{S_0} \cup \mathcal{D}_{una} \cup \mathcal{K}_{Init} \cup \{(1)\} \models \mathbf{Knows}(\phi, S_0)$.

(2) *Moreover, whenever $\mathcal{K}_{Init}$ consists of any subset of the accessibility relations Reflexive, Symmetric, Transitive, Euclidean, then*

$$\mathcal{D} \models \mathbf{Knows}(\phi, S_0) \text{ iff } \mathcal{D}_{S_0} \cup \mathcal{D}_{una} \cup \mathcal{K}_{Init} \models \mathbf{Knows}(\phi, S_0).$$

PROOF. $\mathcal{D} \models \mathbf{Knows}(\phi, S_0)$ iff $\mathcal{D} \cup \{\neg\mathbf{Knows}(\phi, S_0)\}$ is unsatisfiable. Observe, that because $\neg\mathbf{Knows}(\phi, S_0)$ is a syntactically legal sentence to include in an initial database, then $\mathcal{D} \cup \{\neg\mathbf{Knows}(\phi, S_0)\}$ is a basic action theory with knowledge whose initial database is $\mathcal{D}_{S_0} \cup \{\neg\mathbf{Knows}(\phi, S_0)\}$. Now use Theorem 2.1. □

Next, we introduce a special class of initial databases. Suppose the sentences of $\mathcal{D}_{S_0}$ are all of the form $\mathbf{Knows}(\kappa_i, S_0), i = 1, \ldots, n$, where each κ_i is objective. In other words, the initial database consists exclusively of sentences declaring what the agent knows about the world he inhabits, but there are no sentences declaring what is actually true of the world, or what he knows about what he knows. So for objective $\kappa_1, \ldots, \kappa_n$, $\mathcal{D}_{S_0} = \{\mathbf{Knows}(\kappa_1, S_0), \ldots, \mathbf{Knows}(\kappa_n, S_0)\}$, and since this is logically equivalent to $\mathbf{Knows}(\kappa_1 \wedge \cdots \wedge \kappa_n, S_0)$, we can simply suppose that $\mathcal{D}_{S_0}$ consists of a single sentence of the form $\mathbf{Knows}(\kappa, S_0)$, where κ is objective, and that is what we shall do from here on.

LEMMA 3.1. *Suppose that ϕ is objective, that $\mathcal{D}_{S_0} = \mathbf{Knows}(\kappa, S_0)$ where κ is objective, and that $\mathcal{K}_{init}$ consists of any subset of the accessibility relations Reflexive, Symmetric, Transitive, Euclidean. Then,*

$$\mathcal{D} \models \mathbf{Knows}(\phi, S_0) \text{ iff } \mathcal{D}_{una} \cup \{\kappa[S_0]\} \models \phi[S_0].$$

PROOF. The $\Leftarrow$ direction follows from the fact that, because the axioms of $\mathcal{D}_{una}$ are situation independent, they are known in S_0, and the fact that all logical consequences of what is known are also known.

($\Rightarrow$). By Theorem 3.1, it is sufficient to prove:

If $\mathcal{K}_{init} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \models \mathbf{Knows}(\phi, S_0)$, then $\mathcal{D}_{una} \cup \{\kappa[S_0]\} \models \phi[S_0]$.

By hypothesis, $\mathcal{K}_{init} \cup \mathcal{D}_{una} \cup \{\mathbf{Knows}(\kappa, S_0), \neg\mathbf{Knows}(\phi, S_0)\}$ is unsatisfiable. Then it must continue to be so with $K(s, s')$ taken to be $s = s'$. With this choice for K, all sentences of $\mathcal{K}_{init}$ become tautologies, $\mathbf{Knows}(\kappa, S_0)$ simplifies to $\kappa[S_0]$, and $\neg\mathbf{Knows}(\phi, S_0)$ simplifies to $\neg\phi[S_0]$. Therefore, $\mathcal{D}_{una} \cup \{\kappa[S_0], \neg\phi[S_0]\}$ is unsatisfiable, and therefore, $\mathcal{D}_{una} \cup \{\kappa[S_0]\} \models \phi[S_0]$. □

Therefore, for the initial situation, we have reduced the entailment problem for knowledge sentences to that of knowledge-free sentences. This result relies on the stated assumptions that:

(1) $\mathcal{D}_{S_0}$ consists of a sentence of the form $\mathbf{Knows}(\kappa, S_0)$, where κ is objective. Therefore, $\mathbf{Knows}((\forall x).clear(x) \vee ontable(x), S_0)$ qualifies; the following would not:

$$(\exists x)\mathbf{Knows}(\kappa(x), S_0), \quad \mathbf{Knows}(\mathbf{Knows}(\kappa), S_0), \quad \neg\mathbf{Knows}(\kappa, S_0),$$
$$\mathbf{Knows}(\kappa_1, S_0) \vee \mathbf{Knows}(\kappa_2, S_0).$$

(2) The sentence to be proved has the form $\mathbf{Knows}(\phi, S_0)$, where ϕ is objective.

Lemma 3.1 gives us a provability story for entailments from $\mathcal{D}$ of the form $\mathbf{Knows}(\phi, S_0)$. What about entailments of the form $\neg\mathbf{Knows}(\phi, S_0)$? We defer treating negative knowledge until Section 5 below, where we shall introduce the *closed-world assumption on knowledge*, whose effect will be that entailments of negative knowledge will reduce to non provability of knowledge-free sentences.

4. ON-LINE EXECUTION OF KNOWLEDGE-BASED PROGRAMS

As discussed earlier, we have in mind executing knowledge-based programs like $allToTable$ on-line. This means that each time a program interpreter adds a new program action to its action history, the robot also physically performs this action. Some of these actions will be sense actions; since these normally increase the robot's knowledge of its world, this means that its axioms must be augmented by knowledge about the outcomes of its on-line sense actions.[1] To capture this idea formally, we need some notation for describing this incrementally growing set of axioms. Initially, before it has performed any actions on-line, the robot's background consists of a basic action theory $\mathcal{D}$, as defined in Section 2.2. Suppose that σ is the current situation recording all the actions performed on-line by the robot. We can suppose that σ mentions no variables, since it makes no sense to perform a non ground action on-line. We want to define the result of augmenting $\mathcal{D}$ with knowledge about the outcomes of all the sense actions occurring in σ.

DEFINITION 4.1. *(Sense Outcome Function).* A sense outcome function is any mapping Ω from ground situation terms to sets of knowledge sentences, such that:

(1) $\Omega(S_0) = \{\,\}$,

(2) If α is not a sense action,

$$\Omega(do(\alpha, \sigma)) = \Omega(\sigma).$$

[1]In this respect, our work has much in common with prior approaches to on-line sensing, for example, [Pirri and Finzi 1999; Lakemeyer 1999].

(3) If α is a sense action $sense_\psi(\vec{g})$,

$$\Omega(do(\alpha,\sigma)) = \Omega(\sigma) \cup \{\mathbf{Knows}(\psi(\vec{g}), do(sense_\psi(\vec{g}),\sigma))\} \text{ or}$$
$$\Omega(do(\alpha,\sigma)) = \Omega(\sigma) \cup \{\mathbf{Knows}(\neg\psi(\vec{g}), do(sense_\psi(\vec{g}),\sigma))\}.$$

In general, we shall be interested in $\mathcal{D} \cup \Omega(\sigma)$, namely, the original basic action theory $\mathcal{D}$, augmented by knowledge about the outcomes of all sense actions in the action history σ, according to the sense outcome function Ω. To analyze the properties of this dynamically growing theory, we need the concept of regression.

4.1 Regression

Regression [Waldinger 1977; Pednault 1994; Pirri and Reiter 1999] is the principal mechanism in the situation calculus for answering queries about hypothetical futures. The intuition underlying regression is this: Suppose we want to prove that a sentence W is entailed by some basic action theory. Suppose further that W mentions a relational fluent atom $F(\vec{t}, do(\alpha,\sigma))$, where F's successor state axiom is $F(\vec{x}, do(a,s)) \equiv \Phi_F(\vec{x},a,s)$. Then we can easily determine a logically equivalent sentence W' by substituting $\Phi_F(\vec{t},\alpha,\sigma)$ for $F(\vec{t}, do(\alpha,\sigma))$ in W. After doing so, the fluent atom $F(\vec{t}, do(\alpha,\sigma))$, involving the complex situation term $do(\alpha,\sigma)$, has been eliminated from W in favour of $\Phi_F(\vec{t},\alpha,\sigma)$, and this involves the simpler situation term σ. In this sense, W' is "closer" to the initial situation S_0 than was W. Moreover, this operation can be repeated until the resulting goal formula mentions only the situation term S_0, after which, intuitively, it should be sufficient to establish this resulting goal using only the sentences of the initial database. Regression is a mechanism that repeatedly performs the above reduction starting with a goal W, ultimately obtaining a logically equivalent goal W_0 whose only situation term is S_0. In [Pirri and Reiter 1999], the soundness and completeness of regression is proved for basic action theories without knowledge and sensing actions. [Scherl and Levesque 1993] defines regression for formulas involving knowledge, but we shall not need that notion in this paper; our definition will be for knowledge-free formulas, and is really a simpler version of that in [Pirri and Reiter 1999].

DEFINITION 4.2. *(The Regressable Formulas).* A formula W is *regressable wrt* a situation term σ iff:

(1) W is a fluent atom $F(\vec{t},\sigma)$, or an equality atom $t_1 = t_2$, where t_1 and t_2 are not situation terms, or
(2) W has the form $\neg W_1$ or $W_1 \vee W_2$ or $(\exists x)W_1$ where x is not a situation variable, and W_1 and W_2 are regressable wrt σ.

W is *regressable* iff it is regressable wrt σ for some ground situation term σ.[2]

We can now define the regression operator for regressable formulas.

DEFINITION 4.3. *(The Regression Operator).* The *regression operator* $\mathcal{R}$ when applied to a regressable formula W is determined relative to a basic theory of actions

[2]This definition is less general than the definition of the regressable formulas given in [Pirri and Reiter 1999]; it has the virtue of being simpler, and it will be sufficient for the purposes of this paper.

that serves as a background axiomatization. In what follows, $\vec{t}$ is a tuple of terms, α is a ground term of sort *action*, and σ is a ground term of sort *situation*.

(1) Suppose W is an atom. Since W is regressable, we have two possibilities:
   (a) W is an equality atom between non situation terms, or W is a fluent atom of the form $F(\vec{t}, S_0)$. In these cases,
   $$\mathcal{R}[W] = W.$$
   (b) W is a relational fluent atom of the form $F(\vec{t}, do(\alpha, \sigma))$. Let F's successor state axiom in $\mathcal{D}_{ss}$ be
   $$F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s).$$
   Without loss of generality, assume that all quantifiers (if any) of $\Phi_F(\vec{x}, a, s)$ have had their quantified variables renamed to be distinct from the free variables (if any) of $F(\vec{t}, do(\alpha, \sigma))$. Then
   $$\mathcal{R}[W] = \mathcal{R}[\Phi_F(\vec{t}, \alpha, \sigma)].$$
   In other words, replace the atom $F(\vec{t}, do(\alpha, \sigma))$ by a suitable instance of the right-hand side of the equivalence in F's successor state axiom, and regress this formula. The above renaming of quantified variables of $\Phi_F(\vec{x}, a, s)$ prevents any of these quantifiers from capturing variables in the instance $F(\vec{t}, do(\alpha, \sigma))$.

(2) For the remaining cases, regression is defined inductively.
$$\mathcal{R}[\neg W] = \neg \mathcal{R}[W],$$
$$\mathcal{R}[W_1 \wedge W_2] = \mathcal{R}[W_1] \wedge \mathcal{R}[W_2],$$
$$\mathcal{R}[(\exists v)W] = (\exists v)\mathcal{R}[W].$$

We shall also need to regress certain kinds of situation-suppressed expressions.

Definition 4.4. *(Regression for situation-suppressed expressions).* Recall that when ϕ is a situation-suppressed expression, and σ a situation term, then $\phi[\sigma]$ is that situation calculus formula obtained from ϕ by restoring the situation argument σ to every predicate of ϕ that takes a situation argument. Let ϕ be an objective situation-suppressed expression. Therefore, ϕ is the result of suppressing the situation argument in some regressable formula of the situation calculus. Introduce a "one-step" regression operator, $\mathcal{R}^1(\phi, \alpha)$, whose role is to perform a single regression step for ϕ through the action α as follows: $\mathcal{R}^1(\phi, \alpha)$ is that situation-suppressed expression obtained from ϕ by first replacing all fluents in $\phi[do(\alpha, s)]$ with situation argument $do(\alpha, s)$ by the corresponding right-hand sides of their successor state axioms (renaming quantified variables if necessary), and next, suppressing the situation arguments in the resulting formula. Clearly, $\mathcal{R}^1(\phi, \alpha)[s]$ is logically equivalent to $\phi[do(\alpha, s)]$ relative to the successor state axioms.

Finally, we define the "multi step" regression operator on situation-suppressed expressions. Let σ be a ground situation term. $\mathcal{R}(\phi, \sigma)$ is the regression of the situation-suppressed expression ϕ through the actions of σ, defined by
$$\mathcal{R}(\phi, S_0) = \phi,$$
$$\mathcal{R}(\phi, do(\alpha, \sigma)) = \mathcal{R}(\mathcal{R}^1(\phi, \alpha), \sigma).$$

We are here overloading the $\mathcal{R}$ notation. The one-argument version, $\mathcal{R}[W]$ regresses regressable formulas W of the situation calculus (to obtain a formula about S_0); the two-argument version $\mathcal{R}(\phi, \sigma)$ regresses situation-suppressed expressions ϕ through the actions of σ. Clearly, $\mathcal{R}(\phi, \sigma)[S_0]$ and $\mathcal{R}[\phi[\sigma]]$ are logically equivalent, relative to the background basic action theory.

4.2 Reducing Knowledge To Provability for On-Line Programs

Here, we focus on conditions under which the theory $\mathcal{D} \cup \Omega(\sigma)$, consisting of the original theory $\mathcal{D}$ augmented by the outcomes of all sense actions in σ, entails sentences of the form $\mathbf{Knows}(\phi, \sigma)$.

LEMMA 4.1. *Let $\phi(\vec{x})$ be an objective, situation-suppressed expression. Then:*

(1) *When $A(\vec{y})$ is not a sense action,*

$$\mathcal{D} \models (\forall \vec{x}, \vec{y}, s).\mathbf{Knows}(\phi(\vec{x}), do(A(\vec{y}), s)) \equiv \mathbf{Knows}(\mathcal{R}^1(\phi(\vec{x}), A(\vec{y})), s).$$

(2)
$$\mathcal{D} \models (\forall \vec{x}, \vec{y}, s).\mathbf{Knows}(\phi(\vec{x}), do(sense_\psi(\vec{y}), s)) \equiv \{\psi(\vec{y}, s) \supset \mathbf{Knows}(\psi(\vec{y}) \supset \phi(\vec{x}), s)\} \wedge \{\neg\psi(\vec{y}, s) \supset \mathbf{Knows}(\neg\psi(\vec{y}) \supset \phi(\vec{x}), s)\}.$$

PROOF. First use the successor state axiom (2) for K. For 1, use the fact that, when $\phi(\vec{x})$ is an objective, situation-suppressed expression,

$$\mathcal{D} \models (\forall \vec{x}, \vec{y}, s).\phi(\vec{x}, do(A(\vec{y}), s)) \equiv \mathcal{R}^1(\phi(\vec{x}), A(\vec{y})), s).$$

For 2, use the fact that, when $\phi(\vec{x})$ is an objective, situation-suppressed expression,

$$\mathcal{D} \models (\forall \vec{x}, \vec{y}, s).\phi(\vec{x}, do(sense_\psi(\vec{y}), s)) \equiv \phi(\vec{x}, s)$$

by the no side-effects assumption for sense actions. □

COROLLARY 4.1.

$$\mathcal{D} \models (\forall \vec{y}, s).\mathbf{Knows}(\pm\psi(\vec{y}), do(sense_\psi(\vec{y}), s)) \equiv \pm\psi(\vec{y}, s) \vee \mathbf{Knows}(\pm\psi(\vec{y}), s)^3$$

PROOF. Take ϕ to be ψ in item 2 of Lemma 4.1. □

COROLLARY 4.2. *When $\mathcal{K}_{Init}$ includes the reflexivity axiom,*

$$\mathcal{D} \models (\forall \vec{y}, s).\mathbf{Knows}(\pm\psi(\vec{y}), do(sense_\psi(\vec{y}), s)) \equiv \pm\psi(\vec{y}, s).$$

PROOF. Use Corollary 4.1, and the fact that when reflexivity holds in the initial situation, it holds in all situations, so that what is known in s is true in s. □

LEMMA 4.2. *When $\mathcal{K}_{Init}$ includes the reflexivity axiom,*

$$\mathcal{D} \models (\forall \vec{x}, \vec{y}, s).\mathbf{Knows}(\pm\psi(\vec{y}), do(sense_\psi(\vec{y}), s)) \supset \mathbf{Knows}(\phi(\vec{x}), do(sense_\psi(\vec{y}), s)) \equiv \mathbf{Knows}(\pm\psi(\vec{y}) \supset \phi(\vec{x}), s).$$

PROOF. By item 2 of Lemma 4.1 and Corollary 4.2. □

We shall need the following bit of notation:

DEFINITION 4.5. $\Omega^{\mathcal{R}}(\sigma)$. Suppose Ω is a sense outcome function, and σ is a ground situation term.

$$\Omega^{\mathcal{R}}(\sigma) = \bigwedge\{\mathcal{R}(\psi(\vec{g}), \sigma') \mid \mathbf{Knows}(\psi(\vec{g}), do(sense_\psi(\vec{g}), \sigma')) \in \Omega(\sigma)\} \wedge$$

[3]The notation $\pm\psi$ occurring in a sentence means that two sentences are being expressed: one in which $\pm\psi$ is uniformly replaced by ψ in the sentence, the other in which it is replaced by $\neg\psi$.

$$\bigwedge\{\mathcal{R}(\neg\psi(\vec{g}),\sigma') \mid \mathbf{Knows}(\neg\psi(\vec{g}), do(sense_\psi(\vec{g}),\sigma')) \in \Omega(\sigma)\}.$$

So, for example, if

$$\Omega(\sigma) = \{\mathbf{Knows}(\psi_1, do(sense_{\psi_1},\sigma_1), \mathbf{Knows}(\neg\psi_2, do(sense_{\psi_2},\sigma_2), \mathbf{Knows}(\neg\psi_3, do(sense_{\psi_3},\sigma_3)\},$$

then

$$\Omega^{\mathcal{R}}(\sigma) = \mathcal{R}(\psi_1,\sigma_1) \wedge \mathcal{R}(\neg\psi_2,\sigma_2) \wedge \mathcal{R}(\neg\psi_3,\sigma_3).$$

Notice that $\Omega^{\mathcal{R}}(\sigma)$ is a situation-suppressed sentence. By convention, $\Omega^{\mathcal{R}}(\sigma) = true$ when $\Omega(\sigma) = \{\ \}$.

LEMMA 4.3. *If $\mathcal{K}_{Init}$ includes the reflexivity axiom, then $\Omega(\sigma)$ and $\Omega^{\mathcal{R}}(\sigma)[S_0]$ are logically equivalent,[4] relative to $\mathcal{D}$. Recall that the notation $\Omega^{\mathcal{R}}(\sigma)[S_0]$ stands for the result of restoring situation argument S_0 back into the situation-suppressed sentence $\Omega^{\mathcal{R}}(\sigma)$.*

PROOF. Corollary 4.2. □

LEMMA 4.4. *Suppose ϕ is objective, σ is a ground situation term, and $\mathcal{K}_{Init}$ includes the reflexivity axiom. Then*

$$\mathcal{D} \cup \Omega(\sigma) \models \mathbf{Knows}(\phi,\sigma) \equiv \mathbf{Knows}(\Omega^{\mathcal{R}}(\sigma) \supset \mathcal{R}(\phi,\sigma), S_0).$$

PROOF. Induction on the number of actions in σ. When this is 0, σ is S_0, and the result is immediate. For the inductive step, there are two cases:

Case 1. Suppose α is not a sense action. Then by item 1 of Lemma 4.1,

$$\mathcal{D} \cup \Omega(do(\alpha,\sigma)) \models \mathbf{Knows}(\phi, do(\alpha,\sigma)) \equiv \mathbf{Knows}(\mathcal{R}^1(\phi,\alpha),\sigma).$$

By induction hypothesis,

$$\mathcal{D} \cup \Omega(\sigma) \models \mathbf{Knows}(\mathcal{R}^1(\phi,\alpha),\sigma) \equiv \mathbf{Knows}(\Omega^{\mathcal{R}}(\sigma) \supset \mathcal{R}(\mathcal{R}^1(\phi,\alpha),\sigma), S_0).$$

The result now follows from the fact that when α is not a sense action, $\Omega^{\mathcal{R}}(\sigma) = \Omega^{\mathcal{R}}(do(\alpha,\sigma))$ and the fact that $\mathcal{R}(\mathcal{R}^1(\phi,\alpha),\sigma) = \mathcal{R}(\phi, do(\alpha,\sigma))$.

Case 2. α is a sense action, say $sense_\psi(\vec{g})$. Without loss of generality, assume that Ω's sense outcome is $\psi(\vec{g})$, so $\Omega(do(\alpha,\sigma)) = \Omega(\sigma) \cup \{\mathbf{Knows}(\psi(\vec{g}), do(\alpha,\sigma))\}$. Then by Lemma 4.2,

$$\mathcal{D} \cup \Omega(do(\alpha,\sigma)) \models \mathbf{Knows}(\phi, do(\alpha,\sigma)) \equiv \mathbf{Knows}(\psi(\vec{g}) \supset \phi, \sigma).$$

By induction hypothesis,

$$\mathcal{D} \cup \Omega(\sigma) \models \mathbf{Knows}(\psi(\vec{g}) \supset \phi,\sigma) \equiv \mathbf{Knows}(\Omega^{\mathcal{R}}(\sigma) \supset \mathcal{R}(\psi(\vec{g}) \supset \phi,\sigma), S_0).$$

The result now follows because $\Omega^{\mathcal{R}}(\sigma) \supset \mathcal{R}(\psi(\vec{g}) \supset \phi,\sigma)$ is the same as $\Omega^{\mathcal{R}}(\sigma) \wedge \mathcal{R}(\psi(\vec{g}),\sigma) \supset \mathcal{R}(\phi,\sigma)$, which is the same as $\Omega^{\mathcal{R}}(do(\alpha,\sigma)) \supset \mathcal{R}(\phi,\sigma)$. □

DEFINITION 4.6. *(Deciding equality sentences).* Suppose T is any situation calculus theory. We say that T decides all equality sentences iff for any sentence β over the language of T whose only predicate symbol is equality, $T \models \beta$ or $T \models \neg\beta$.

[4] We are slightly abusing terminology here; strictly speaking, we should say that the conjunction of the sentences in $\Omega(\sigma)$ is logically equivalent to $\Omega^{\mathcal{R}}(\sigma)[S_0]$.

The following is a purely technical lemma that we shall find very useful in establishing our principal results.

LEMMA 4.5. *Suppose that* $\mathcal{D}_{S_0} = \mathbf{Knows}(\kappa, S_0)$ *where* κ *is objective, that* $\mathcal{D}_{una} \cup \{\kappa[S_0]\}$ *decides all equality sentences, and* $\mathcal{K}_{init}$ *consists of any subset of the accessibility relations Reflexive, Symmetric, Transitive, Euclidean. Suppose further that* $\psi_0, \ldots, \psi_n$ *are objective, and that*

$$\mathcal{D} \models \psi_0[S_0] \vee \mathbf{Knows}(\psi_1, S_0) \vee \cdots \vee \mathbf{Knows}(\psi_n, S_0).$$

Then for some $0 \leq i \leq n$, $\mathcal{D} \models \mathbf{Knows}(\psi_i, S_0)$.

PROOF. This takes a bit of proof theory. By hypothesis,

$$\mathcal{D} \cup \{\mathbf{Knows}(\kappa, S_0), \neg\psi_0[S_0], \neg\mathbf{Knows}(\psi_1, S_0), \ldots, \neg\mathbf{Knows}(\psi_n, S_0)\}$$

is unsatisfiable. Therefore, by item 2 of Theorem 2.1,

$$\mathcal{K}_{init} \cup \mathcal{D}_{una} \cup \{\mathbf{Knows}(\kappa, S_0), \neg\psi_0[S_0], \neg\mathbf{Knows}(\psi_1, S_0), \ldots, \neg\mathbf{Knows}(\psi_n, S_0)\}$$

is unsatisfiable. Therefore, after expanding the **Knows** notation, we have that

$$\begin{aligned}\mathcal{K}_{init} \cup \mathcal{D}_{una} \cup \{&(\forall s).K(s, S_0) \supset \kappa[s], \neg\psi_0[S_0], (\exists s).K(s, S_0) \wedge \neg\psi_1(s), \ldots, \\ &(\exists s).K(s, S_0) \wedge \neg\psi_n(s)\}\end{aligned}$$

is unsatisfiable. Therefore, after skolemizing the existentials, we get that

$$\begin{aligned}&\mathcal{K}_{init} \cup \mathcal{D}_{una} \cup \\ &\{(\forall s).K(s, S_0) \supset \kappa[s], \neg\psi_0[S_0], K(\sigma_1, S_0), \neg\psi_1(\sigma_1), \ldots, K(\sigma_n, S_0), \neg\psi_n(\sigma_n)\}\end{aligned}$$

is unsatisfiable. It remains unsatisfiable if we take $K(s, s')$ to be the complete graph on $\{S_0, \sigma_1, \ldots, \sigma_n\}$, i.e.,

$$K(s, s') = (s = S_0 \vee s = \sigma_1 \vee \cdots \vee s = \sigma_n) \wedge (s' = S_0 \vee s' = \sigma_1 \vee \cdots \vee s' = \sigma_n).$$

With this choice for K, all sentences of $\mathcal{K}_{init}$ become tautologies, and we have that

$$\mathcal{D}_{una} \cup \{\kappa[S_0], \neg\psi_0[S_0]\} \cup \{\kappa[\sigma_1], \neg\psi_1[\sigma_1]\} \cup \cdots \cup \{\kappa[\sigma_n], \neg\psi_n[\sigma_n]\}$$

must be unsatisfiable. Let

$$\{\{\kappa[\sigma_1], \neg\psi_1[\sigma_1]\}, \ldots, \{\kappa[\sigma_m], \neg\psi_m[\sigma_m]\}\}$$

be a minimal subset of

$$\{\{\kappa[\sigma_1], \neg\psi_1[\sigma_1]\}, \ldots, \{\kappa[\sigma_n], \neg\psi_n[\sigma_n]\}\}$$

such that

$$\mathcal{D}_{una} \cup \{\kappa[S_0], \neg\psi_0[S_0]\} \cup \{\kappa[\sigma_1], \neg\psi_1[\sigma_1]\} \cup \cdots \cup \{\kappa[\sigma_m], \neg\psi_m[\sigma_m]\}$$

is unsatisfiable. If $m = 0$, then $\mathcal{D}_{una} \cup \{\kappa[S_0], \neg\psi_0[S_0]\}$ is unsatisfiable, so by Lemma 3.1, the result is immediate. Therefore, we can suppose that $m \geq 1$. By the above minimal subset property,

$$\mathcal{D}_{una} \cup \{\kappa[S_0], \neg\psi_0[S_0]\} \cup \{\kappa[\sigma_1], \neg\psi_1[\sigma_1]\} \cup \cdots \cup \{\kappa[\sigma_{m-1}], \neg\psi_{m-1}[\sigma_{m-1}]\} \quad (3)$$

is satisfiable. By the Craig Interpolation Theorem, there exists a sentence I in the intersection language of (3) and $\mathcal{D}_{una} \cup \{\kappa[\sigma_m], \neg\psi_m[\sigma_m]\}$ such that $(3) \cup \{\neg I\}$ and $\mathcal{D}_{una} \cup \{I, \kappa[\sigma_m], \neg\psi_m[\sigma_m]\}$ are both unsatisfiable. But the intersection language consists of equality sentences only,[5] so I is such a sentence, and since $\mathcal{D}_{una} \cup \{\kappa[S_0]\}$ decides all such sentences, either $\mathcal{D}_{una} \cup \{\kappa[S_0]\} \models I$ or $\mathcal{D}_{una} \cup \{\kappa[S_0]\} \models \neg I$. The latter case is impossible by the satisfiability of (3). We earlier concluded that $\mathcal{D}_{una} \cup \{I, \kappa[\sigma_m], \neg\psi_m[\sigma_m]\}$ is unsatisfiable. It remains unsatisfiable when S_0 is uniformly substituted for σ_m, so that $\mathcal{D}_{una} \cup \{I, \kappa[S_0], \neg\psi_m[S_0]\}$ is unsatisfiable. This, together with the fact that $\mathcal{D}_{una} \cup \{\kappa[S_0]\} \models I$ implies the unsatisfiability of $\mathcal{D}_{una} \cup \{\kappa[S_0], \neg\psi_m[S_0]\}$. Therefore, by Lemma 3.1, $\mathcal{D} \models \mathbf{Knows}(\psi_m, S_0)$. □

Something like the assumption that $\mathcal{D}_{una} \cup \{\kappa[S_0]\}$ decides all equality sentences in the above lemma seems necessary. To see why, consider:

EXAMPLE 4.1. Let F and G be unary fluents, and let κ be the conjunction of the following three sentences:

$$a = b \vee c = d, \quad a = b \supset F, \quad c = d \supset G.$$

Then it is easy to see that

$$\mathbf{Knows}(\kappa, S_0) \models a = b \vee \mathbf{Knows}(F, S_0) \vee \mathbf{Knows}(G, S_0),$$

but

$$\mathbf{Knows}(\kappa, S_0) \not\models \mathbf{Knows}(a = b, S_0),\ \mathbf{Knows}(\kappa, S_0) \not\models \mathbf{Knows}(F, S_0) \text{ and } \mathbf{Knows}(\kappa, S_0) \not\models \mathbf{Knows}(G, S_0).$$

THEOREM 4.1. *Suppose that ϕ is objective, $\mathcal{D}_{S_0} = \mathbf{Knows}(\kappa, S_0)$ where κ is objective, $\mathcal{D}_{una} \cup \{\kappa[S_0]\}$ decides all equality sentences, σ is a ground situation term, and $\mathcal{K}_{Init}$ includes the reflexivity axiom. Then*

$$\mathcal{D} \cup \Omega(\sigma) \models \mathbf{Knows}(\phi, \sigma) \textit{ iff } \mathcal{D}_{una} \cup \{\kappa[S_0], \Omega^{\mathcal{R}}(\sigma)[S_0]\} \models \mathcal{R}[\phi[\sigma]].$$

PROOF. The ($\Leftarrow$) direction follows from Lemmas 3.1 and 4.4.
For the ($\Rightarrow$) direction, suppose $\mathcal{D} \cup \Omega(\sigma) \models \mathbf{Knows}(\phi, \sigma)$. By Lemma 4.4,

$$\mathcal{D} \cup \Omega(\sigma) \models \mathbf{Knows}(\Omega^{\mathcal{R}}(\sigma) \supset \mathcal{R}(\phi, \sigma), S_0).$$

By Lemma 4.3,

$$\mathcal{D} \cup \{\Omega^{\mathcal{R}}(\sigma)[S_0]\} \models \mathbf{Knows}(\Omega^{\mathcal{R}}(\sigma) \supset \mathcal{R}(\phi, \sigma), S_0).$$

Therefore,

$$\mathcal{D} \models \neg\Omega^{\mathcal{R}}(\sigma)[S_0] \vee \mathbf{Knows}(\Omega^{\mathcal{R}}(\sigma) \supset \mathcal{R}(\phi, \sigma), S_0).$$

By Lemma 4.5,

$$\mathcal{D} \models \mathbf{Knows}(\neg\Omega^{\mathcal{R}}(\sigma), S_0) \text{ or } \mathcal{D} \models \mathbf{Knows}(\Omega^{\mathcal{R}}(\sigma) \supset \mathcal{R}(\phi, \sigma), S_0).$$

If the latter is the case, we are done. Suppose the former. Then certainly,

$$\mathcal{D} \models \mathbf{Knows}(\neg\Omega^{\mathcal{R}}(\sigma) \vee \mathcal{R}(\phi, \sigma), S_0),$$

i.e., $\mathcal{D} \models \mathbf{Knows}(\Omega^{\mathcal{R}}(\sigma) \supset \mathcal{R}(\phi, \sigma), S_0)$. The result now follows from Lemma 3.1. □

This is the central result of this section; it completely characterizes entailments of the form $\mathbf{Knows}(\phi, \sigma)$ relative to $\mathcal{D} \cup \Omega(\sigma)$ in terms of provability, in the initial

[5]Recall that we assume the underlying language of the situation calculus has no non fluent predicate symbols and no functional fluents (Section 2.1).

situation, for knowledge-free sentences. What about entailments of sentences of the form $\neg\mathbf{Knows}(\phi, \sigma)$? That is the topic of the next section.

5. THE DYNAMIC CLOSED-WORLD ASSUMPTION

We now consider the problem, discussed in item 1 of Section 1, of characterizing an agent's lack of knowledge, and we begin first by considering knowledge in the initial situation $\mathcal{D}_{S_0} = \mathbf{Knows}(\kappa, S_0)$. Here, we shall make the *closed-world assumption on knowledge*, namely, that $\mathbf{Knows}(\kappa, S_0)$ characterizes *everything* that the agent knows initially, and whatever knowledge does not follow from this will be taken to be lack of knowledge. How can we characterize this closed-world assumption in a way that relieves the axiomatizer from having to figure out the relevant lack of knowledge axioms when given what the agent does know? We propose the following:

$$closure(\mathcal{D}) = \mathcal{D} \cup \{\neg\mathbf{Knows}(\theta, S_0) \mid \theta \text{ is objective and } \mathcal{D} \not\models \mathbf{Knows}(\theta, S_0)\}.$$

Under the closed-world assumption on knowledge, the *official* basic action theory is taken to be $closure(\mathcal{D})$. To make her life easier in specifying the initial database of a basic action theory, we ask the axiomatizer only to provide a specification of the positive initial knowledge $\mathcal{D}_{S_0}$ that the robot has of its domain, but this is understood to be a convenient shorthand for what holds initially, and $closure(\mathcal{D})$, as defined above, specifies the actual basic action theory.

EXAMPLE 5.1. Here, we specify the positive initial knowledge available to the agent inhabiting the blocks world of Example 2.1. While she knows nothing about which blocks are where, this does not mean she knows nothing at all. There are state constraints associated with this world, and we must suppose the agent knows these. Specifically, she must know that these constraints hold initially, and therefore, her initial database consists of the following:

$$\mathbf{Knows}((\forall x, y).on(x, y) \supset \neg on(y, x), S_0),$$
$$\mathbf{Knows}((\forall x, y, z).on(y, x) \wedge on(z, x) \supset y = z, S_0),$$
$$\mathbf{Knows}((\forall x, y, z).on(x, y) \wedge on(x, z) \supset y = z, S_0).$$

By making the closed-world assumption about this initial database, the axiomatizer need not concern herself with writing additional lack of knowledge axioms like $\neg\mathbf{Knows}((\exists x, y)on(x, y), S_0)$, or $\neg\mathbf{Knows}((\exists x).clear(x) \wedge ontable(x), S_0)$. Neither does she have to worry about whether she has succeeded in expressing *all* the relevant lack of knowledge axioms. The closed-world assumption takes care of these problems for her.

However, as noted above, the axioms of $\mathcal{D}$ are being continuously augmented by sentences asserting knowledge about the outcomes of the sense actions that have been performed during the on-line execution of a program, and $\mathcal{D} \cup \Omega(\sigma)$ specifies what these axioms are, when the program is currently in situation σ. Therefore, under the closed-world assumption in situation σ, we are supposing that $\mathcal{D} \cup \Omega(\sigma)$ characterizes all of the agent's positive knowledge about the initial situation, so we are actually interested in the closure of $\mathcal{D}_{S_0}$ relative to $\mathcal{D} \cup \Omega(\sigma)$. So here we are actually making a *dynamic closed-world assumption*.

DEFINITION 5.1. **Dynamic Closed-World Assumption on Knowledge.**

$$closure(\mathcal{D} \cup \Omega(\sigma)) = \\ \mathcal{D} \cup \Omega(\sigma) \cup \{\neg Knows(\theta, S_0) \mid \theta \text{ is objective and } \mathcal{D} \cup \Omega(\sigma) \not\models Knows(\theta, S_0)\}.$$

Under the dynamic closed-world assumption on knowledge, the *official* basic action theory, when the on-line execution of a program is in situation σ, is taken to be $closure(\mathcal{D} \cup \Omega(\sigma))$.

This closed-world assumption on knowledge is a metalevel account of a special case of Levesque's logic of *only knowing* [Levesque 1990]. His results have been considerably extended (to include an account of knowledge within the situation calculus) in [Levesque and Lakemeyer 2001; Lakemeyer and Levesque 1998].

Next, we need to study the properties of $closure(\mathcal{D} \cup \Omega(\sigma))$, with the ultimate objective of reducing negative knowledge to non provability.

LEMMA 5.1. *Suppose that $\mathcal{D}_{S_0} = \mathbf{Knows}(\kappa, S_0)$ where κ is objective, $\mathcal{D}_{una} \cup \kappa[S_0]$ decides all equality sentences, and $\mathcal{K}_{Init}$ includes the reflexivity axiom, together with any subset of the accessibility relations Symmetric, Transitive, Euclidean. Then*

$$closure(\mathcal{D} \cup \Omega(\sigma)) \textit{ is satisfiable iff } \mathcal{D}_{una} \cup \{\kappa[S_0], \Omega^{\mathcal{R}}(\sigma)[S_0]\} \textit{ is satisfiable.}$$

PROOF.
($\Rightarrow$) Suppose $\mathcal{D}_{una} \cup \{\kappa[S_0], \Omega^{\mathcal{R}}(\sigma)[S_0]\}$ is unsatisfiable. Then by Lemma 3.1, $\mathcal{D} \models \mathbf{Knows}(\neg\Omega^{\mathcal{R}}(\sigma), S_0)$. By reflexivity, $\mathcal{D} \models \neg\Omega^{\mathcal{R}}(\sigma)[S_0]$, and therefore, $\mathcal{D} \cup \{\Omega^{\mathcal{R}}(\sigma)[S_0]\}$ is unsatisfiable. Because, by Lemma 4.3, $\Omega^{\mathcal{R}}(\sigma)[S_0]$ and $\Omega(\sigma)$ are logically equivalent (relative to $\mathcal{D}$), $\mathcal{D} \cup \Omega(\sigma)$ is unsatisfiable, and therefore, so is $closure(\mathcal{D} \cup \Omega(\sigma))$.

($\Leftarrow$) Suppose $closure(\mathcal{D} \cup \Omega(\sigma))$ is unsatisfiable. Therefore,

$$\mathcal{D} \cup \Omega(\sigma) \cup \{\neg Knows(\theta, S_0) \mid \theta \text{ is objective and } \mathcal{D} \cup \Omega(\sigma) \not\models Knows(\theta, S_0)\}$$

is unsatisfiable. Therefore, by Lemma 4.3,

$$\mathcal{D} \cup \{\Omega^{\mathcal{R}}(\sigma)[S_0]\} \cup \\ \{\neg Knows(\theta, S_0) \mid \theta \text{ is objective and } \mathcal{D} \cup \Omega(\sigma) \not\models Knows(\theta, S_0)\}$$

is unsatisfiable. By item 2 of Theorem 2.1,

$$\mathcal{K}_{init} \cup \mathcal{D}_{una} \cup \{\mathbf{Knows}(\kappa, S_0), \Omega^{\mathcal{R}}(\sigma)[S_0]\} \cup \\ \{\neg Knows(\theta, S_0) \mid \theta \text{ is objective and } \mathcal{D} \cup \Omega(\sigma) \not\models Knows(\theta, S_0)\}$$

is unsatisfiable. By compactness, there is a finite, possibly empty subset of

$$\{\neg Knows(\theta, S_0) \mid \theta \text{ is objective and } \mathcal{D} \cup \Omega(\sigma) \not\models Knows(\theta, S_0)\},$$

say

$$\{\neg Knows(\theta_1, S_0), \ldots, \neg Knows(\theta_n, S_0)\}$$

such that

$$\mathcal{K}_{init} \cup \mathcal{D}_{una} \cup \{\mathbf{Knows}(\kappa, S_0), \Omega^{\mathcal{R}}(\sigma)[S_0]\} \cup \\ \{\neg Knows(\theta_1, S_0), \ldots, \neg Knows(\theta_n, S_0)\}$$

is unsatisfiable. Therefore,

$$\mathcal{D} \cup \{\Omega^{\mathcal{R}}(\sigma)[S_0]\} \cup \{\neg Knows(\theta_1, S_0), \ldots, \neg Knows(\theta_n, S_0)\}$$

is unsatisfiable, so

$$\mathcal{D} \models \neg\Omega^{\mathcal{R}}(\sigma)[S_0] \vee Knows(\theta_1, S_0) \vee \cdots \vee Knows(\theta_n, S_0).$$

By Lemma 4.5, $\mathcal{D} \models \mathbf{Knows}(\neg\Omega^{\mathcal{R}}(\sigma), S_0)$, or for some i, $\mathcal{D} \models Knows(\theta_i, S_0)$. The latter case is impossible, because for $i = 1, \ldots, n$ $\mathcal{D} \cup \Omega(\sigma) \not\models Knows(\theta_i, S_0)$. Therefore, $\mathcal{D} \models \mathbf{Knows}(\neg\Omega^{\mathcal{R}}(\sigma), S_0)$, and by Lemma 3.1, $\mathcal{D}_{una} \cup \{\kappa[S_0], \Omega^{\mathcal{R}}(\sigma)[S_0]\}$ is unsatisfiable. □

LEMMA 5.2. *Suppose $\mathcal{D}_{S_0} = \mathbf{Knows}(\kappa, S_0)$ where κ is objective, $\mathcal{D}_{una} \cup \kappa[S_0]$ decides all equality sentences, σ is a ground situation term, and $\mathcal{K}_{Init}$ includes the reflexivity axiom. Suppose further that $closure(\mathcal{D} \cup \Omega(\sigma))$ is satisfiable. Then*

$$closure(\mathcal{D} \cup \Omega(\sigma)) \models \mathbf{Knows}(\phi, \sigma) \text{ iff } \mathcal{D}_{una} \cup \{\kappa[S_0], \Omega^{\mathcal{R}}(\sigma)[S_0]\} \models \mathcal{R}[\phi[\sigma]].$$

PROOF. The ($\Leftarrow$) direction follows from Theorem 4.1.

($\Rightarrow$) If $\mathcal{D} \cup \Omega(\sigma) \models \mathbf{Knows}(\phi, \sigma)$, the result is immediate by Theorem 4.1. If $\mathcal{D} \cup \Omega(\sigma) \not\models \mathbf{Knows}(\phi, \sigma)$, then by Lemma 4.4, $\mathcal{D} \cup \Omega(\sigma) \not\models \mathbf{Knows}(\Omega^{\mathcal{R}}(\sigma) \supset \mathcal{R}(\phi, \sigma), S_0)$. Hence, $closure(\mathcal{D} \cup \Omega(\sigma)) \models \neg\mathbf{Knows}(\Omega^{\mathcal{R}}(\sigma) \supset \mathcal{R}(\phi, \sigma), S_0)$. By Lemma 4.4, $closure(\mathcal{D} \cup \Omega(\sigma)) \models \neg\mathbf{Knows}(\phi, \sigma)$, contradicting the hypothesis that $closure(\mathcal{D} \cup \Omega(\sigma))$ is satisfiable. □

DEFINITION 5.2. *(Subjective Sentences).* We say a sentence is a *subjective sentence about* a ground situation term σ iff it has the form $\mathbf{Knows}(\beta, \sigma)$, where β is objective, or it has the form $\neg W$, where W is a subjective sentence about σ, or it has the form $W_1 \vee W_2$, where W_1 and W_2 are subjective sentences about σ.

LEMMA 5.3. *Suppose $\mathcal{K}_{Init}$ includes the reflexivity axiom. Then for any subjective sentence W about a ground situation term σ,*

$$closure(\mathcal{D} \cup \Omega(\sigma)) \models W \text{ or } closure(\mathcal{D} \cup \Omega(\sigma)) \models \neg W.$$

PROOF. Induction on the syntactic form of W, using Lemma 4.4 to help prove the base case. □

We can now combine Lemmas 5.1, 5.2, and 5.3 to obtain our main result:

THEOREM 5.1. *Let Ω be a sense outcome function. Suppose that*

(1) $\mathcal{D}_{S_0} = \mathbf{Knows}(\kappa, S_0)$, where κ is objective.
(2) $\mathcal{D}_{una} \cup \{\kappa[S_0]\}$ decides all equality sentences.
(3) σ is a ground situation term.
(4) $\mathcal{D}_{una} \cup \{\kappa[S_0], \Omega^{\mathcal{R}}(\sigma)[S_0]\}$ is satisfiable.
(5) $\mathcal{K}_{Init}$ consists of the reflexivity axiom, together with any subset of the accessibility axioms Symmetric, Transitive, Euclidean.

Then,

(1) When ϕ is objective,

$$closure(\mathcal{D} \cup \Omega(\sigma)) \models \mathbf{Knows}(\phi, \sigma) \text{ iff } \mathcal{D}_{una} \cup \{\kappa[S_0], \Omega^{\mathcal{R}}(\sigma)[S_0]\} \models \mathcal{R}[\phi[\sigma]].$$

(2) When W is a subjective sentence about σ,

$$closure(\mathcal{D} \cup \Omega(\sigma)) \models \neg W \text{ iff } closure(\mathcal{D} \cup \Omega(\sigma)) \not\models W.$$

(3) When W_1 and W_2 are subjective sentences about σ,

$$closure(\mathcal{D} \cup \Omega(\sigma)) \models W_1 \vee W_2 \text{ iff}$$
$$closure(\mathcal{D} \cup \Omega(\sigma)) \models W_1 \text{ or } closure(\mathcal{D} \cup \Omega(\sigma)) \models W_2.$$

6. AN INTERPRETER FOR KNOWLEDGE-BASED PROGRAMS WITH SENSING

Under the stated conditions, Theorem 5.1 justifies the following decisions in implementing an interpreter for an on-line knowledge-based program.

(1) If $\mathcal{D}_{S_0} = \mathbf{Knows}(\kappa, S_0)$, the implementation uses $\kappa[S_0]$ as its initial database.

(2) Whenever a $sense_\psi(\vec{g})$ action is performed by the program in a situation σ, the implementation adds the regression of $\psi(\vec{g}, \sigma)$ or $\neg\psi(\vec{g}, \sigma)$ to the current initial database, depending on the outcome of the sense action.

(3) Suppose a test condition W is evaluated by the program in a situation σ. Using items 2 and 3 of Theorem 5.1, the implementation recursively breaks down $W[\sigma]$ into appropriate subgoals of proving, or failing to prove, sentences of the form $\mathbf{Knows}(\phi, \sigma)$. By item 1, these reduce to proving, or failing to prove, the regression of $\phi[\sigma]$ relative to the current initial database. So for these base cases, the implementation performs this regression step, then invokes a theorem prover on the regressed sentence, using the current initial database (plus unique names for actions) as premises. Notice the assumption here, required by the theorem, that every test condition W of the program will be such that, at evaluation time, $W[\sigma]$ will be a subjective sentence about σ.

(4) **Guarded Sense Actions.** Condition 4 of Theorem 5.1 requires that $\mathcal{D}_{una} \cup \{\kappa[S_0], \Omega^{\mathcal{R}}(\sigma)[S_0]\}$ be satisfiable. Therefore, an implementation must perform this satisfiability test, and it must do so after each sense action. However, there is one natural condition on a knowledge-based program that would eliminate the need for such a test, and that is that every sensing action in the program be *guarded*. By this, we mean that a sense action is performed only when its outcome is not already known to the robot. In the $allToTable$ program, the statement **if** $\neg\mathbf{KWhether}(clear(x))$ **then** $sense_{clear}(x)$ **endIf** provides such a guard for the $sense_{clear}$ action. Whenever the program guards all its sense actions, as $allToTable$ does, then condition 4 of Theorem 5.1 reduces to requiring the satisfiability of $\mathcal{D}_{una} \cup \{\kappa[S_0]\}$, and this can be performed once only, when the initial database is first specified. This is the content of the following:

PROPOSITION 6.1. *Assume the conditions of Theorem 5.1. Assume further that $sense_\psi(\vec{g})$ is a ground sense action, and that*

$$closure(\mathcal{D} \cup \Omega(\sigma)) \models \neg\mathbf{KWhether}(\psi(\vec{g}), \sigma).$$

Then $\mathcal{D}_{una} \cup \{\kappa[S_0], \Omega^{\mathcal{R}}(do(sense_\psi(\vec{g}), \sigma))[S_0]\}$ is satisfiable.

PROOF. Without loss of generality, assume that Ω's sense outcome is $\psi(\vec{g})$, so

$$\Omega(do(sense_\psi(\vec{g}), \sigma)) = \Omega(\sigma) \cup \{\mathbf{Knows}(\psi(\vec{g}), do(\alpha, \sigma))\}.$$

Since $closure(\mathcal{D} \cup \Omega(\sigma)) \models \neg\mathbf{Knows}(\neg\psi(\vec{g}), \sigma)$ and since $closure(\mathcal{D} \cup \Omega(\sigma))$ is satisfiable,

$$\mathcal{D} \cup \Omega(\sigma)) \not\models \mathbf{Knows}(\neg(\psi(\vec{g}), \sigma).$$

Therefore, by Lemma 4.4,

$$\mathcal{D} \not\models \mathbf{Knows}(\Omega^{\mathcal{R}}(\sigma) \supset \mathcal{R}(\neg\psi(\vec{g}), \sigma), S_0).$$

Hence, by Lemma 3.1,

$$\mathcal{D}_{una} \cup \{\kappa[S_0]\} \not\models \{\Omega^{\mathcal{R}}(\sigma) \supset \mathcal{R}(\neg\psi(\vec{g}), \sigma)\}[S_0],$$

and therefore, $\mathcal{D}_{una} \cup \{\kappa[S_0], \Omega^{\mathcal{R}}(\sigma)[S_0] \wedge \neg\mathcal{R}(\neg\psi(\vec{g}), \sigma)[S_0]\}$ must be satisfiable. The result now follows because $\neg\mathcal{R}(\neg\psi(\vec{g}), \sigma)[S_0]$ is the same thing as $\mathcal{R}(\psi(\vec{g}), \sigma)[S_0]$, and because $\Omega^{\mathcal{R}}(\sigma) \wedge \mathcal{R}(\psi(\vec{g}), \sigma)$ is $\Omega^{\mathcal{R}}(do(sense_{\psi}(\vec{g}), \sigma))$. □

7. COMPUTING CLOSED-WORLD KNOWLEDGE

The reduction of knowledge to provability under the closed-world assumption on knowledge makes little computational sense for full first-order logic, because its provability relation is not computable. Therefore, in what follows, we shall restrict ourselves to the quantifier-free case. Specifically, we shall assume:

(1) The only function symbols not of sort *action* or *situation* are constants.

(2) $\mathcal{D}_{S_0}$ includes knowledge of unique names axioms for these constants:

$$\mathbf{Knows}(C \neq C', S_0), \text{ for distinct constant symbols } C \text{ and } C'.$$

(3) All quantifiers mentioned in $\mathcal{D}_{S_0}$, and all quantifiers mentioned in test expressions of a knowledge-based program are typed, and these types are abbreviations for descriptions of finite domains of constants:

$$\tau(x) \stackrel{def}{=} x = T_1 \vee \cdots \vee x = T_k,$$

where there will be one such abbreviation for each type τ.

Therefore, typed quantifiers can be eliminated in formulas in favour of conjunctions and disjunctions, so we end up with sentences of propositional logic, for which the provability relation is computable. Because the agent has knowledge of unique names, $\mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ will decide all typed equality sentences. Therefore, the conditions of Theorem 5.1 will hold.

8. PUTTING IT ALL TOGETHER

We have implemented an on-line Golog interpreter for knowledge-based programs based on Theorem 5.1. It assumes that all sense actions in the program are guarded, and it therefore does not perform the consistency check required by condition 4 of Theorem 5.1. Here, we include the principal new features that need to be added to a standard Golog interpreter [Reiter 2001] to obtain this knowledge-based program interpreter.

A Golog Interpreter for Knowledge-Based Programs with Sense Actions Using Provability to Implement Knowledge

```
% The clauses for do remain as for standard Golog, except that an extra clause is
% added to treat sense actions by interactively asking the user for the outcome of
% the action, and updating the initial database with the regression of this outcome.
% This clause for sense actions appeals to a user-provided declaration
% senseAction(A,SensedOutcome), meaning that A is a sense action, and SensedOutcome
% is the formula whose truth value the action A is designed to determine.
% restoreSitArgThroughout(W,S,F) means F is the result of restoring the situation
% argument S into every fluent mentioned by the formula W.

do(A,S,do(A,S)) :- senseAction(A,SensedOutcome), poss(A,S),
```

```
     queryUser(SensedOutcome,YN), restoreSitArgThroughout(SensedOutcome,S,Outcome),
     regress(Outcome,R),
     (YN = y, updateInitDatabase(R) ; YN = n, updateInitDatabase(-R)).

queryUser(SensedOutcome,YN) :- nl, write("Is "),
       write(SensedOutcome), write(" true now? y or n."), read(YN).

% Add the following clauses to those for holds in the standard Golog interpreter.

holds(kWhether(W),S) :- holds(knows(W),S), ! ; holds(knows(-W),S).

% Implementing knowledge with provability .

holds(knows(W),S) :- restoreSitArgThroughout(W,S,F), prove(F).
```

In the above interpreter, two Prolog predicates were left unspecified:

(1) The theorem prover `prove`. Any complete propositional prover will do. We use one that supposes the current initial database is a set of prime implicates. There is no significance to this choice; we simply happen to have had available a prime implicate generating program. `prove(F)` first regresses `F`, converts the result to clausal form, then tests these clauses for subsumption against the initial database of prime implicates.

(2) `updateInitDatabase(R)`, whose purpose is to add the sentence `R`, which is the regression of the outcome of a sense action, to the initial database. Because in our implementation this is a database of prime implicates, `updateInitDatabase(R)` converts `R` to clausal form, adds these to the initial database, and recomputes the prime implicates of the resulting database.

Here is an execution of the $allToTable$ program, with a four block domain, using this prover and the above Golog interpreter.[6]

Running the Program for Four Blocks

```
[eclipse 2]: compile. % Compile the initial database to prime implicate form.

Clausal form completed.   CPU time (sec): 0.06  Clauses: 34
Database compiled.   CPU time (sec): 0.04   Prime implicates: 34

yes.
[eclipse 3]: run.

Is clear(a) true now? y or n.   y.

Is ontable(a) true now? y or n. y.

Is clear(b) true now? y or n.   n.

Is clear(c) true now? y or n.   y.

Is ontable(c) true now? y or n. n.
```

[6] All code needed to run this blocks world example is available, on request, from the author.

```
Performing moveToTable(c).

Is clear(b) true now? y or n.   n.

Performing moveToTable(d).

Final situation: [senseClear(a), senseOnTable(a), senseClear(b), senseClear(c),
                  senseOnTable(c), moveToTable(c), senseClear(b), moveToTable(d)]
```

Notice how smart the program is: after learning that `a` is clear and need not be moved, and after moving `c` to the table and learning that `b` is still not clear, it figures that `d` must therefore be on `b` and that `b` must be on the table, so it simply moves `d`.

9. DISCUSSION

The concept of knowledge-based programming was first introduced in [Halpern and Fagin 1989]. Chapter 7 of [Fagin et al. 1995] contains an extensive discussion, with specific reference to the specification of communication protocols. Knowledge-based programs play a prominent role in the literature on agent programming; see, for example, [Shoham 1993].

The closed-world assumption on knowledge is also made in [de Giacomo et al. 1996], where they reduce the entailment problem for knowledge to entailment of knowledge-free sentences. Their work differs from ours in two essential ways: theirs is an epistemic description logic, and their epistemic modality is for the purposes of planning, not knowledge-based programming.

This paper is a close relative of [Pirri and Finzi 1999]. There, Pirri and Finzi give a mechanism for on-line execution of action sequences, with sense actions. Their concept of sense actions and their outcomes is more sophisticated than ours, partly because they allow for perceptions that may conflict with an agent's theory of the world, but in one special case—the so-called *safe* action sequences—their treatment of a sense actions outcome is the same as ours: update the initial database with the regression of this outcome. Their concept of safety also corresponds closely to our notion of guarded sense actions. The basic differences between us is that our action theories are formulated with knowledge, while theirs are not, and we are interested in the on-line execution of programs, not just action sequences. Nevertheless, it seems that one way of viewing (some of) the results of this paper is as a specification of agent behaviours in terms of knowledge and the closed-world assumption, for which the Pirri-Finzi account is a provably correct implementation. But this possibility raises so many issues, they are best dealt with in future work.

10. POSTSCRIPT

It is particularly gratifying to be able to provide this paper in honour of Bob Kowalski's 60th birthday because he has long been an advocate of metalevel reasoning, especially for reasoning about modalities. So although Bob has not been a great fan of the situation calculus,[7] I expect he would approve of my reduction to provability

[7]But who knows, perhaps he has had a change of heart recently; turning 60 does have certain mellowing effects.

of the knowledge modality. I do hope so, because this is my birthday gift to him, and there is no exchange policy where it came from.

REFERENCES

DE GIACOMO, G., IOCCHI, L., NARDI, D., AND ROSATI, R. 1996. Moving a robot: The KR&R approach at work. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96)*, L. Aiello, J. Doyle, and S. Shapiro, Eds. Morgan Kaufmann Publishers, San Francisco, CA, 198–209.

FAGIN, R., HALPERN, J., MOSES, Y., AND VARDI, M. 1995. *Reasoning about Knowledge.* MIT Press, Cambridge, Mass.

HALPERN, J. AND FAGIN, R. 1989. Modelling knowledge and action in distributed systems. *Distributed Computing 3,* 4, 159–179.

LAKEMEYER, G. 1999. On sensing and off-line interpreting in GOLOG. In *Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter*, H. Levesque and F. Pirri, Eds. Springer, 173–189.

LAKEMEYER, G. AND LEVESQUE, H. June 1998. AOL: a logic of acting, sensing, knowing, and only knowing. In *Proc. of KR-98, Sixth International Conference on Principles of Knowledge Representation and Reasoning.* 316–327.

LEVESQUE, H. 1990. All I know: A study in autoepistemic logic. *Artificial Intelligence 42*, 263–309.

LEVESQUE, H. AND LAKEMEYER, G. 2001. *The Logic of Knowledge Bases.* MIT Press. to appear.

LEVESQUE, H., REITER, R., LESPÉRANCE, Y., LIN, F., AND SCHERL, R. 1997. GOLOG: a logic programming language for dynamic domains. *J. of Logic Programming, Special Issue on Actions 31,* 1-3, 59–83.

MOORE, R. 1980. Reasoning about knowledge and action. Tech. rep., SRI International. Technical Note 191.

MOORE, R. 1985. A formal theory of knowledge and action. In *Formal Theories of the Commonsense World*, J. B. Hobbs and R. C. Moore, Eds. Ablex Publishing Corp., Norwood, New Jersey, Chapter 9, 319–358.

PEDNAULT, E. 1994. ADL and the state-transition model of action. *J. Logic and Computation 4,* 5, 467–512.

PIRRI, F. AND FINZI, A. 1999. An approach to perception in theory of actions: Part 1. *Linköping Electronic Articles in Computer and Information Science 4.* http://www.ep.liu.se/ea/cis/1999/041/.

PIRRI, F. AND REITER, R. 1999. Some contributions to the metatheory of the situation calculus. *Journal of the ACM 46,* 3, 261–325.

REITER, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* MIT Press, Cambridge, Mass.

SCHERL, R. AND LEVESQUE, H. 1993. The frame problem and knowledge producing actions. In *Proc. AAAI-93.* Washington, DC, 689–695.

SHOHAM, Y. 1993. Agent oriented programming. *Artificial Intelligence 60,* 1, 51–92.

WALDINGER, R. 1977. Achieving several goals simultaneously. In *Machine Intelligence 8*, E. Elcock and D. Michie, Eds. Ellis Horwood, Edinburgh, Scotland, 94–136.