On planning with programs that sense

Jorge A. Baier and **Sheila A. McIlraith**
Department of Computer Science
University of Toronto
Toronto, Canada.
{jabaier,sheila}@cs.toronto.edu

Abstract

In this paper we address the problem of planning by composing *programs*, rather than or in addition to primitive actions. The programs that form the building blocks of such plans can, themselves, contain both sensing and world-altering actions. Our work is primarily motivated by the problem of automated Web service composition, since Web services are programs that can sense and act. Our further motivation is to understand how to exploit macro-actions in existing operator-based planners that plan with sensing. We study this problem in the language of the situation calculus, appealing to Golog to represent our programs. To this end, we propose an offline execution semantics for Golog programs with sensing. We then propose a compilation method that transforms our action theory with programs into a new theory where programs are replaced by primitive actions. This enables us to use state-of-the-art, operator-based planning techniques to plan with programs that sense for a restricted but compelling class of programs. Finally, we discuss the applicability of these results to existing operator-based planners that support sensing and illustrate the computational advantage of planning with programs that sense via an experiment. The work presented here is cast in the situation calculus to facilitate formal analysis. Nevertheless, both the results and the algorithm can be trivially modified to take PDDL as input and output. This work has broad applicability to planning with programs or macro-actions with or without sensing.

1 Introduction

Classical planning takes an initial state, a goal state and an action theory as input and generates a sequence of actions that, when performed starting in the initial state, will terminate in a goal state. Typically, actions are primitive and are described in terms of their precondition, and (conditional) effects. Classical planning has been extended to planning with sensing actions, where the agent uses such actions to gather information from its environment. In this setting, plans are usually conditional, and in practice they are much harder to generate than in the classical setting.

Our interest here is in using *programs*, rather than or in addition to primitive actions, as the building blocks for plans. The programs that we consider may both sense and act in the world. Our approach is to develop a technique for compiling programs into new primitive actions that can be exploited by standard operator-based planning techniques. To achieve this, we automatically extract (knowledge) preconditions and (knowledge) effects from programs. We study this problem in the language of the situation calculus, appealing to Golog to represent our programs.

Our primary motivation for investigating this topic is to address the problem of automated *Web service composition* (WSC) (e.g., (McIlraith & Son 2002)). Web services are self-contained, Web-accessible computer programs, such as the airline ticket service at www.aircanada.com, or the weather service at www.weather.com. These services are indeed programs that sense—e.g. by determining the balance of an account or flight costs by querying a database—and act in the world—e.g. by arranging for the delivery of goods, by debiting accounts, etc. As such, the task of WSC can be conceived as the task of planning with programs, or as a specialized version of a program synthesis task.

A secondary motivation for this work is to improve the efficiency of planning with sensing by representing useful (conditional) plan segments as programs. Planning with some form of macro-actions (e.g., (Fikes, Hart, & Nilsson 1972; Sacerdoti 1974; Korf 1987; McIlraith & Fadel 2002; Erol, Hendler, & Nau 1994)) can dramatically improve the efficiency of plan generation. The advantage of using macro-actions in general, are two fold. By encapsulating programs into operators plan length is reduced, which in turn significantly reduces the search space. Also, operators provide a means of achieving a subtask without search. This leads to increased performance of planners.

Levesque (1996) argued that when planning with sensing, the outcome of the planning process should be a plan which the executing agent knows at the outset will lead to a final situation in which the goal is satisfied. Even in cases where no uncertainty in the outcome of actions, and no exogenous actions are assumed this remains challenging because of incomplete information about the initial state. To plan effectively with programs, we must consider whether we have the knowledge to actually execute the program prior to using it in a plan. To that end, in Section 3 we propose an offline execution semantics for Golog programs with sensing that enables us to determine that we know how to execute a program. We prove the equivalence of our semantics to the original Golog semantics, under certain conditions.

Then, in Section 4.1, we propose a compilation method that transforms our action theory with programs into a new theory where programs are replaced by primitive actions. This enables us to use traditional operator-based planning techniques to plan with programs that sense in a restricted but compelling set of cases. In Section 5 we discuss the applicability of these results to existing operator-based planners that allow sensing. Finally, in Section 6 we discuss the practical relevance of this work by illustrating the potential computational advantages of planning with programs that sense. We also discuss the relevance of this work to Web service composition.

2 Preliminaries

The situation calculus and Golog provide the theoretical foundations for our work. In the two subsections that follow we briefly review the situation calculus (McCarthy & Hayes 1969; Reiter 2001), including a treatment of sensing actions and knowledge. We also review the transition semantics for Golog, a high-level agent programming language that we employ to represent the programs we are composing. For those familiar with the situation calculus and Golog, we draw your attention to the decomposition of successor state axioms for the K fluent leading to Proposition 1, the perhaps less familiar distinction of deterministic tree programs, and the definition of the $Trans^-$ and Do^-predicates.

2.1 The situation calculus

The situation calculus, as described by Reiter (2001), is a second-order language for specifying and reasoning about dynamical systems. In the situation calculus, the world changes as the result of *actions*. A *situation* is a term denoting the history of actions performed from an initial distinguished situation, S_0. The function $do(a,s)$ denotes the situation that results from performing action a in situation s[1]. Relational *fluents* (resp. functional fluents) are situation-dependent predicates (resp. functions) that capture the changing state of the world. The distinguished predicate $Poss(a,s)$ is used to express that it is possible to execute action a in situation s. The dynamics of a particular domain is described by *action theories*.

Reiter's basic action theory has the form $\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \cup \mathcal{K}_{init}$, (Reiter 2001, pg. 305) where,

- Σ is a set of foundational axioms.
- $\mathcal{D}_{ss}$ is a set of successor state axioms (SSAs), of the form:

$$F(\vec{x}, do(a,s)) \equiv \Phi_F(a, \vec{x}, s). \tag{1}$$

  The set of SSAs can be compiled from a set of *effect axioms*, $\mathcal{D}_{eff}$ (Reiter 2001). An effect axiom describes the effect of an action on the truth value of certain fluents, e.g., $a = startCar \supset engineStarted(do(a,s))$. $\mathcal{D}_{ss}$ may also contain SSAs for functional fluents.
- $\mathcal{D}_{ap}$ contains action precondition axioms.
- $\mathcal{D}_{una}$ contains unique names axioms for actions.
- $\mathcal{D}_{S_0}$ describes the initial state of the world.
- $\mathcal{K}_{init}$ defines the properties of the K fluent in the initial situations and preserved in all situations.

[1]$do([a_1,\ldots,a_n],s)$ abbreviates $do(a_n, do(\ldots, do(a_1,s)\ldots))$.

Following Scherl & Levesque (2003), we use the distinguished fluent K to capture the knowledge of an agent in the situation calculus. The K fluent reflects a first-order adaptation of Moore's possible-world semantics for knowledge and action (Moore 1985). $K(s',s)$ holds iff when the agent is in situation s, she considers it possible to be in s'. Thus, we say that a first-order formula ϕ is *known* in a situation s if ϕ holds in every situation that is K-accessible from s. For notational convenience, we adopt the abbreviations[2]

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

$$\mathbf{KWhether}(\phi,s) \stackrel{\text{def}}{=} \mathbf{Knows}(\phi,s) \lor \mathbf{Knows}(\neg\phi,s).$$

To define properties of the knowledge of agents we can define restrictions over the K fluent. One common restriction is reflexivity (i.e., $(\forall s)\,K(s,s)$) which implies that everything that is known by the agent in s is also true in s.

Scherl & Levesque (2003) define a standard SSA for the K fluent. Given sensing actions $a_1,\ldots,a_n$ such that a_i $(1 \le i \le n)$ senses whether or not formula ψ_i is true, $\mathcal{D}_{ss}$ contains:

$$K(s', do(a,s)) \equiv (\exists s'').\,s' = do(a,s'') \land K(s'',s) \land \bigwedge_{i=1}^{n} \{a = a_i \supset (\psi_i(s) \equiv \psi_i(s''))\}. \tag{2}$$

Intuitively, when performing a non-sensing action a in s, if s'' was K-accessible from s then so is $do(a,s'')$ from $do(a,s)$. However, if sensing action a_i is performed in s and s'' was K-accessible from s then $do(a_i,s'')$ is K-accessible from $do(a_i,s)$ only if s and s'' agree upon the truth value of ψ_i.

In contrast to Scherl & Levesque (2003), we assume that the SSA for K is compiled from a set of *sufficient condition axioms*, $\mathcal{K}_s$, rather than simply given. We do this to be able to cleanly modify the SSA for K without appealing to syntactic manipulations. If $a_1,\ldots,a_n$ are sensing actions and each action a_i senses formula ψ_i, the axiomatizer must provide the following sufficient condition axioms for each a_i,

$$K(s'',s) \land a = a_i \land (\psi_i(s) \equiv \psi_i(s'')) \supset K(do(a,s''), do(a,s)), \tag{3}$$

which intuitively express the same dynamics of the K-reachability for situations as (2) but with one axiom for each action. Furthermore, in order to model the dynamics of the K-reachability for the remaining non-sensing actions, the following axiom must be added:

$$K(s'',s) \land \bigwedge_{i=1}^{n} a \neq a_i \supset K(do(a,s''), do(a,s)). \tag{4}$$

Axioms (3) and (4) can be shown to be equivalent to the SSA of K when one assumes that all sufficient conditions are also necessary.

Proposition 1 *Predicate completion on axioms of the form* (3) *and* (4) *is equivalent to the SSA for K defined in* (2).

[2]We assume ϕ is a *situation-suppressed* formula (i.e. a situation calculus formula whose situation terms are suppressed). $\phi[s]$ denotes the formula that restores situation arguments in ϕ by s.

Finally, our compilation procedure will make extensive use of Reiter's *regression operator* (Reiter 2001). The regression of $\alpha = \varphi(do([a_1,\dots,a_n],S_0))$, denoted by $\mathcal{R}[\alpha]$, is a formula equivalent to α but such that the only situation terms occurring in it are S_0. To regress a formula, one iteratively replaces each occurrence of $F(x,do(a,s))$ by the right-hand side of equation (1) until all atomic subformulae mention only situation S_0. In our compilation, we use the regression operator $\mathcal{R}^s$, which does the same as $\mathcal{R}$ but "stops" when all situation terms mentioned in the formula are s.

2.2 Golog's syntax and semantics

Golog is a high-level agent programming language whose semantics is based on the situation calculus (Reiter 2001). A Golog program is a complex action[3] potentially composed from:

- nil – the empty program
- a – primitive action
- $\phi?$ – test action
- $\pi x.\,\delta$ – nondeterministic choice of argument
- $\delta_1;\delta_2$ – sequences (δ_1 is followed by δ_2)
- $\delta_1|\delta_2$ – nondeterministic choice between δ_1 and δ_2
- **if** ϕ **then** δ_1 **else** δ_2 **endif** – conditional
- **while** ϕ **do** δ **endW** – loop

Below we will propose a compilation algorithm for Golog programs that are *deterministic tree* programs.

Definition 1 (Tree program) *A Golog tree program is a Golog program that does not contain any loop constructs.*

Definition 2 (Deterministic program) *A deterministic Golog program is one that does not contain any nondeterministic choice construct.*

The restriction to tree programs may seem strong. Nevertheless, in practical applications most loops in terminating programs can be replaced by a bounded loop (i.e. a loop that is guaranteed to end after a certain number of iterations). Therefore, following McIlraith & Fadel (2002), we extend the Golog language with a *bounded loop* construct. Thus, we define **while**$_k$ ϕ **do** δ **endW** as equal to nil if $k=0$ and equal to **if** ϕ **then** $\{\delta;$**while**$_{k-1}$ ϕ **do** δ **endW**$\}$ **else** nil **endif**, for $k>0$. We include this as an admissible construct for a tree program.

Golog has both an evaluation semantics (Reiter 2001) and a transition semantics (de Giacomo, Lespérance, & Levesque 2000). The transition semantics is defined in terms of single steps of computation, using two predicates *Trans* and *Final*. $Trans(\delta,s,\delta',s')$ is true iff when a single step of program δ is executed in s, it ends in the situation s', with program δ' remaining to be executed, and $Final(\delta,s)$ is true if program δ terminates in s. Some axioms for *Trans* and *Final* are shown below.

$$Final(\delta,s) \equiv \delta = nil,$$

$$Trans(a,s,\delta',s') \equiv Poss(a,s) \wedge \delta' = nil \wedge s' = do(a,s), \qquad (5)$$

$$Trans(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ endif},\delta,s,\delta',s') \equiv$$
$$\phi[s] \wedge Trans(\delta_1,s,\delta',s') \vee \neg\phi[s] \wedge Trans(\delta_2,s,\delta',s').$$

[3]We use the symbol δ to denote complex actions. ϕ is a situation-suppressed formula.

Using the transitive closure of *Trans*, $Trans^*$, the predicate $Do(\delta,s,s')$ can be defined to be true iff program δ, executed in situation s, terminates in situation s'.

In the rest of the paper, we also consider a less restrictive definition of *Trans*, $Trans^-$. $Trans^-(\delta,s,\delta',s')$ is true iff program δ can make a transition to s' by possibly violating an action precondition. The definition of $Trans^-$ is exactly the same as that of *Trans* save replacing the right-hand side of (5) by $\delta' = nil \wedge s' = do(a,s)$. Likewise, we define the Do^- predicate in terms of $Trans^-$. We introduce this definition merely for efficiency; in our translation process we will regress Do^- to obtain simpler formulae. The following is a simple relationship between *Do* and Do^-.

Proposition 2 *Let δ be a Golog program and let $\mathcal{T}$ be the axioms defining Golog's transition semantics. Then,*

$$\mathcal{T} \models Do(\delta,s,s') \supset Do^-(\delta,s,s')$$

Proof: By induction in the structure of δ.

3 Semantics for executable Golog programs

Again, as Levesque (1996) argued, when planning with sensing, the outcome of the planning process should be a plan which the executing agent knows at the outset will lead to a final situation in which the goal is satisfied. When planning with programs, as we are proposing here, we need to be able to determine when it is possible to execute a program with sensing actions and what situations could be the result of the program. Unfortunately, Golog's original semantics does not consider sensing actions and furthermore does not consider whether the agent has the ability to execute a given program.

Example 1 Consider a theory $\mathcal{D}$ and Golog transition semantics axioms $\mathcal{T}$ such that $\mathcal{D}\cup\mathcal{T} \models \phi[S_0]$ and $\mathcal{D}\cup\mathcal{T} \not\models \neg\phi[S_0]$, and let $\Delta \overset{\text{def}}{=}$ **if** ϕ **then** a **else** b **endif**. Assume furthermore that a and b are always possible. Then, it holds that $\mathcal{D}\cup\mathcal{T} \models (\exists s)Do(\Delta,S_0,s)$, i.e. δ is executable in S_0 (in fact, $\mathcal{D}\cup\mathcal{T} \models Do(\Delta,S_0,s) \equiv s = do(a,S_0) \vee s = do(b,S_0)$). This fact is counter-intuitive since in S_0 the agent does not have enough information to determine whether ϕ holds, so Δ is not really executable.

As a *first objective* towards planning with programs that sense, we define what property a Golog program must satisfy to ensure it will be executable. Our *second objective* is to define a semantics that will enable us to determine the family of situations resulting from executing a program with sensing actions. This semantics provides the foundation for results in subsequent sections.

To achieve our first objective, we need to ensure that at each step of program execution, an agent has all the knowledge necessary to execute that step. In particular, we need to ensure that the program is *epistemically feasible*. Once we define the conditions underwhich a program is epistemically feasible, we can either use them as constraints on the planner, or we can ensure that our planner only builds plans using programs that are known to be epistemically feasible at the outset.

Several papers have addressed the problem of knowing how to execute a plan (Davis 1994) or more specifically, a Golog program. Lespérance *et al.* (2000) define a predicate *CanExec* to establish when a program can be executed by an agent. Sardina *et al.* (2004) define epistemically feasible programs using the online semantics of de Giacomo & Levesque (1999). Finally, a simple definition is given by McIlraith & Son (2002), which defines a self-sufficient property, *ssf*, such that $ssf(\delta,s)$ is true iff an agent knows how to execute program δ in situation s. We appeal to this property to characterize when a Golog program is executable. Its definition is given below.

$$ssf(nil) \stackrel{\text{def}}{=} True,$$

$$ssf(a,s) \stackrel{\text{def}}{=} \mathbf{KWhether}(Poss(a),s),$$

$$ssf(\pi x.\,\delta,s) \stackrel{\text{def}}{=} (\exists x)\, ssf(\delta,s),$$

$$ssf(\delta_1|\delta_2,s) \stackrel{\text{def}}{=} ssf(\delta_1,s) \wedge ssf(\delta_2,s),$$

$$ssf(\phi?,s) \stackrel{\text{def}}{=} \mathbf{KWhether}(\phi,s),$$

$$ssf(\delta_1;\delta_2,s) \stackrel{\text{def}}{=} ssf(\delta_1,s) \wedge (\forall s').\, Trans^*(\delta_1,s,nil,s') \supset ssf(\delta_2,s'),$$

$$ssf(\mathbf{if}\ \phi\ \mathbf{then}\ \delta_1\ \mathbf{else}\ \delta_2\ \mathbf{endif},s) \stackrel{\text{def}}{=} \mathbf{KWhether}(\phi,s) \wedge (\phi[s] \supset ssf(\delta_1,s)) \wedge (\neg\phi[s] \supset ssf(\delta_2,s)),$$

$$ssf(\mathbf{while}\ \phi\ \mathbf{do}\ \delta\ \mathbf{endW},s) \stackrel{\text{def}}{=} \mathbf{KWhether}(\phi,s) \wedge (\phi[s] \supset ssf(\delta,s) \wedge (Trans^*(\delta,s,nil,s') \supset ssf(\mathbf{while}\ \phi\ \mathbf{do}\ \delta\ \mathbf{endW},s'))).$$

We now focus on our second objective, i.e. to define a semantics for Golog programs with sensing actions. To our knowledge, no such semantics exists. Nevertheless, there is related work. de Giacomo & Levesque (1999) define the semantics of programs with sensing in an *online* manner, i.e. it is determined during the execution of the program. An execution is formally defined as a mathematical object, and the semantics of the program depends on such an object. The semantics is thus defined in the metalanguage, and therefore it is not possible to refer to the situations that would result from the execution of a program within the language.

To define a semantics for executable programs with sensing, we modify the existing Golog transition semantics so that it refers to the knowledge of the agent, defining two new predicates $Trans_K$ and $Final_K$ as follows.

$$Final_K(\delta,s) \equiv Final(\delta,s),$$

$$Trans_K(nil,s,\delta',s') \equiv False,$$

$$Trans_K(\phi?,s,\delta',s') \equiv \mathbf{Knows}(\phi,s) \wedge \delta' = nil \wedge s' = s,$$

$$Trans_K(a,s,\delta',s') \equiv \mathbf{Knows}(Poss(a),s) \wedge \delta' = nil \wedge s' = do(a,s),$$

$$Trans_K(\delta_1|\delta_2,s,\delta',s') \equiv Trans_K(\delta_1,s,\delta',s') \vee Trans_K(\delta_2,s,\delta',s'),$$

$$Trans_K(\delta_1;\delta_2,s,\delta',s') \equiv (\exists\sigma)\,(\delta' = \sigma;\delta_2 \wedge Trans_K(\delta_1,s,\sigma,s')) \vee Final_K(\delta_1,s) \wedge Trans_K(\delta_2,s,\delta',s'),$$

$$Trans_K(\pi v.\,\delta,s,\delta',s') \equiv (\exists x)\, Trans_K(\delta_x,s,\delta',s'),$$

$$Trans_K(\mathbf{if}\ \phi\ \mathbf{then}\ \delta_1\ \mathbf{else}\ \delta_2\ \mathbf{endif},\delta,s,\delta',s') \equiv \mathbf{Knows}(\phi,s)\wedge Trans_K(\delta_1,s,\delta',s') \vee \mathbf{Knows}(\neg\phi,s) \wedge Trans_K(\delta_2,s,\delta',s'),$$

$$Trans_K(\mathbf{while}\ \phi\ \mathbf{do}\ \delta\ \mathbf{endW},\delta,s,\delta',s') \equiv \mathbf{Knows}(\neg\phi,s) \wedge s = s' \wedge \delta' = nil \vee \mathbf{Knows}(\phi,s) \wedge Trans_K(\delta;\mathbf{while}\ \phi\ \mathbf{do}\ \delta\ \mathbf{endW},s,\delta',s').$$

We define $Do_K(\delta,s,s') \stackrel{\text{def}}{=} (\exists\delta')\, Trans^*_K(\delta,s,\delta',s') \wedge Final_K(\delta',s')$, analogously to the definition of *Do*. Henceforth we use $\mathcal{T}$ to refer to a set containing all of Golog's transition semantics axioms.

In contrast to *Trans*, $Trans_K$ of an *if-then-else* explicitly requires the agent to know the value of the condition. Returning to Example 1, if now $\mathcal{D} \cup \mathcal{T} \not\models \mathbf{KWhether}(\phi,S_0)$, then $\mathcal{D} \cup \mathcal{T} \models \neg(\exists s)\, Do_K(\Delta,S_0,s)$. However, if $sense_\phi$ senses ϕ, then $\mathcal{D} \cup \mathcal{T} \models (\exists s)\, Do_K(sense_\phi;\Delta,S_0,s)$.

A natural question to ask is when this semantics is equivalent to the original semantics. We can prove that both are equivalent for self-sufficient programs (in the sense of McIlraith & Son (2002)).

Lemma 1 *Let $\mathcal{D}$ be a theory of action such that $\mathcal{K}_{init}$ contains the reflexivity axiom for K. Then,*

$$\mathcal{D} \cup \mathcal{T} \models (\forall\delta,s).\, ssf(\delta,s) \supset \{(\forall s').\, Do(\delta,s,s') \equiv Do_K(\delta,s,s')\}$$

Proof: By induction in the structure of δ. □

The preceding lemma is fundamental to the rest of our work. In the following sections we show how theory compilation relies strongly on the use of regression of the Do_K predicate. Given our equivalence we can now regress *Do* instead of Do_K which produces significantly simpler formulae.

An important point is that the equivalence of the semantics is achieved for self-sufficient programs. Proving that a program is self-sufficient may be as hard as doing the regression of Do_K. Fortunately, there are syntactic accounts of self-sufficiency (McIlraith & Son 2002; Sardina *et al.* 2004), such as programs in which each *if-then-else* and *while* loop that conditions on ϕ is preceded by a $sense_\phi$, or more generally that ϕ is established prior to these constructs and persists until their usage.

4 Planning with programs that sense

We now return to the main objective of this paper – how to plan with programs that sense by enabling operator-based planners to treat programs as black-box primitive actions. A plan in the presence of sensing is a program that may contain conditionals and loops (Levesque 1996). As such, we define a plan as a Golog program.

Definition 3 (A plan) *Given a theory of action $\mathcal{D}$, and a goal G we say that Golog program δ is a plan for situation-suppressed formula G in situation s relative to theory $\mathcal{D}$ iff* $\mathcal{D} \cup \mathcal{T} \models (\forall s').\, Do_K(\delta,s,s') \supset G[s']$.

In classical planning, a planner constructs plan δ by choosing actions from a set A of primitive actions. Here, the planner has an additional set C of programs from which to construct plans.

Example 2 Consider an agent that uses the following complex action to paint objects:

$$\delta(o) \stackrel{\text{def}}{=} sprayPaint(o); look(o); \mathbf{if}\ \neg wellPainted(o)\ \mathbf{then}\ brushPaint(o)\ \mathbf{else}\ nil\ \mathbf{endif}$$

The action $sprayPaint(o)$ paints an object o with a spray gun, and action $brushPaint(o)$ paints it with a brush. We assume that action $sprayPaint(o)$ well-paints o if the spray is not malfunctioning, whereas action $brushPaint(o)$ always well-paints o (this agent prefers spray-painting for cosmetic reasons). Action $look(o)$ is a sense action that senses whether or not o is well painted.

Below we show some axioms in $\mathcal{D}_{ap}$ and $\mathcal{D}_{eff}$ that are relevant for our example.

$$Poss(sprayPaint(o),s) \equiv have(o),$$
$$Poss(look(o),s) \equiv have(o),$$
$$a = sprayPaint(o) \wedge \neg malfunct(s) \supset wellPainted(o,do(a,s)),$$
$$a = brushPaint(o) \supset wellPainted(o,do(a,s))$$

The SSAs for the fluents *wellPainted* and K are as follows.

$$\begin{aligned}&wellPainted(x,do(a,s)) \equiv\\ &\quad a = brushPaint(x) \vee (a = sprayPaint(x) \wedge \neg malfunct(s)) \vee\\ &\quad wellPainted(x,s) \wedge a \neq scratch(x),\end{aligned}$$

$$\begin{aligned}&K(s',do(a,s)) \equiv (\exists s'').\, s' = do(a,s'') \wedge K(s'',s) \wedge\\ &\quad \{(\forall x).\, a = look(x) \supset\\ &\qquad (wellPainted(x,s'') \equiv wellPainted(x,s))\}.\end{aligned}$$

The SSA for *wellPainted* says that x is well painted if it has just been brush painted, or it has just been spray painted and the spray is not malfunctioning or if it was well painted in the preceding situation and x has not been scratched. On the other hand, the SSA for K talks about a unique sensing action, $look(x)$, which senses whether x is well painted.

Suppose we want to use action δ to construct a plan using an operator-based planner. Instead of a program, we would need to consider δ's effects and preconditions (i.e. we would need to represent δ as a primitive action). Among the effects we must describe both physical effects (e.g., after we perform $\delta(B)$, B is *wellPainted*) and knowledge effects (e.g., if we know that o is not *wellPainted*, after we perform $\delta(B)$, we know whether or not *malfunct*!).

The rest of this section presents a method that, under certain conditions, transforms a theory of action $\mathcal{D}$ and a set of programs with sensing C into a new theory, $\mathsf{Comp}[\mathcal{D},C]$, that describes the same domain as $\mathcal{D}$ but that is such that programs in $\mathsf{Comp}[\mathcal{D},C]$ each appear modeled by a new primitive action.

4.1 Theory compilation

A program with sensing may produce both effects in the world and in the knowledge of the agent. Therefore, we want to replace a program δ by one primitive action $prim_\delta$, with the same preconditions and the same physical and knowledge effects as δ. We now describe how we can generate a new theory of action that contains this new action. Then we prove that when $prim_\delta$ is executed, it captures all world and knowledge-level effects of the original program δ.

Our translation process is restricted to tree programs. This is because programs containing loops can be problematic since they may have arbitrarily long executions. However, in many practical cases loops can be replaced by bounded loops using the **while**$_k$ construct introduced in Section 2.

We start with a theory $\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \cup \mathcal{K}_{init} \cup \mathcal{T}$, describing a set $\mathcal{A}$ of primitive actions and a set C of tree programs, and we generate a new theory $\mathsf{Comp}[\mathcal{D},C]$ that contains new, updated SSA, precondition and unique name axioms.

We assume that the set of successor state axioms, $\mathcal{D}_{ss}$, has been compiled from sets $\mathcal{D}_{eff}$ and $\mathcal{K}_s$. Furthermore, assume we have a set of Golog tree programs C which may contain sensing actions such that for every $\delta \in C$ it holds that $\mathcal{D} \models (\forall s).\, ssf(\delta,s)$.

Intuitively, since $prim_\delta$ replaces δ, we want $prim_\delta$ to be executable exactly when δ is executable in s. Note that action δ is executable in s iff there exists a situation s' such that the action theory entails $Do_k(\delta,s,s')$. Moreover, we want $prim_\delta$ to preserve the physical effects of δ. To that end, for each fluent, we add effect axioms for $prim_\delta$ such that whenever δ makes F true/false, $prim_\delta$ will also make it true/false. Finally, because we want to preserve knowledge effects of δ, $prim_\delta$ will emulate δ with respect to the K fluent. To write these new axioms we use the regression operator $\mathcal{R}^s$ of Section 2 because we will need that precondition and effect axioms only talk about situation s. We generate the new theory $\mathsf{Comp}[\mathcal{D},C]$ in the following way.

1. Make $\mathcal{D}'_{eff} := \mathcal{D}_{eff}$, $\mathcal{D}'_{ap} := \mathcal{D}_{ap}$, $\mathcal{K}'_s := \mathcal{K}_s$, and $\mathcal{D}'_{una} := \mathcal{D}_{una}$.
2. For each $\delta(\vec{y}) \in C$, we add the following precondition axioms to $\mathcal{D}'_{ap}$,
$$Poss(prim_\delta(\vec{y}),s) \equiv \mathcal{R}^s[(\exists s')\, Do(\delta(\vec{y}),s,s')]$$
therefore, $prim_\delta(\vec{y})$ can be executed iff program δ could be executed in s.
3. For each relational fluent $F(\vec{x},s)$ in the language of $\mathcal{D}$ that is not the K fluent, and each complex action $\delta(\vec{y}) \in C$ we add the following effect axioms to $\mathcal{D}'_{eff}$:
$$\begin{aligned}a = prim_\delta(\vec{y}) \wedge \mathcal{R}^s[(\exists s')\,(Do^-(\delta(\vec{y}),s,s')\\ \wedge F(\vec{x},s'))] \supset F(\vec{x},do(a,s)),\end{aligned} \tag{6}$$
$$\begin{aligned}a = prim_\delta(\vec{y}) \wedge \mathcal{R}^s[(\exists s')\,(Do^-(\delta(\vec{y}),s,s')\\ \wedge \neg F(\vec{x},s'))] \supset \neg F(\vec{x},do(a,s)).\end{aligned} \tag{7}$$
The intuition here is that F must be true (resp. false) after executing $prim_\delta$ in s if after executing δ in s it is true (resp. false).
4. For each functional fluent $f(\vec{x},s)$ in the language of $\mathcal{D}$, and each program $\delta(\vec{y}) \in C$ we add the following effect axiom to $\mathcal{D}'_{eff}$:
$$\begin{aligned}a = prim_\delta(\vec{y}) \wedge \mathcal{R}^s[(\exists s')\,(Do^-(\delta(\vec{y}),s,s') \wedge\\ z = f(\vec{x},s'))] \supset z = f(\vec{x},do(a,s)),\end{aligned}$$
5. For each $\delta(\vec{y}) \in C$, we add the following sufficient condition axiom to $\mathcal{K}'_s$:
$$\begin{aligned}a =& prim_\delta(\vec{y}) \wedge s'' = do(a,s') \wedge\\ &\mathcal{R}^{s'}[\mathcal{R}^s[(\exists s_1,s_2)\,(Do^-(\delta(\vec{y}),s,s_1) \wedge\\ &Do^-(\delta(\vec{y}),s',s_2) \wedge K(s_2,s_1))]] \supset K(s'',do(a,s)).\end{aligned} \tag{8}$$

Intuitively, suppose s' is K accessible from s, and that performing δ in s' leads to s_2, whereas performing δ in s leads to s_1. Then, if s_2 is K-accessible from s_1, we want that $do(prim_\delta, s')$ be K-accessible from $do(prim_\delta, s)$.

6. For each $\delta, \delta' \in C$ such that $\delta \neq \delta'$ add $prim_\delta(\vec{y}) \neq prim_{\delta'}(\vec{y}')$ to $\mathcal{D}'_{una}$. For each $\alpha(\vec{x}) \in \mathcal{A}$, add $\alpha(\vec{x}) \neq prim_\delta(\vec{y})$ to $\mathcal{D}'_{una}$.
7. Compile a new set of SSAs $\mathcal{D}'_{ss}$ from $\mathcal{D}'_{eff}$ and $\mathcal{K}'_s$. The new theory is defined as follows.

$$\mathsf{Comp}[\mathcal{D}, C] = \Sigma \cup \mathcal{D}'_{ss} \cup \mathcal{D}'_{ap} \cup \mathcal{D}'_{una} \cup \mathcal{D}_{S_0} \cup \mathcal{K}_{init} \cup \mathcal{T}.$$

The reader very familiar with the situation calculus may have noticed that the formula $(\exists s')\, Do^-(\delta(\vec{y}), s, s')$—as well as others in the compilation—is not directly regressable (Reiter 2001, pg. 62). However it is simple to get around this technical difficulty. This is justified by the following two results.

Proposition 3 *Let δ be a tree program, then there exists a set of action sequences $\{\vec{a}_i\}$, and a set of formulae $\{\psi_i(\vec{y}, s)\}$, where each $\psi_i(\vec{y}, s)$ is regressable in s (in particular, it only mentions the situation variable s), such that:*

$$\mathcal{T} \models Do^-(\delta, s, s') \equiv \bigvee_i \psi_i(\vec{y}, s) \wedge s' = do(\vec{a}_i, s)$$

Proof: By induction in the number of constructs of δ. The proof also defines a way to construct ψ_i. □

Corollary 1 *There is a procedure to construct regressable equivalent formulae for $(\exists s')\,(Do^-(\delta(\vec{y}), s, s')$, $(\exists s_1, s_2)\,(Do^-(\delta(\vec{y}), s, s_1) \ \wedge Do^-(\delta(\vec{y}), s', s_2) \wedge K(s_2, s_1))$, and $(\exists s')\,(Do^-(\delta(\vec{y}), s, s') \wedge [\neg]F(s))$.*

Proof: Replace each of the Do^-'s by its equivalent formula following proposition 3. Now it is possible to eliminate the existential quantifiers in each of the formulae, and a regressable formula is obtained. □

We now turn to the analysis of some properties of the resulting theory $\mathsf{Comp}[D, C]$.

Theorem 1 *If $\mathcal{D}$ is consistent and C contains only deterministic tree programs then* $\mathsf{Comp}[\mathcal{D}, C]$ *is consistent.*

Proof: We prove that the SSAs of $\mathsf{Comp}[\mathcal{D}, C]$ are consistent following (Reiter 2001, pg. 31). □

Indeed, if C contains one non-deterministic tree program, we cannot guarantee that $\mathsf{Comp}[\mathcal{D}, C]$ is consistent. Now we establish a complete correspondence at the physical level between our original programs and the compiled primitive actions after performing $prim_\delta$.

Theorem 2 *Let $\mathcal{D}$ be a theory of action such that $\mathcal{K}_{init}$ contains the reflexivity axiom. Let C be a set of deterministic Golog tree programs. Finally, let $\phi(\vec{x})$ be an arbitrary situation-suppressed formula that does not mention the K fluent. Then,*

$$\mathsf{Comp}[\mathcal{D}, C] \models (\forall s, s', \vec{x}).\, Do_K(\delta, s, s') \supset \\ (\phi(\vec{x})[s'] \equiv \phi(\vec{x})[do(prim_\delta, s)])$$

Proof: See appendix.

It is worth noting that the preceding theorem is also valid when δ does not contain sensing actions.

Also, there is a complete correspondence at a knowledge level between our original complex actions and the compiled primitive actions after performing $prim_\delta$.

Theorem 3 *Let $\mathcal{D}$ be a theory of action such that $\mathcal{K}_{init}$ contains the reflexivity axiom. Let C be a set of deterministic Golog tree programs, and $\phi(\vec{x})$ be a situation-suppressed formula that does not mention the K fluent. Then,*

$$\mathsf{Comp}[\mathcal{D}, C] \models (\forall \vec{x}, s, s_1).\, Do_K(\delta, s, s_1) \supset \\ \{\mathbf{Knows}(\phi(\vec{x}), s_1) \equiv \mathbf{Knows}(\phi(\vec{x}), do(prim_\delta, s))\}.$$

Proof: See the appendix.

Now that we have established the correspondence between $\mathcal{D}$ and $\mathsf{Comp}[\mathcal{D}, C]$ we return to planning. In order to achieve a goal G in a situation s, we now obtain a plan using theory $\mathsf{Comp}[\mathcal{D}, C]$. In order to be useful, this plan should have a counterpart in $\mathcal{D}$, since the executor cannot execute any of the "new" actions in $\mathsf{Comp}[\mathcal{D}, C]$. The following result establishes a way to obtain such a counterpart.

Theorem 4 *Let $\mathcal{D}$ be a theory of action, C be a set of deterministic Golog tree programs, and G be a formula of the situation calculus. Then, if Δ is a plan for G in theory* $\mathsf{Comp}[D, C]$ *and situation s, then there exists a plan Δ' for G in theory $\mathcal{D}$ and situation s. Moreover, Δ' can be constructed from Δ.*

Proof sketch: We construct Δ' by replacing every occurrence of $prim_\delta$ in Δ by δ. Then we prove that Δ' also achieves the goal, from theorems 2 and 3. □

Example 2 (cont.) The result of applying theory compilation to the action theory of our example follows. The precondition axiom obtained for $prim_\delta$ is

$$Poss(prim_\delta(o), s) \equiv have(o, s).$$

For the fluent *wellPainted*, the negative effect axiom

$$a = prim_\delta(o) \wedge \mathcal{R}^s[(\exists s')\,(Do^-(\delta(o), s, s') \wedge \\ \neg wellPainted(o, s'))] \supset \neg wellPainted(o, do(a, s)),$$

simplifies to

$$a = prim_\delta(o) \wedge \\ \mathcal{R}^s[(\neg wellPainted(o, S_3) \wedge \neg wellPainted(o, S_1)) \vee \\ (wellPainted(o, S_3) \wedge \neg wellPainted(o, S_2))] \supset \\ \neg wellPainted(o, do(a, s)),$$

where $S_1 = do([sprayPaint(o), look(o), brushPaint(o)], s)$, $S_2 = do([sprayPaint(o), look(o)], s)$, and $S_3 = do([sprayPaint(o)], s)$, which finally simplifies into the futile axiom $a = prim_\delta(o) \wedge False \supset \neg wellPainted(o, do(a, s))$. Analogously, the positive effect axiom obtained for *wellPainted* is, $a = prim_\delta(o) \supset wellPainted(o, do(a, s))$. The resulting SSA for *wellPainted* is:

$$wellPainted(o, do(a, s)) \equiv \\ a = brushPaint(o) \vee a = prim_\delta(o) \vee \\ a = sprayPaint(o) \wedge \neg malfunct(s) \vee \\ wellPainted(o, s) \wedge a \neq scratch(o),$$

which means that o is well painted after $prim_\delta(o)$ is performed.

For K, we obtain the following SSA,

$$\begin{aligned}&K(s', do(a,s)) \equiv (\exists s'').\, s' = do(a,s'') \wedge K(s',s) \wedge \\ &\quad \{a = look(o) \supset (wellPainted(o,s'') \equiv wellPainted(o,s))\} \wedge \\ &\quad \{a = prim_\delta(o) \supset (\psi(o,s'') \equiv \psi(o,s))\},\end{aligned}$$

where $\psi(o,s) \stackrel{\text{def}}{=} malfunct(s) \wedge \neg wellPainted(o,s)$. The axiom for K reflects the fact that $prim_\delta(o)$ is obtaining the truth value of ψ.

Clearly, the process has captured the world-altering effect of $\delta(o)$, namely that $wellPainted(o)$. Moreover, it is easy to confirm a conditional knowledge effect: $\mathbf{Knows}(\neg wellPainted(o), s) \supset \mathbf{KWhether}(malfunct, do(prim_\delta(o), s))$.

Note that our theory compilation can only be used for complex actions that can be proved self-sufficient for all situations. As noted previously, an alternative was to use the conditions that need to hold true for a program to be self-sufficient as a precondition for the newly generated primitive actions. Indeed, formula $ssf(\delta, s)$ encodes all that is required to hold in s to be able to know how to execute δ, and therefore we could have added $Poss(prim_\delta(\vec{y}), s) \equiv \mathcal{R}^s[(\exists s')\, Do(\delta, s, s') \wedge ssf(\delta, s)]$ in step 2 of theory compilation. This modification keeps the validity of our theorems but the resulting precondition expression may contain complex formulae referring to the knowledge of the agent, which we view as problematic for practical applications. The good news is that most Web services are self-sufficient by design.

Finally, the compilation method we have described here is only defined for programs that contain primitive actions, i.e. it does not allow programs to invoke other programs. However, the method can be extended for a broad class of programs that include such calls. If there are no unbounded recursions or the programs can be stratified with respect to recursive calls, it is always possible to iteratively apply the compilation method presented until all programs have been reduced to a primitive action.

5 From theory to practice

We have shown that under certain circumstances, planning with programs can be in theory reduced to planning with primitive actions. In this section we identify properties necessary for operator-based planners to exploit these results, with particular attention to some of the more popular existing planners. There are several planning systems that have been proposed in the literature that are able to consider the knowledge of an agent and (in some cases) sensing actions. These include Sensory Graphplan (SGP) (Weld, Anderson, & Smith 1998), the MDP-based planner GPT (Bonet & Geffner 2000), the model-checking-based planner MBP[4] (Bertoli *et al.* 2001), the logic-programming-based planner $\pi(\mathcal{P})$ (Son, Tu, & Baral 2004), the knowledge-level planner PKS (Petrick & Bacchus 2002), and Contingent FF (CFF) (Hoffmann & Brafman 2005).

All of these planners are able to represent conditional effects of physical actions, therefore, the representation of the physical effects of $prim_\delta$ is straightforward. Unfortunately, the representation of the knowledge effects of $prim_\delta$ is not trivial in some cases. Indeed, without loss of generality, suppose that C contains only one program $\delta(\vec{y})$. After theory compilation, the SSA for the K fluent in $\mathsf{Comp}[\mathcal{D}, C]$ has the general form:

$$\begin{aligned}&K(s', do(a,s)) \equiv \\ &\qquad (\exists s'').\, s' = do(a,s'') \wedge K(s'',s) \wedge \varphi(s) \wedge \\ &\qquad \bigwedge_j \{(\forall \vec{y}).\, a = prim_\delta(\vec{y}) \wedge \alpha_j(\vec{y},s) \supset \\ &\qquad\qquad \bigwedge_i \beta_{ij}(\vec{y},s) \equiv \beta_{ij}(\vec{y},s'')\}, \qquad (9)\end{aligned}$$

where $\varphi(s)$ describes the knowledge effect for the original actions in $\mathcal{D}$, and therefore does not mention the action term $prim_\delta$. Intuitively, as before, β_{ij} are the (regressed) properties that are sensed and α_j are the (regressed) conditions of if-then-else constructs that had to be true for the program to sense β_{ij}.

From the syntax of K, we determine the following requirements for achieving planning with programs that sense in practical planners.

1. The planner must be able to represent *conditional* sensing actions. These are the α_j formulae appearing in (9).
2. The planner must be able to represent that $prim_\delta$ senses the truth value of, in general, arbitrary formulae. This is because β_{ij} in (9) could be any first-order formula.

Most of the planners do not satisfy these requirements directly. However, in most cases one can modify the planning domain, and still plan with our compiled actions. Below we show how this can be done.

Belief-state-based planners

All the planners we investigated, except PKS, are in this category. They represent explicitly or implicitly all the states in which the agent could be during the execution of the plan (sometimes called *belief states*). They are propositional and cannot represent functions[5]. In our view, the expressiveness of these planners is extremely restrictive, especially because they are unable to represent functions of arbitrary range, which is of great importance in many practical applications including WSC.

Among the planners investigated, SGP is the only one that cannot be adapted to achieve requirement 1. The reason is that sensing actions in SGP cannot have preconditions or conditional effects. Others ($\pi(\mathcal{P})$, MBP) can be adapted to simulate conditional sensing actions by splitting $prim_\delta$ into several actions with different preconditions.

Regarding requirement 2, SGP and MBP can handle arbitrary (propositional) observation formulae. However, all the

[4]MBP does not consider sensing actions explicitly, however they can be 'simulated' by representing within the state the last action executed.

[5]GPT can indeed represent functions, but with limited, integer range.

remaining planners are only able to sense propositions (GPT, $\pi(\mathcal{P})$, PKS, and CFF).

In GPT, or in any other propositional planner able to handle actions that have both physical and knowledge effects, this limitation can be overcome by adding two extra fluents for each $prim_\delta$ action. For each formula β_{ij}, add the the fluents F_{ij} and G_{ij} to the compiled theory. Fluent $F_{ij}(\vec{y},s)$ is such that its truth value is equivalent to that of formula $\beta_{ij}(\vec{y},s)$. The SSA for F_{ij} can be obtained by the following expression (Lin & Reiter 1994):

$$F_{ij}(\vec{y},do(a,s)) \equiv \mathcal{R}^s[\beta_{ij}(\vec{y},do(a,s))]. \qquad (10)$$

Furthermore, we define $F_{ij}(\vec{y},S_0) \equiv \beta_{ij}(\vec{y},S_0)$. On the other hand, the fluent $G_{ij}(\vec{y},do(a,s))$ is such that its truth value is equivalent to that of $\beta_{ij}(\vec{y},s)$ (i.e., it "remembers" the truth value that β_{ij} had in the previous situation). The SSA for G_{ij} is simply $G_{ij}(\vec{y},do(a,s)) \equiv F_{ij}(\vec{y},s)$.

To model $prim_\delta$ in these planners we can obtain their *world-level* effects by looking into the SSA of every fluent (Pednault 1989). On the other hand, the *knowledge-level* effect is simply that $prim_\delta(\vec{y})$ senses the truth value of fluent $G_{ij}(\vec{y},do(a,s))$, for all i, conditioned on whether $\alpha_j(\vec{y},s)$ is true. The correctness of this approach is justified by the following result.

Proposition 4 *Let* Comp$[\mathcal{D},C]$ *be a theory of action that contains axiom* (9)*, and fluents* F_{ij} *and* G_{ij}*. Then* Comp$[\mathcal{D},C]$ *entails that* (9) *is equivalent to*

$$\begin{aligned}K(s',do(a,s)) \equiv&\\ &(\exists s'').s' = do(a,s'') \wedge K(s'',s) \wedge \varphi(s,s'')\wedge\\ &\quad\bigwedge_j\{a = prim_\delta(\vec{y}) \wedge \alpha_j(\vec{y},s) \supset\\ &\qquad\bigwedge_i G_{ij}(\vec{y},do(a,s)) \equiv G_{ij}(\vec{y},do(a,s''))\}.\end{aligned}$$

Proof: Follows from the correctness of regression. □

The immediate consequence of this result is that

$$\mathsf{Comp}[\mathcal{D},C] \models \alpha_j(\vec{y},s) \supset \bigwedge_i \mathbf{KWhether}(G_{ij}(\vec{y},do(prim_\delta,s))),$$

which intuitively expresses that $prim_\delta(\vec{y})$ is observing the truth value of $G_{ij}(\vec{y})$.

As we mentioned, the previous construction works with planners like GPT, where actions can have both *world effects* and *observations*. However, this still doesn't solve the problem completely for the planners like $\pi(\mathcal{P})$ and CFF, since (currently) they do not support actions with both world-level and knowledge-level effects. Nonetheless, this can be addressed by splitting $prim_\delta$ into two actions, say $Phys_\delta$ and Obs_δ. Action $Phys_\delta$ would have all the world-level effects of $prim_\delta$ and action Obs_δ would be a sensing action that observes F_{ij}. In this case, we also need to add special preconditions for action $Phys_\delta$, since we would need it to be performed always and only *immediately after* Obs_δ. Such an axiomatization is described by Baier & McIlraith (2005).

Extending PKS

To our knowledge, PKS (Petrick & Bacchus 2002) is the only planner in the literature that does not represent belief states explicitly. Moreover, it can represent domains using first-order logic and functions. Nevertheless, it does not allow the representation of knowledge about arbitrary formulae. In particular it cannot represent disjunctive knowledge.

PKS, does not directly support requirement 2 either. Moreover, its reasoning algorithm is not able to obtain reasonable results when adding the fluents F_{ij} and G_{ij}, due to its incompleteness.

PKS deals with knowledge of an agent using four databases. Among them, database K_w stores formulae whose truth values are known by the agent. In practice, this means that if an action senses property p, then p is added to K_w after performing it. While constructing a conditional plan, the K_w database is used to determine the properties on which it is possible to condition different branches of the plan. PKS's inference algorithm, **IA**, when invoked with ε can return value **T** (resp. **F**) if ε is known to be true (resp. false) by the agent. On the other hand, it returns **W** (resp. **U**) if the truth value of ε is known (resp. unknown) by the agent.

Nevertheless, since K_w can only store first-order conjunctions of literals, this means that in some cases, information regarding sensing actions of the type generated by our translation procedure would be lost. E.g., if $\neg f$ and g are known to the planner and an action that senses $f \vee g \wedge h$ is performed, PKS is not able to infer that it knows the truth value of h. For cases like this, this limitation can by overcome by the extension we propose below.

We propose to allow K_w to contain first-order CNF formulae. In fact, assume that K_w can contain a formula $\Gamma_1(\vec{x}) \wedge \Gamma_2(\vec{x}) \wedge \ldots \wedge \Gamma_k(\vec{x})$, where Γ_i is a first order clause, and free variables $\vec{x}$ are implicitly universally quantified. We now modify PKS's inference algorithm **IA** by replacing rule 7 of the algorithm of Bacchus & Petrick (1998) by the following rule (unfortunately, space precludes us from showing the whole algorithm). We assume the procedure is called with argument ε:

7. If there exists $\phi(\vec{x}) = \Gamma_1(\vec{x}) \wedge \ldots \wedge \Gamma_k(\vec{x}) \in K_w$ and a ground instance of ϕ, $\phi(\vec{x}/\vec{a})$ is such that (1) $\vec{a}$ are constants appearing in K_f, (2) There exists an $\alpha_m \in \Gamma_i$ such that $\alpha_m(\vec{x}/\vec{a}) = \varepsilon$, (3) For every Γ_j $(j \neq i)$ there exists a $\beta \in \Gamma_j$ such that $\mathbf{IA}(\beta(\vec{x}/\vec{a})) = \mathbf{T}$, and (4) For every $\alpha_\ell \in \Gamma_i$ $(\ell \neq m)$, $\mathbf{IA}(\alpha_\ell(\vec{x}/\vec{a})) = \mathbf{F}$. Then, **return(W)**.

Theorem 5 *The modified inference algorithm of PKS is sound.*

Proof sketch: The proof is based on the following facts (1) The modification only affects when the algorithm returns a **W** (2) The new rule's conclusions are based on the following valid formulae $\mathbf{KWhether}(\alpha \wedge \beta, s) \wedge \mathbf{Knows}(\alpha,s) \supset \mathbf{KWhether}(\beta,s)$, $\mathbf{KWhether}(\alpha \vee \beta,s) \wedge \mathbf{Knows}(\neg\beta,s) \supset \mathbf{KWhether}(\alpha,s)$, and $\mathbf{Knows}(\alpha,s) \vee \mathbf{Knows}(\beta,s) \supset \mathbf{Knows}(\alpha \vee \beta,s)$.

To actually use action $prim_\delta$ to plan with PKS, we need to divide it into two primitive actions, a world-altering action, say $Phys_\delta$, and a sensing action, say Obs_δ. Action Obs_δ has the effect of adding β_{ij}—in CNF—to the K_w database. On the other hand, $Phys_\delta$ contains all the world effects of $prim_\delta$. Again, through preconditions, we need to ensure that action $Phys_\delta$ is performed only and always immediately after

N	CFF	PKS	CFF +seek	PKS +seek
1	0.01	5.19	0.0	0.01
2	0.1	nomem	0.01	0.01
3	5.01	nomem	0.01	0.08
4	nomem	nomem	0.02	0.77
5	nomem	nomem	0.03	5.89

Table 1: Instances of the briefcase domain with sensing solved by PKS and CFF. "nomem" means the planner ran out of memory.

Obs_δ. This transformation is essentially the same that was proposed for belief-state-based planners that cannot handle actions with both physical and knowledge effects, and can be proved correct (Baier & McIlraith 2005).

This extension to PKS' inference algorithm is not yet implemented but is part of our future work. In the experiments that follow, we did not need to use this extension since the sensed formulae were simple enough.

6 Practical relevance

There were at least two underlying motivations to the work presented in this paper that speak to its practical relevance.

6.1 Web service composition

Web services are self-contained programs that are published on the Web. The airline ticket service at www.aircanada.com, or the weather service at www.weather.com are examples of Web services. Web services are compellingly modeled as programs comprising actions that effect change in the world (e.g., booking you a flight, etc.) as well as actions that sense (e.g., telling you flight schedules, the weather in a particular city, etc.). Interestingly, since Web services are self-contained, they are generally self-sufficient in the formal sense of this term, as described in this paper. As such, they fall into the class of programs that can be modeled as planning operators. This is just what is needed for WSC.

WSC is the task of composing existing Web services to realize some user objective. Planning your trip to the KR2006 conference over the Web is a great example of a WSC task. WSC is often conceived as a planning or restricted program synthesis task (McIlraith & Son 2002). Viewed as a planning task, one must plan with programs that sense to achieve WSC. While there has been significant work on WSC, few have addressed the issue of distinguishing between world-altering and sensing actions, fewer still have addressed the problem of how to represent and plan effectively with programs rather than primitive (one step) services. This work presents an important contribution towards addressing the WSC task.

6.2 Experiments

Beyond WSC, the second more general motivation for this work was to understand how to plan with macro-actions or programs, using operator-based planners. The advantages of using operator-based planners are many, including availability of planners and the ability to use fast heuristic search techniques. In general, the search space of plans of length k is exponential in k. When using macro-actions usually we can find shorter plans (composed by such macro-actions), therefore, the planner will effectively explore an exponentially smaller search space. When planning with sensing actions, plans are normally contingent, i.e. they have branches to handle different situations. The search space, therefore, is much bigger and any reduction in the length of the plan may exponentially reduce the time needed for planning.

```
$seek(o) = go(R_1); look(o);$ if $at(o,R_1)$ then $grasp(o); go(LR)$
        else $go(R_2); look(o);$ if $at(o,R_2)$ then $grasp(o); go(LR)$
        else $go(R_3); look(o);$ if $at(o,R_3)$ then $grasp(o); go(LR)$
        else $go(R_4); look(o);$ if $at(o,R_4)$ then $grasp(o); go(LR)$
        endIf endIf endIf endIf
```

Figure 1: Program *seek* is a tree program that makes the agent move through all the rooms looking for o and then bringing it to *LR*

To illustrate the computational advantages of planning with programs that sense, we performed experiments with a version of the *briefcase* domain (Pednault 1988), enhanced with sensing actions. In this domain, there are K rooms. The agent carries a briefcase for transporting objects. In any room r, the agent can perform an action $look(o)$ to determine whether o is in r. In the initial state, the agent is in room *LR*. There are N objects in rooms other than *LR*. The agent does not know the exact location of any of the objects. The goal is to be in *LR* with all the objects.

We performed experiments with PKS and CFF for $K = 4$ and $N = 1, \ldots, 5$. Each planner was required to find a plan with and without the use of macro-action $seek(o)$ (Figure 1). $seek(o)$ was compiled into a primitive action by our technique. We compared the running time of the planners using a 2 GHz linux machine with 512MB of main memory. PKS was run in iterative deepening mode. Table 1 shows running times for both planners with and without the *seek* action. These experiments illustrate the applicability of our approach in a domain that is challenging for state-of-the-art planners when only simple primitive actions are considered.

7 Summary and discussion

In this paper we addressed the problem of enabling operator-based planners to plan with *programs*. A particular challenge of this work was to ensure that the proposed method worked for programs that include sensing, though all the contributions are applicable to programs without sensing. We studied the problem in the situation calculus, using Golog to represent our programs. We did this to facilitate formal analysis of properties of our work. Nevertheless, because of the well-understood relationship between ADL and the situation calculus (Pednault 1989), the results apply very broadly to the class of planning operators represented in the popular plan domain description language PDDL (McDermott 1998).

Our contributions include a compilation algorithm for transforming programs into operators that are guaranteed to preserve program behaviour for the class of self-sufficient deterministic Golog tree programs. Intuitively, these are programs whose execution is guaranteed to be finite and whose outcome is determinate upon execution. We then showed how to plan with these new operators using existing operator-based planners that sense. In the case of PKS, we proposed a modification to the code to enable its use in the general case. For those interested in Golog, a side effect of this work was to define an offline transition semantics for executable Golog programs.

There were two underlying motivations to this work that speak to its practical relevance. The first was to address the problem of WSC. The class of programs that we can encapsulate as operators corresponds to most, if not all, Web services. As such, this work provides an important contribution to addressing WSC as a planning task. Our second motivation was the use of programs to represent macro-actions and how to use them effectively in operator-based planners. Again, our compilation algorithm provides a means of representing macro-actions as planning operators. Our experimental results, though in no way rigorous, illustrate the effectiveness of our technique.

Acknowledgments

We wish to thank Sebastian Sardina for his comments on an earlier version of this paper. We also thank the anonymous reviewers of NRAC-05 and KR2006 for their helpful comments. This research was funded by Natural Sciences and Engineering Research Council of Canada (NSERC) grant number 229717-04.

Appendix

Proof for Theorem 2

We first prove the following Lemma.

Lemma 2 *Let $\mathcal{D}$ be a theory of action such that $\mathcal{K}_{init}$ contains the reflexivity axiom. Let C be a set of deterministic Golog tree programs. Then, for all fluents F in the language of $\mathcal{D}$ that are not K, and for every $\delta \in C$ such that $\mathcal{D} \models ssf(\delta, s)$, theory* $\mathsf{Comp}[\mathcal{D}, C]$ *entails*

$$Do_K(\delta,s,s') \supset (F(\vec{x},s') \equiv F(\vec{x},do(prim_\delta,s))), \textit{ and}$$
$$Do_K(\delta,s,s') \supset (z = f(\vec{x},s') \equiv z = f(\vec{x},do(prim_\delta,s)))$$

Proof: In the interest of space, we show the proof for the first assertion. The proof of the second assertion is analogous.

Let $\mathcal{D}' = \mathsf{Comp}[\mathcal{D}, C]$. It suffices to prove that

1. $\mathcal{D}' \models Do_K(\delta,s,s') \wedge F(\vec{x},s') \supset F(\vec{x},do(prim_\delta,s)))$, and
2. $\mathcal{D}' \models Do_K(\delta,s,s') \wedge F(\vec{x},do(prim_\delta,s)) \supset F(\vec{x},s')$.

Proof for 1: Suppose $\mathcal{M}$ is a model of $\mathcal{D}'$ such that $\mathcal{M} \models (Do_K(\delta,s,S') \wedge F(\vec{x},S'))$, for some situation denoted by S'. From Proposition 2 and Lemma 1, we have that $\mathcal{M} \models Do^-(\delta,s,S') \wedge F(\vec{x},S')$. Since regression is correct and $\mathcal{M}$ also satisfies axiom (6), it follows immediately that $\mathcal{M} \models F(\vec{x},do(prim_\delta,s))$.
Proof for 2: Assume $\mathcal{M}$ is a model for $\mathcal{D}'$ such that $\mathcal{M} \models (Do_K(\delta,s,s') \wedge F(\vec{x},do(prim_\delta,s)))$, for any situations s, s'. By the successor state axiom of F, and correctness of regression, we conclude that $\mathcal{M} \models F(\vec{x},do(prim_\delta,s))$ iff

$$\begin{aligned}\mathcal{M} \models &(Do^-(\delta,s,S_1) \wedge F(\vec{x},S_1)) \vee \\ &\quad F(\vec{x},s) \wedge (\forall s_2)(Do^-(\delta,s,s_2) \supset F(\vec{x},s_2)),\end{aligned}$$

for some situation S_1. Since δ is deterministic and given that $\mathcal{M} \models Do_K(\delta,s,S')$, by Proposition 2 and Lemma 1, we have that $\mathcal{M} \models S_1 = s'$. The assertion above reduces to $\mathcal{M} \models F(\vec{x},s') \vee F(\vec{x},s) \wedge F(\vec{x},s')$, from which we conclude that $\mathcal{M} \models F(\vec{x},s')$. □

The proof of the theorem is now straightforward by using Lemma 2. □

Proof for Theorem 3

First we need the following result.

Lemma 3 *Let $\mathcal{D}$ be a theory of action such that $\mathcal{K}_{init}$ contains the reflexivity axiom,*

$$\begin{aligned}\mathcal{D} \cup \mathcal{T} \models K(s',s) \wedge K(\sigma',\sigma) \supset& \\ \{(\forall\delta).Do_K(\delta,s,\sigma) \supset& Do(\delta,s',\sigma')\}\end{aligned}$$

Proof: By induction on the structure of δ. □

Now, let $\mathcal{D}' = \mathsf{Comp}[\mathcal{D}, C]$. It suffices to prove the theorem for any arbitrary situation-suppressed fluent symbol F different from K. By expanding the definition of **Knows**, it suffices to prove

$$\begin{aligned}\mathcal{D}' \models (\forall\vec{x},s,s_1).Do_K(\delta,s,s_1) \supset& \\ \{(\exists s'')(K(s'',s_1) \wedge F(\vec{x})[s'']) \equiv& \\ (\exists s'')(K(s'',do(prim_\delta,s)) \wedge F(\vec{x})[s''])\},&\end{aligned}$$

($\Rightarrow$) We prove that

$$\begin{aligned}\mathcal{D}' \models (\forall\vec{x},s,s_1).Do_K(\delta,s,s_1) \supset& \\ \{(\exists s'')(K(s'',s_1) \wedge F(\vec{x})[s'']) \supset& \\ (\exists s'')(K(s'',do(prim_\delta,s)) \wedge F(\vec{x})[s''])\},&\end{aligned}$$

Suppose $\mathcal{M} \models \mathcal{D}'$ and that for some situation denoted by S'',

$$\mathcal{M} \models Do_K(\delta,s,s_1) \wedge K(S'',s_1) \wedge F(\vec{x})[S''],$$

Notice that $\mathcal{M} \models s \sqsubseteq s_1$, and since $\mathcal{M} \models K(S'',s_1)$, there exists situation denoted by S''' such that $\mathcal{M} \models S''' \sqsubseteq S''$ and such that

$$\mathcal{M} \models Do_K(\delta,s,s_1) \wedge K(S''',s) \wedge K(S'',s_1) \wedge F(\vec{x})[S'']. \quad (11)$$

From Lemma 3, and Proposition 2, we have that

$$\mathcal{M} \models Do^-(\delta,S''',S'') \wedge F(\vec{x})[S'']. \quad (12)$$

From (12) and (6) it follows immediately that $\mathcal{M} \models F(\vec{x})[do(prim_\delta,S''')]$. Now notice that $\mathcal{M} \models Do^-(\delta,s,s_1)$ (from Lem. 1 and Prop. 2), which together with (11) and (12) and the fact that $\mathcal{M}$ satisfies (8), implies $\mathcal{M} \models K(do(prim_\delta,S'''))$. This finishes the proof for $\Rightarrow$.
($\Leftarrow$) Suppose that for some situation S'',

$$\mathcal{M} \models Do_K(\delta,s,s_1) \wedge K(S'',do(prim_\delta,s)) \wedge F(\vec{x})[S'']$$

From the successor state axiom of K, for some S''',

$$\mathcal{M} \models Do_K(\delta, s, s_1) \wedge K(do(prim_\delta, S'''), do(prim_\delta, s)) \wedge \\ K(S''', s) \wedge F(\vec{x})[do(prim_\delta, S''')].$$

Since $\mathcal{M}$ satisfies (8),

$$\mathcal{M} \models Do_K(\delta, s, s_1) \wedge K(S_2, S_1) \wedge Do^-(\delta, S''', S_2) \wedge \\ Do^-(\delta, s, S_1) \wedge K(S''', s) \wedge F(\vec{x})[do(prim_\delta, S''')].$$

From Lemma 1 and the fact that δ is deterministic, $\mathcal{M} \models s_1 = S_1$, and therefore,

$$\mathcal{M} \models K(S_2, s_1) \wedge Do^-(\delta, S''', S_2) \wedge \\ K(S''', s) \wedge F(\vec{x})[do(prim_\delta, S''')].$$

Now, given that $\mathcal{M}$ satisfies (7),

$$\mathcal{M} \models K(S_2, s_1) \wedge Do^-(\delta, S''', S_2) \wedge Do^-(\delta, S''', S_3) \wedge F(\vec{x})[S_3].$$

Once again, since δ is deterministic, $\mathcal{M} \models S_2 = S_3$ and therefore $\mathcal{M} \models K(S_2, s_1) \wedge F(\vec{x})[S_2]$, which proves $\Leftarrow$. □

References

Bacchus, F., and Petrick, R. 1998. Modeling an agent's incomplete knowledge during planning and execution. In Cohn, A. G.; Schubert, L.; and Shapiro, S. C., eds., *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*, 432–443. San Francisco, CA: Morgan Kaufmann Publishers.

Baier, J., and McIlraith, S. 2005. Planning with programs that sense. In *Proceedings of the Sixth Workshop on Nonmonotonic Reasoning, Action, and Change (NRAC-05)*, 7–14.

Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2001. Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI-01*, 473–478.

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS)*, 52–61.

Davis, E. 1994. Knowledge preconditions for plans. *Journal of Logic and Computation* 4(5):721–766.

de Giacomo, G., and Levesque, H. 1999. An incremental interpreter for high-level programs with sensing. In Levesque, H., and Pirri, F., eds., *Logical foundation for cognitive agents: contributions in honor of Ray Reiter*. Berlin: Springer Verlag. 86–102.

de Giacomo, G.; Lespérance, Y.; and Levesque, H. 2000. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121(1–2):109–169.

Erol, K.; Hendler, J.; and Nau, D. 1994. HTN planning: Complexity and expressivity. In *Proc. of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, 1123–1128.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3:251–288.

Hoffmann, J., and Brafman, R. 2005. Contingent planning via heuristic forward search with implicit belief states. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, 71–80. Monterey, CA, USA: Morgan Kaufmann.

Korf, R. E. 1987. Planning as search: A quantitative approach. *Artificial Intelligence* 33(1):65–88.

Lespérance, Y.; Levesque, H.; Lin, F.; and Scherl, R. 2000. Ability and knowing how in the situation calculus. *Studia Logica* 66(1):165–186.

Levesque, H. 1996. What is planning in the presence of sensing? In *The Proceedings of the Thirteenth National Conference on Artificial Intelligence, AAAI-96*, 1139–1146. Portland, Oregon: American Association for Artificial Intelligence.

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

McCarthy, J., and Hayes, P. J. 1969. Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., and Michie, D., eds., *Machine Intelligence 4*. Edinburgh University Press. 463–502.

McDermott, D. V. 1998. PDDL — The Planning Domain Definition Language. Technical Report TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

McIlraith, S. A., and Fadel, R. 2002. Planning with complex actions. In *9th International Workshop on Non-Monotonic Reasoning (NMR)*, 356–364.

McIlraith, S., and Son, T. C. 2002. Adapting Golog for composition of semantic web services. In *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002)*, 482–493.

Moore, R. C. 1985. A formal Theory of Knowledge and Action. In Hobbs, J. B., and Moore, R. C., eds., *Formal Theories of the Commonsense World*. Norwood, New Jersey: Ablex Publishing Corp. chapter 9, 319–358.

Pednault, E. 1988. Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence* 4(4):356–372.

Pednault, E. P. D. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the First International Conference of Knowledge Representation and Reasoning*, 324–332.

Petrick, R. P. A., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS)*, 212–222.

Reiter, R. 2001. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.

Sacerdoti, E. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* 5:115–135.

Sardina, S.; de Giacomo, G.; Lespérance, Y.; and Levesque, H. 2004. On the semantics of deliberation in IndiGolog – from theory to implementation. *Annals of Mathematics and Artificial Intelligence* 41(2–4):259–299. Previous version appeared in Proc. of KR-2002.

Scherl, R., and Levesque, H. J. 2003. Knowledge, action, and the frame problem. *Artificial Intelligence* 144(1–2):1–39.

Son, T. C.; Tu, P. H.; and Baral, C. 2004. Planning with sensing actions and incomplete information using logic programming. In *Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, volume 2923 of *Lecture Notes in Computer Science*, 261–274. Fort Lauderdale, FL, USA: Springer.

Weld, D. S.; Anderson, C. R.; and Smith, D. E. 1998. Extending graphplan to handle uncertainty & sensing actions. In *Proceedings of AAAI-98*, 897–904.