Planning with programs that sense

Jorge Baier and **Sheila McIlraith**
Department of Computer Science
University of Toronto
{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 traditional operator-based planning techniques to plan with programs that sense for a restricted but compelling class of problems. We conclude by discussing the applicability of these results to existing operator-based planners that allow 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. In most instances the planners are propositional and the generated plans are conditional. 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 wish to consider may both sense and act in the world. 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* (e.g., [12]). 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 and/or act in the world – e.g., determining flight costs or credit card approval, arranging for the delivery of goods and the debiting of accounts, etc. As such, the task of automated Web service composition (WSC) can be conceived as the task of planning with programs, or as a specialized version of a program synthesis task. While space precludes detailed discussion of the WSC task, this paper addresses some key remaining challenges to achieving it.

A secondary motivation for this work is to improve the efficiency of planning with sensing by representing useful (conditional) plan segments as programs. Though we do not study its effectiveness in this paper, planning with some form of macro-actions (e.g., [18; 8; 6; 13; 5]) can dramatically improve the efficiency of plan generation by reducing the search space size and the length of a plan. This is of particular importance in planning problems that involve sensing actions.

Levesque argued in [10] 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 we assume no uncertainty in the outcome of actions, and no exogenous actions, 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 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. We conclude by briefly discussing the applicability of these results to existing operator-based planners that allows sensing.

2 Preliminaries

In the two subsections that follow we briefly review the situation calculus [17], including a treatment of sensing actions and knowledge [20]. We also review the transition seman-

tics 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 2.1 and the perhaps less familiar distinction of deterministic tree programs found in Section 2.2.

2.1 The situation calculus

The situation calculus [11; 17] 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. Following Scherl and Levesque [20], 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 [14]. $K(s',s)$ 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] $\mathbf{Knows}(\phi,s) \stackrel{\text{def}}{=} (\forall s').K(s',s) \supset \phi[s']$, and $\mathbf{KWhether}(\phi,s) \stackrel{\text{def}}{=} \mathbf{Knows}(\phi,s) \vee \mathbf{Knows}(\neg\phi,s)$. 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 in s is also true in s.

A situation calculus theory of action, $\mathcal{D}$ logically describes the dynamics of a domain. Following the axiomatization of [17], the theories of action we consider comprise at least the following:

- Σ, a set of foundational axioms.
- $\mathcal{D}_{ss}$, a set of successor state axioms (SSAs). The set of SSAs can be compiled from a set of *effect axioms*, $\mathcal{D}_{eff}$ [16]. An effect axiom describes the effect of an action on the truth value of certain fluents, e.g., $a = startCar \supset engineStarted(do(a,s))$.
- $\mathcal{D}_{ap}$, a set of action precondition axioms, one for each action. They are usually compiled from a set $\mathcal{D}_{nec}$ of *necessary conditions* on the fluent *Poss*, e.g. $Poss(startCar,s) \supset batteryOK(s)$.
- $\mathcal{D}_{una}$, the set of unique names axioms for actions.
- $\mathcal{D}_{S_0}$, a set that describes the initial state of the world.
- $\mathcal{K}_{init}$, a set that defines the properties of the K fluent in the initial situations and preserved in all situations.
- D_{golog} a set of axioms for Golog's semantics.

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

[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.

Agents can gather information from the world using sensing actions. A sensing action results in the agent knowing whether a particular property of the world is true, or knowing the value of a particular term. In [20] sensing actions do not alter the state of the world. They only alter the agent's state of knowledge. [20] introduces 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, the SSA for K is:

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

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. Since a_i is not world-altering this means that in all situations reachable from $do(a_i,s)$ either ψ_i or $\neg\psi_i$ holds, i.e. the agent knows whether ψ_i holds.

In contrast to [20], 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. To model an agent with sensing actions $a_1,\ldots,a_n$ such that each action a_i formula ψ_i, the axiomatizer must generate the following sufficient condition axioms for each a_i,

$$K(s'',s) \wedge a = a_i \wedge (\psi_i(s) \equiv \psi_i(s'')) \supset K(do(a,s''),do(a,s)), \quad (2)$$

which intuitively express the same dynamics of the K-reachability for situations as (1) 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:

$$s' = do(a,s'') \wedge K(s'',s) \wedge \bigwedge_{i=1}^{n} a \neq a_i \supset K(s',do(a,s)), \quad (3)$$

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

Proposition 2.1 *Predicate completion on axioms of the form* (2) *and* (3) *is equivalent to the SSA for K defined in* (1)*.*

2.2 Golog's syntax and semantics

Golog is a high-level agent programming language whose semantics is based on the situation calculus [17]. A Golog program is a complex action[3] potentially composed from:

nil – the empty program	a – primitive action
ϕ? – test action	$\pi x.\,\delta$ – nondet. choice of argument
$\delta_1;\delta_2$ – sequences	$\delta_1 \vert \delta_2$ – nondet. choice of action
while ϕ **do** δ **endW** – loop	**if** ϕ **then** δ_1 **else** δ_2 **endif** – conditional

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

In Section 4.1 we will propose a compilation algorithm for Golog programs that are *deterministic tree* programs. A Golog tree program is one that does not contain loops. A Golog program is *deterministic* if it does not contain non-deterministic constructs. 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). Thus, following [13], we extend the Golog language with the bounded loop construct, **while**$_k$ ϕ **do** δ **endW** defined equal to **if** ϕ **then** $\{\delta;$ **while**$_{k-1}$ ϕ **do** δ **endW**$\}$ **else** *nil* **endif**, for $k > 0$ and equal to *nil* if $k = 0$. We include this as an admissible construct for a tree program.

Golog has both an evaluation semantics [17] and a transition semantics [7]. 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. Using the transitive closure of *Trans*, $Trans^*$, the predicate $Do(\delta, s, s')$ such that it is true iff program δ terminates in situation s' if executed in situation s. Some axioms for *Trans* and *Final* are shown below.

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

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

3 A semantics for executable Golog programs

As Levesque [10] 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. Even in cases where we assume no uncertainty in the outcome of actions, and no exogenous actions, this remains challenging because of incomplete information about the initial state. When planning with programs, as we are proposing here, the problem only gets worse. In particular, Golog's existing semantics does not consider sensing actions and furthermore does not consider whether the agent has the ability to execute a given program. As a first step towards planning with programs that sense, we define a semantics for Golog that ensures that any Golog program with a terminating situation will also be executable by an agent. This semantics provides the foundation for results in subsequent sections.

For example, consider an action theory $\mathcal{D}$ such that $\mathcal{D} \not\models \phi[S_0]$ and $\mathcal{D} \not\models \neg\phi[S_0]$, and let $\delta \stackrel{\text{def}}{=}$ **if** ϕ **then** a **else** b **endif**. Assume furthermore that $\mathcal{D} \models Poss(a, S_0)$ and $\mathcal{D} \models Poss(b, S_0)$. Then, it holds that $\mathcal{D} \models (\exists s) Do(\delta, S_0, s)$, i.e. δ is executable in S_0 (in fact, $\mathcal{D} \models Do(\delta, S_0, s) \equiv s = do(a, S_0) \vee s = do(b, S_0)$). This fact is certainly counter-intuitive since in S_0, the agent does not have enough information to determine whether ϕ holds.

Intuitively, 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.

To our knowledge, no such semantics exists. Nevertheless, there is related work. In [4], the semantics of programs with sensing is defined 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. Several papers have addressed the problem of knowing how to execute a plan [3] or more specifically, a Golog program. In [9], a predicate *CanExec* is defined to establish when a program can be executed by an agent. In [19], epistemically feasible programs are defined using the online semantics of [4]. Finally, a simple definition is given in [12], which defines a self-sufficient property, such that $ssf(\delta, s)$ is true iff an agent knows how to execute program δ in situation s. Its definition is given below.

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

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

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

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

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

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

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

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

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 $Final_K$. We conjecture that our proposed semantics is equivalent to that of [4] in an online setting. (We plan to prove this in future work.) The definitions of $Trans_K$ and $Final_K$ follow.

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

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

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

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

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

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

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

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

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

Analogous to the definition of Do, we define $Do_K(\delta,s,s') \stackrel{\text{def}}{=} (\exists\delta')\,Trans^*_K(\delta,s,\delta',s') \wedge Final_K(\delta',s')$. Observe that in contrast to *Trans*, $Trans_K$ of *if-then-else* explicitly requires the agent to know the value of the condition. Consequently, if $\mathcal{D} \not\models \mathbf{KWhether}(\phi,S_0)$, then $\mathcal{D} \models \neg(\exists s)\,Do_K(\Delta,S_0,s)$. However, if $sense_\phi$ senses ϕ, then $\mathcal{D} \models (\exists s)\,Do_K(sense_\phi;\Delta,S_0,s)$.

A natural question is when this semantics equivalent to the original semantics. We can prove that both semantics are equivalent for self-sufficient programs (in the sense of [12]).

Lemma 3.1 *Let $\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{K}_{init} \cup \mathcal{D}_{golog} \cup \mathcal{D}_{ssf}$, where $\mathcal{D}_{ssf}$ is the set of axioms defining the ssf fluent. Then if $\mathcal{K}_{init}$ contains the reflexivity axiom for K,*

$$\mathcal{D} \models (\forall\delta,s).\,ssf(\delta,s) \supset \{(\forall s').\,Do(\delta,s,s') \equiv Do_K(\delta,s,s')\}$$

The preceding lemma is fundamental for the rest of the 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 [12; 19], such as tree programs in which each *if-then-else* that conditions on ϕ is preceded by a $sense_\phi$.

4 Planning with programs that sense

Motivated by the problem of Web service composition, and by the desire to use macro-actions in conventional planning settings, our main concern in this paper is with planning with programs that sense. As such, we extend the notion of planning with primitive actions to planning with programs that sense as the fundamental building blocks of a plan. One of our interests is to enable the use of pre-existing programs as macro-actions in a classical planning setting. As pointed out by [10], a plan in the presence of sensing is a program that may contain conditional and loop constructs. In our framework we define a plan in the presence of sensing as a Golog program.

Definition 1 (A plan) *Given a theory of action $\mathcal{D}$, and a goal G we say that Golog program δ is a plan for G in situation s relative to theory $\mathcal{D}$ iff $\mathcal{D} \models (\forall s').\,Do_K(\delta,s,s') \supset G(s')$.*

In classical planning, a planning algorithm constructs plan δ by choosing actions from a set A of primitive actions. Rather, in planning with programs that sense, the planner has an additional set C of programs, which may contain sensing actions, that it can use to construct plans.

Example Consider an agent working on an assembly line constructing several types of widgets. The agent is able to achieve high-level goals including building complex objects using the widgets of the assembly line. In order to achieve her goals the agent must do planning. The agent can perform a variety of primitive actions and also some built-in, high-level programs. For example, the following program picks up blocks from the assembly chain, possibly repairs them, and then delivers them to a production zone.

$$\delta = pick(b); checkDamaged(b);$$
$$\textbf{if } damaged(b) \textbf{ then } repair(b); register(b) \textbf{ else } nil \textbf{ endif}; deliver(b)$$

The action $pick(b)$ picks a block b from the assembly line, action $checkDamaged(b)$ is a sense action that senses whether or not b is damaged, action $deliver(b)$ delivers b to a production zone, and action $register(b)$ logs b in the "damaged" database.

In the interest of space, we do not show all the axioms in the theory; rather, we show some axioms that compose $\mathcal{D}_{nec}$ and $\mathcal{D}_{eff}$.

$$Poss(pick(b),s) \supset inChain(b),$$
$$Poss(repair(b),s) \supset damaged(b),$$
$$a = repair(b) \supset \neg damaged(b, do(a,s)),$$
$$a = register(b) \supset logged(b, do(a,s)).$$

The successor state axioms for the fluents $logged$, $damaged$ (generated from $\mathcal{D}_{eff}$), and for K (generated from $\mathcal{K}_s$) are as follows.

$$logged(b, do(a,s)) \equiv a = register(b) \vee logged(b,s),$$
$$damaged(b, do(a,s)) \equiv paintFresh(b,s) \wedge a = pick(b) \vee damaged(b,s) \wedge a \neq repair(b)$$
$$K(s', do(a,s)) \equiv (\exists s'').\, s'' = do(a,s') \wedge K(s'',s) \wedge \{a = checkDamaged(b) \supset (damaged(s'') \equiv damaged(s))\}$$

Suppose we want an operator-based planner (e.g., STRIPS, Graphplan, SATplan, etc.) to use the complex action δ. Instead of a program, we would need to have an operator-based action representation (i.e. we need to represent δ as a primitive action). This representation would not only describe the physical effects of the action (e.g., after we perform $\delta(B)$, block B is in the production zone and not damaged), but also at a knowledge level (if we know that B is not damaged, after we perform B we know whether or not B's paint is fresh!).

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 $\mathsf{Comp}[\mathcal{D},C]$ appear modeled by new primitive actions. In so doing, we are able to use traditional operator-based planners to plan with macro-actions, and to perform WSC with so-called composite services.

4.1 Theory compilation

A program with sensing may produce both effects in the world and in the knowledge of the agent. Therefore, if we want to replace a program by one primitive action, this action should have both knowledge and physical effects. In the standard situation calculus, though, it is normally assumed that actions either affect the world or the knowledge of the agent but not both. Therefore, we will compile each program into one sensing action and one physical action.

We now describe how we can generate a new theory of action that contains a new sensing action Obs_δ and a new

physical action $Phys_\delta$ for each program δ. Then we prove that those actions, when executed one immediately after the other, capture all physical and knowledge-level effects of the original program δ.

We start with a theory of action $\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \cup \mathcal{K}_{init} \cup \mathcal{D}_{golog} \cup \mathcal{D}_{ssf}$, about a set $\mathcal{A}$ of primitive actions, and we generate a new theory $\mathsf{Comp}[\mathcal{D}, C]$ that contains a new set for 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 $\mathcal{K}_s$, and that the set of precondition axioms, $\mathcal{D}_{ap}$, has been compiled from a set of necessary precondition conditions, $\mathcal{D}_{nec}$. 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 $\mathcal{D} \models (\forall s).ssf(\delta, s)$. Finally, assume that the fluent symbol *Enabled* is not part of the language of $\mathcal{D}$. We generate the new theory in the following way.

1. Make $\mathcal{D}'_{eff} := \mathcal{D}_{eff}$, $\mathcal{D}'_{nec} := \mathcal{D}_{nec}$, and $\mathcal{K}'_s := \mathcal{K}_s$, and $\mathcal{D}_{una} := \mathcal{D}'_{una}$.
2. We need that actions Obs_δ and $Phys_\delta$ are used either in sequence by the planner or not used at all. We use the predicate *Enabled* to enforce this. The following axioms are added to $\mathcal{D}'_{eff}$ for each $\delta \in C$:

$$a = Obs_\delta(\vec{y}) \supset Enabled(Phys_\delta(\vec{y}), do(a,s)),$$
$$a = Phys_\delta(\vec{y}) \supset \neg Enabled(Phys_\delta(\vec{y}), do(a,s)),$$

   i.e., $Phys_\delta$ becomes *Enabled* immediately after Obs_δ, is executed.
3. For each $\delta \in C$, we add the following necessary precondition axioms to $\mathcal{D}'_{ap}$,

$$Poss(Obs_\delta(\vec{y}), s) \supset \mathcal{R}^s[(\exists s')\, Do(\delta, s, s')],$$
$$Poss(Phys_\delta(\vec{y}), s) \supset Enabled(Phys_\delta(\vec{y}), s),$$

   therefore, $Obs_\delta(\vec{y})$ can be executed iff program δ could be executed in s, and $Phys_\delta(\vec{y})$ iff it is *Enabled*. The operator $\mathcal{R}^s$ is Reiter's regression operator over φ but relativized to situation s, i.e. $\mathcal{R}^s[\varphi]$ is a formula uniform in s [4] and equivalent to φ.
4. For each $\alpha \in \mathcal{A}$, and every $\delta \in C$, we add the following necessary precondition axioms to $\mathcal{D}'_{ap}$:

$$Poss(\alpha(\vec{x}), s) \supset \neg Enabled(Phys_\delta(\vec{y}), s).$$

   We add this because we do not want to allow an arbitrary primitive action after the execution of Obs_δ. Furthermore, for each $\delta, \delta' \in C$ such that $\delta \neq \delta'$,

$$Poss(Obs_\delta(\vec{z}), s) \supset \neg Enabled(Phys_{\delta'}(\vec{y}), s),$$
$$Poss(Phys_\delta(\vec{z}), s) \supset \neg Enabled(Phys_{\delta'}(\vec{y}), s).$$

5. For each fluent $F(\vec{x}, s)$ in the language of $\mathcal{D}$ that is not the K fluent, and each complex action $\delta \in C$ we add the following effect axioms to $\mathcal{D}'_{eff}$:

$$a = Phys_\delta(\vec{y}) \wedge \mathcal{R}^s[(\exists s')\,(Do(\delta, s, s') \wedge F(\vec{x}, s'))] \supset F(\vec{x}, do(a, s)), \quad (4)$$

$$a = Phys_\delta(\vec{y}) \wedge \mathcal{R}^s[(\exists s')\,(Do(\delta, s, s') \wedge \neg F(\vec{x}, s'))] \supset \neg F(\vec{x}, do(a, s)). \quad (5)$$

   i.e. F is true (resp. false) after executing $Phys_\delta$ in s if after executing δ in s it is true (resp. false).
6. For each functional fluent $f(\vec{x}, s)$ in the language of $\mathcal{D}$, and each complex action $\delta \in C$ we add the following effect axiom to $\mathcal{D}'_{eff}$:

$$a = Phys_\delta(\vec{y}) \wedge \mathcal{R}^s[(\exists s')\,(Do(\delta, s, s') \wedge z = f(\vec{x}, s'))] \supset z = f(\vec{x}, do(a, s)),$$

7. For each $\delta \in C$, we add the following sufficient condition axiom to $\mathcal{K}'_s$:

$$a = Obs_\delta(\vec{y}) \wedge s'' = do(a, s') \wedge \mathcal{R}^s[(\exists s_1, s_2)\,(Do(\delta, s, s_1) \wedge Do(\delta, s', s_2) \wedge K(s_2, s_1))] \supset K(s'', do(a, s)).$$

8. For each $\delta, \delta' \in C$ such that $\delta \neq \delta'$ and $\alpha \in \mathcal{A}$, add the following to $\mathcal{D}'_{una}$.

$$\alpha(\vec{x}) \neq Obs_\delta(\vec{y}), \alpha(\vec{x}) \neq Phys_\delta(\vec{y}), Obs_\delta(\vec{y}) \neq Phys_\delta(\vec{y}').$$

9. Compile a new set of SSAs $\mathcal{D}'_{ss}$ from $\mathcal{D}'_{eff}$, and a new set of precondition axioms $\mathcal{D}'_{ap}$ from $\mathcal{D}'_{nec}$. The new theory, is defined as follows.

$$\mathsf{Comp}[\mathcal{D}, C] = \Sigma \cup \mathcal{D}'_{ss} \cup \mathcal{D}'_{ap} \cup \mathcal{D}'_{una} \cup \mathcal{D}_{S_0} \cup \mathcal{K}_{init} \cup \mathcal{D}_{golog} \cup \mathcal{D}_{ssf}.$$

Theorem 4.1 *If $\mathcal{D}$ is consistent and C contains only deterministic tree programs then $\mathsf{Comp}[\mathcal{D}, C]$ is consistent.*

Indeed, if C contains one non-deterministic action, we cannot guarantee that $\mathsf{Comp}[\mathcal{D}, C]$ is consistent. Furthermore, we can prove that $Phys_\delta$ emulates δ.

Lemma 4.1 *Let $\mathcal{D}$ be a theory of action where K is reflexive, and 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*

$(\forall s, s', \vec{x}).Do_K(\delta, s, s') \supset (F(\vec{x}, s') \equiv F(\vec{x}, do(Phys_\delta, s)))$, *and*

$(\forall s, s', \vec{x}, z).Do_K(\delta, s, s') \supset (z = f(\vec{x}, s') \equiv z = f(\vec{x}, do(Phys_\delta, s)))$

Now we establish a complete correspondence at the physical level between our original programs and the compiled primitive actions after performing $[Obs_\delta, Phys_\delta]$.

Theorem 4.2 *Under the same assumptions as Lemma 4.1, let $\phi(\vec{x})$ be an arbitrary situation-suppressed, objective formula that does not mention the K fluent. Then,*

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

[4] A formula is uniform in s iff all terms of sort situation it mentions are s.

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

Theorem 4.3 *Let $\mathcal{D}$ be a the theory of action where K is reflexive, and C be a set of deterministic Golog tree programs, and $\phi(\vec{x})$ be a situation-suppressed, objective formula. If $\delta \in C$ and $\mathcal{D} \models (\forall s).ssf(\delta, s)$, then if $\mathcal{D}$ contains the reflexivity axiom for K,*

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

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 4.4 *Let D be a theory of action, C be a set of deterministic Golog tree programs, and $G(s)$ be a formula of the situation calculus. Then, if Δ is a plan for $G(s) \wedge \bigwedge_{\delta \in C} \neg(\exists \vec{y}) Enabled(Phys_\delta(\vec{y}), s)$ 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 $[Obs_\delta; Phys_\delta]$ in Δ by δ. Then we prove that Δ' also achieves the goal, from theorems 4.2 and 4.3. □

It is worth noting that the preceding proof would not have worked if plans (Definition 1) had been defined as Golog programs with a concurrent construct (such as that of Congolog). Such a construct, say $\delta_1 || \delta_2$, would specify that complex actions δ_1 and δ_2 can be executed concurrently, i.e. any interleaved execution of δ_1 and δ_2 would reach the goal. Handling this case is important since some actual planners (even in the classical setting) are able to generate plans that are *nonlinear*, i.e., that contain partially ordered sequences of actions (which in practice means concurrent execution).

Imagine that the planner has returned plan $\{Obs_\delta; Phys_\delta\} || A$ for goal G. Given the preconditions in the theory, this means that executing either $A; Obs_\delta; Phys_\delta$ or $Obs_\delta; Phys_\delta; A$ would achieve the goal in $\mathsf{Comp}[\mathcal{D}, C]$. Unfortunately, this does not necessarily mean that $\delta || A$ will achieve the goal in $\mathcal{D}$, since δ is a complex action. Allowing the execution of A during the execution of δ may invalid some precondition of an action in δ or change the truth value of a fluent in a way that is not predicted by the theory compilation. A simple—though not totally satisfactory—fix for this is that whenever the planner returns a plan of the form $\{Obs_\delta; Phys_\delta\} || \Delta$ in theory $\mathsf{Comp}[D, C]$, then a plan in $\mathcal{D}$ is either $\delta; \Delta$ or $\Delta; \delta$. We think that less restrictive solutions would imply tweaking the preconditions of the actions in δ. This issue is part of our current and future research.

Example (cont.) We now show the result of applying theory compilation to the action theory of our example. For the fluent *damaged*, axioms of the form:

$$a = Phys_\delta(b) \wedge \mathcal{R}^s[(\exists s')(Do(\delta, s, s') \wedge (\neg) damaged(\vec{x}, s'))] \supset (\neg) damaged(\vec{x}, do(a, s)),$$

simplify into

$$a = Phys_\delta(b) \wedge inChain(b, s) \supset \neg damaged(b, do(a, s)).$$

Following the same procedure, the SSA generated for *logged* is the following:

$$logged(b, do(a, s)) \equiv a = register(b) \vee a = Phys_\delta(b) \wedge (paintFresh(b, s) \vee damaged(b, s)) \vee logged(b, s)$$

and the following is the SSA for K,

$$K(s', do(a, s)) \equiv (\exists s'').s' = do(a, s'') \wedge K(s', s) \wedge$$
$$(a = checkDamaged(b) \supset damaged(s'') \equiv damaged(s)) \wedge$$
$$(a = Obs_\delta(b) \supset \{(damaged(s'') \vee paintFresh(b, s'')) \equiv (damaged(s) \vee paintFresh(b, s))\}).$$

A last thing worth pointing out is that our theory compilation can only be used for complex actions that can be proved self-sufficient for all situations. We could have done this differently. As said before, we could 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 something like $Poss(Obs_\delta(\vec{y}), s) \supset \mathcal{R}^s[(\exists s')Do(\delta, s, s') \wedge ssf(\delta, s)]$ in step 3 of theory compilation. This modification keeps the validity of our theorems but the resulting expression in the precondition may usually contain complex formulae referring to the knowledge of the agent we view as a problematic in practical applications.

5 From theory to practice

We have shown that under certain circumstances, planning with programs can theoretically be 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) [22], the MDP-based planner GPT [2], the model-checking-based planner MBP[5] [1], the logic-programming-based planner $\pi(\mathcal{P})$ [21], and the knowledge-level planner PKS [15].

All of these planners but PKS keep an implicit or explicit representation of 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. In our view, the limited expressiveness of these planners is extremely restrictive, especially because they are unable to represent functions, which is of a great importance in many practical applications including WSC. To our knowledge, PKS [15] 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 abou

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

arbitrary formulae. In particular it cannot represent disjunctive knowledge.

All of these planners are able to represent conditional effects of physical actions, therefore, the representation of action $Phys_\delta$ is straightforward. Unfortunately, the representation of the effects of Obs_δ is not trivial in some cases. Examining the general structure of the SSA for K after theory compilation we observe that to represent the effects of Obs_δ we need two characteristics from a planner.

- The planner must be able to represent *conditional* sensing actions. Among the planners investigated, SGP is the only one that cannot be adapted to this requirement. 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 Obs_δ into several actions with different preconditions.
- The planner must be able to represent that Obs_δ reports the truth value of, in general, arbitrary formulae. Some planners (SGP, MBP) can represent arbitrary (propositional) observation formulae but others, e.g., (GPT, $\pi(\mathcal{P})$, PKS) cannot. This is a somewhat serious limitation for planners that cannot represent such knowledge effects. In our example, this would prevent those planners from realizing that if it is known that the block b is not damaged, then after executing $[Obs_\delta(b); Phys_\delta(b)]$, it knows whether $paintFresh(b)$.

  To overcome the limitations of such planners, we have designed an algorithm (which we omit for lack of space) that takes the final SSA for K and is able to generate some useful knowledge effect axioms. In our example, the algorithm would generate the effect axiom $\mathbf{Knows}(\neg damaged(b), s) \Rightarrow \mathbf{KWhether}(paintFresh(b), do(Obs_\delta, s))$. Unfortunately, the algorithm is incomplete in the sense that the rules generated cannot capture all knowledge effects stemming from sensing arbitrary formulae.

6 Summary and discussion

In this paper we examined 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. We studied this problem in the language of the situation calculus, appealing to Golog to represent our programs. To this end, we proposed an offline execution semantics for Golog programs with sensing, proving its equivalence to previous online execution semantics, under certain conditions. We then proposed a compilation method that transforms our action theory with programs into a new theory where programs are replaced by primitive actions. This enabled us, in theory, to use traditional operator-based planning techniques to plan with programs that sense for a restricted but compelling class of problems. We concluded by discussing the applicability of these results to existing operator-based planners that allow sensing. This work makes an important contribution to the general problem of Web service composition by enabling the composition of so-called composite services using traditional operator-based planners that include sensing. The work also provides a mechanism for including macro-actions in operator-based planners that include sensing. We continue to explore these topics in future work.

References

[1] Piergiorgio Bertoli, Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI-01*, pages 473–478, Seattle, WA, USA, 2001.

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

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

[4] Giuseppe de Giacomo and Hector Levesque. An incremental interpreter for high-level programs with sensing. In Hector Levesque and Fiora Pirri, editors, *Logical foundation for cognitive agents: contributions in honor of Ray Reiter*, pages 86–102. Springer Verlag, Berlin, 1999.

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

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

[7] Giuseppe De Giacomo, Yves Lespérance, and Hector Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1–2):109–169, 2000.

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

[9] Yves Lespérance, Hector Levesque, Fangzhen Lin, and Richard Scherl. Ability and knowing how in the situation calculus. *Studia Logica*, 66(1):165–186, October 2000.

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

[11] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[12] Sheila McIlraith and Tran Cao Son. Adapting Golog for composition of semantic web services. In *Proceedings*

of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002), pages 482–493, Toulouse, France, April 2002.

[13] Sheila A. McIlraith and Ronald Fadel. Planning with complex actions. In *9th International Workshop on Non-Monotonic Reasoning (NMR)*, pages 356–364, Toulouse, France, 2002.

[14] Robert C Moore. A formal Theory of Knowledge and Action. In Jerry B. Hobbs and Robert C. Moore, editors, *Formal Theories of the Commonsense World*, chapter 9, pages 319–358. Ablex Publishing Corp., Norwood, New Jersey, 1985.

[15] Ronald P. A. Petrick and Fahiem Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 212–222, Toulouse, France, 2002.

[16] Raymond Reiter. *The Frame Problem in the Situation Calculus: A Simple Solution (sometimes) and a completeness result for goal regression*, pages 359–380. Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy. Academic Press, San Diego, CA, 1991.

[17] Raymond Reiter. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

[18] Earl Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[19] Sebastian Sardina, Giuseppe De Giacomo, Yves Lespérance, and Hector Levesque. On the semantics of deliberation in IndiGolog – from theory to implementation. *Annals of Mathematics and Artificial Intelligence*, 41(2–4):259–299, August 2004. Previous version appeared in Proc. of KR-2002.

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

[21] Tran Cao Son, Phan Huy Tu, and Chitta Baral. Planning with sensing actions and incomplete information using logic programming. In *Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, pages 261–274, Fort Lauderdale, FL, USA, 2004.

[22] Daniel S. Weld, Corin R. Anderson, and David E. Smith. Extending graphplan to handle uncertainty & sensing actions. In *Proceedings of AAAI-98*, pages 897–904, 1998.

A Proofs for section 3

A.1 Proof of Lemma 3.1

The proof is by induction on the structure of δ. Alternatively, we prove that

$$\mathcal{D} \models (\forall \delta, s).\, ssf(\delta, s) \supset \{(\forall s', \delta').\, Trans(\delta, s, \delta', s') \equiv Trans_K(\delta, s, \delta', s')\}$$

Suppose $\mathcal{M} \models \mathcal{D}$. *Base cases:*

1. $\delta = nil$. Trivial.
2. $\delta = \phi?$.
   ($\Rightarrow$) Assume $\mathcal{M} \models ssf(\delta, s) \wedge Trans(\delta, s, \delta', s')$. Then, by definition of *ssf* and *Trans*,
   $$\mathcal{M} \models \mathbf{KWhether}(\phi, s) \wedge \phi[s] \wedge \delta' = nil \wedge s = s'$$
   But $\mathcal{M} \models \mathbf{KWhether}(\phi, s) \wedge \phi[s]$, and the reflexivity of K implies $\mathcal{M} \models \mathbf{Knows}(\phi, s)$, then
   $$\mathcal{M} \models \mathbf{Knows}(\phi, s) \wedge \delta' = nil \wedge s = s'.$$
   which proves this direction.
   ($\Leftarrow$) Trivial from the fact that K is reflexive ($\mathcal{M} \models \mathbf{Knows}(\phi, s) \supset \phi[s]$).
3. $\delta = a$.
   ($\Rightarrow$) Assume $\mathcal{M} \models ssf(\delta, s) \wedge Trans(\delta, s, \delta', s')$. Then, by definition of *ssf* and *Trans*,
   $$\mathcal{M} \models \mathbf{KWhether}(Poss(a), s) \wedge Poss(a, s) \wedge \delta' = nil \wedge s' = do(a, s).$$
   As in the previous case, $\mathcal{M} \models \mathbf{Knows}(Poss(a), s)$, and therefore,
   $$\mathcal{M} \models \mathbf{Knows}(Poss(a), s) \wedge \delta' = nil \wedge s' = do(a, s),$$
   which proves this case.
   ($\Leftarrow$) Trivial from the fact that K is reflexive.

Induction:

1. $\delta = \delta_1 | \delta_2$. Trivial from the inductive hypothesis.
2. $\delta = \delta_1; \delta_2$.
   ($\Rightarrow$) Assume $\mathcal{M} \models ssf(\delta, s) \wedge Trans(\delta, s, \delta', s')$. Then, by definition of *ssf*,
   $$\mathcal{M} \models ssf(\delta_1, s) \wedge \{(\forall s').\, (\exists \delta')\, (Trans^*(\delta_1, s, \delta', s') \wedge Final(\delta', s')) \supset ssf(\delta_2, s')\}$$
   Assuming the antecedent of the implication is true,
   $$\mathcal{M} \models ssf(\delta_1, s) \wedge ssf(\delta_2, s'),$$
   for all s' such that $\mathcal{M} \models (\exists \delta')\, (Trans^*(\delta_1, s, \delta', s') \wedge Final(\delta', s'))$ and using the induction hypothesis, the result follows immediately.
   In case the antecedent of the implication is false, then $\mathcal{M} \models \neg Trans(\delta_1; \delta_2, s, \delta', s')$ for all δ', s'. Furthermore, since $\mathcal{M} \models ssf(\delta_1, s)$ we know by induction hypothesis that
   $$\mathcal{M} \models \neg(\exists \delta')\, (Trans_K^*(\delta_1, s, \delta', s') \wedge Final(\delta', s'))$$
   which also implies that
   $$\mathcal{M} \models \neg Trans(\delta_1; \delta_2, s, \delta', s'),$$
   for all δ', s'.
   ($\Leftarrow$) Analogous to ($\Rightarrow$).
3. $\delta = \pi v.\, \delta$. Trivial by inductive hypothesis.
4. $\delta =$ **if** ϕ **then** δ_1 **else** δ_2 **endif**
   ($\Rightarrow$) Assume $\mathcal{M} \models ssf(\delta, s) \wedge Trans(\delta, s, \delta', s')$. Then, for all situations s in $\mathcal{M}$,
   $$\mathcal{M} \models \mathbf{KWhether}(\phi, s) \wedge (\phi[s] \supset ssf(\delta_1, s)) \wedge (\neg\phi[s] \supset ssf(\delta_2, s)).$$
   Since $\mathcal{M} \models \mathbf{KWhether}(\phi, s)$ we have two cases. First suppose $\mathcal{M} \models \mathbf{Knows}(\phi, s)$. By reflexivity of K, $\mathcal{M} \models ssf(\delta_1, s)$, and the result follows by inductive hypothesis. On the other hand, if $\mathcal{M} \models \mathbf{Knows}(\neg\phi, s)$, the proof is analogous.
   ($\Rightarrow$) Straightforward from the reflexivity of K.
5. $\delta =$ **while** ϕ **do** σ **endW**. Analogous to the previous case.

□

B Proofs for section 4

B.1 Proof of Theorem 4.1

We only need to prove that $\mathsf{Comp}[\mathcal{D},C]$ satisfies the consistency property for all SSAs. We say that a SSA $F(\vec{x},do(a,s)) \equiv \gamma_F^+(a,\vec{x},s) \vee F(\vec{x},s) \wedge \neg\gamma_F^-(a,\vec{x},s)$, satisfies the consistency property (CP) in theory $\mathcal{T}$ if $\mathcal{T} \models (\forall s,\vec{x},a)\neg(\gamma_F^+(a,\vec{x},s) \wedge \gamma_F^-(a,\vec{x},s))$.

In fact if $\mathcal{D}$ is consistent then all its SSAs already satisfy the CP. Suppose $\mathsf{Comp}[\mathcal{D},C]$ does not satisfy the CP for a fluent F. Then, necessarily the following must be true:

$$\mathcal{D} \models (\exists s)\exists s'(Do(\delta,s,s') \wedge F(\vec{x},s')) \wedge \exists s''(Do(\delta,s,s'') \wedge \neg F(\vec{x},s''))$$

since δ is deterministic, the assertion above is equivalent to

$$\mathcal{D} \models (\exists s)\exists s'(Do(\delta,s,s') \wedge F(\vec{x},s') \wedge \neg F(\vec{x},s')),$$

which cannot be true if $\mathcal{D}$ is consistent. We have a contradiction and therefore the SSA for F must satisfy the consistency property.

The proof is analogous if we suppose that a functional fluent f does not satisfy the CP.

B.2 Proof of Lemma 4.1

To prove the equivalence at the fluent level, let $\mathcal{D}' = \mathsf{Comp}[\mathcal{D},C]$. It suffices to prove that

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

To prove 1, using lemma 3.1 we only need to prove that

$$\mathcal{D}' \models (\forall s,\vec{x}).\exists s'(Do(\delta,s,s') \wedge F(\vec{x},s')) \supset F(\vec{x},do(Phys_\delta,s)))$$

Suppose $\mathcal{M}$ is a model of $\mathcal{D}'$ such that $\mathcal{M} \models (\exists s')(Do(\delta,s,s') \wedge F(\vec{x},s'))$. By the successor state axiom generated from (4), and the correctness of regression it follows immediately that $\mathcal{M} \models F(\vec{x},do(Phys_\delta,s)))$. This proves 1.

The proof of 2 is analogous.

Now, for the equivalence at the functional level, it suffices to prove that

1. $\mathcal{D}' \models (\forall s,s',\vec{x},z).Do_K(\delta,s,s') \wedge z = f(\vec{x},s') \supset f(\vec{x},do(Phys_\delta,s)))$, and
2. $\mathcal{D}' \models (\forall s,s',\vec{x}).Do_K(\delta,s,s') \wedge z \neq f(\vec{x},s') \supset z \neq f(\vec{x},do(Phys_\delta,s)))$.

The proof of 1 follows directly from 3.1, the correctness of regression and the successor state axiom for f generated from (6).

To prove 2, assume that $\mathcal{M}$ is a model of $\mathcal{D}'$ such that for arbitrary s, s', $\vec{x}$, and z, $\mathcal{M} \models Do_K(\delta,s,s') \wedge z \neq f(\vec{x},s')$. From the successor state axiom of f we have that

$$z \neq f(\vec{x},do(Phys_\delta,s)) \equiv \neg\gamma(z) \wedge (z \neq f(\vec{x},s) \vee \exists y\gamma(y))$$

where $\gamma(y) \equiv (\exists s').Do(\delta,s,s') \wedge z = f(\vec{x},s')$ by the correctness of regression and lemma 3.1. Therefore $\mathcal{M} \models z \neq f(\vec{x},do(Phys_\delta,s))$ iff (a) $\mathcal{M} \models \neg\gamma(z)$ and (b) $\mathcal{M} \models (z \neq f(\vec{x},s) \vee (\exists y)\gamma(y))$.

For (a), we know that $\mathcal{M} \models \neg\gamma(z)$ iff $\mathcal{M} \models (\forall s').(Do(\delta,s,s') \supset z \neq f(\vec{x},s'))$. Since δ is deterministic this is true by our initial assumption.

For (b), we prove $\mathcal{M} \models (\exists y)\gamma(y)$, which is trivially true by our initial assumption.

B.3 Proof of Theorem 4.3

We first prove the following intermediate results.

Lemma B.1 *Let theory of action $\mathcal{D} = \Sigma \cup \mathcal{K}_{init} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{golog}$. Then,*

$$\mathcal{D} \models K(s',s) \wedge K(\sigma',\sigma) \supset \{(\forall\delta).Do_K(\delta,s,\sigma) \supset Do(\delta,s',\sigma')\}$$

Proof: Without loss of generality, consider that $\mathcal{D} \models K(s',s) \wedge K(\sigma',\sigma) \wedge s' \sqsubseteq \sigma' \wedge s \sqsubseteq \sigma''$, then it is simple to prove by induction that there exists a sequence of actions $\vec{a} = [a_1,\ldots,a_n]$ such that

$$\mathcal{D} \models \sigma' = do(\vec{a},s') \wedge \sigma = do(\vec{a},s) \wedge \bigwedge_{i=0}^{n} K(do([a_1,\ldots,a_i],s'),do([a_1,\ldots,a_i],s))$$

From that fact, and considering that $Do_K(\delta,s,\sigma)$ is equivalent to

$$(\exists\delta_1,\sigma_1)\cdots(\exists\delta_1,\sigma_n)\,Trans_K(\delta,s,\delta_1,\sigma_1) \wedge \cdots \wedge Trans_K(\delta_{n-1},\sigma_{n-1},\delta_n,\sigma_n) \wedge Final(\delta_n,\sigma_n) \wedge \sigma_n = \sigma,$$

it suffices to prove the following.

1. $\mathcal{D} \models K(s',s) \supset \{(\exists s_1)\, Trans_K(\delta,s,\delta',s_1) \supset (\exists s_2)\, Trans(\delta,s',\delta',s_2)\}$.
2. $\mathcal{D} \models K(s',s) \supset \{Final_K(\delta,s) \supset Final(\delta,s')\}$.

Since 2 is trivial by definition, we prove 1 by induction on δ.
Base cases:

1. $\delta = nil$. Trivial.
2. $\delta = \phi?$. Assume $\mathcal{M} \models \mathcal{D}$ and $\mathcal{M} \models K(s',s) \wedge Trans_K(\delta,s,\delta',S_1)$, for some S_1 in $\mathcal{M}$. Then $\mathcal{M} \models \mathbf{Knows}(\phi,s) \wedge \delta' = nil$. By definition, $\mathcal{M} \models \phi[s'] \wedge \delta' = nil$, hence, $\mathcal{M} \models Trans(\delta,s',nil,s')$, which ends the proof for this case.
3. $\delta = a$. Assume $\mathcal{M} \models \mathcal{D}$ and $\mathcal{M} \models K(s',s) \wedge Trans_K(\delta,s,\delta',S_1)$. Then $\mathcal{M} \models \mathbf{Knows}(Poss(a),s) \wedge \delta' = nil$. By definition, $\mathcal{M} \models Poss(a,s') \wedge \delta' = nil$. Therefore, $\mathcal{M} \models Trans(\delta,s',nil,s')$, which ends the proof for this case.

Induction:

1. $\delta = \delta_1;\delta_2$, $\delta = \delta_1;\delta_2$, and $\delta = \pi v.\,\delta$ are trivial from the inductive hypothesis.
2. For the rest of the cases, the proof follows directly from the definition of **Knows** (use the same argument in the base cases) and the inductive hypothesis.

□

Lemma B.2 *Let $\mathcal{D}$ be a the theory of action and C be a set of deterministic Golog tree programs. If $\delta \in C$ and $\mathcal{D} \models (\forall s).\, ssf(\delta,s)$, then,*

$$\mathsf{Comp}[\mathcal{D},C] \models (\forall \vec{x},s,s_1).\, Do_K(\delta,s,s_1) \supset \{(\exists s'')\,(K(s'',s_1) \wedge \phi(\vec{x})[s'']) \equiv (\exists s'')\,(K(s'',do([A^s_\delta, Phys_\delta],s)) \wedge \phi(\vec{x})[s'])\}.$$

Proof: Suppose $\mathcal{D}' = \mathsf{Comp}[D,C]$.
($\Rightarrow$) We prove that

$$\mathcal{D}' \models (\forall \vec{x},s,s_1).\, Do_K(\delta,s,s_1) \supset \{(\exists s'')\,(K(s'',s_1) \wedge F(\vec{x})[s'']) \supset (\exists s'')\,(K(s'',do([A^s_\delta, Phys_\delta],s)) \wedge F(\vec{x})[s''])\},$$

for a situation-suppressed fluent symbol F different from K.
Suppose $\mathcal{M} \models \mathcal{D}'$ and that

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

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

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

From lemma B.1,

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

Given that $\mathcal{M}$ satisfies (7) and (4),

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

Since F is objective,

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

Finally, since $Phys_\delta$ is a physical action, it holds that

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

which finishes the proof for $\Rightarrow$.
($\Leftarrow$) Suppose

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

From the successor state axiom of K,

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

From the SSA of K, $\mathcal{M} \models K(do(Obs_\delta, S'''), do(Obs_\delta, s))$, and since $\mathcal{M}$ satisfies (7),

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

From lemma 3.1 and the fact that δ is deterministic, $\mathcal{M} \models s_1 = S_1$, and therefore,

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

Since F is objective and Obs_δ is a pure sensing action,

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

Now, given that $\mathcal{M}$ satisfies the SSA generated from (7) and (4).

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

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 $\Leftarrow$. □

Proof of the Theorem: Instead, we prove that

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

which follows directly from Lemma B.2. □