On the Semantics of Deliberation in IndiGolog —
From Theory to Implementation

GIUSEPPE DE GIACOMO (degiacomo@dis.uniroma1.it)
Dip. Informatica e Sistemistica, Università di Roma "La Sapienza", Via Salaria 113, 00198 Roma, Italy.

YVES LESPERANCE (lesperan@cs.yorku.ca)
Department of Computer Science, York University, Toronto, ON, M3J 1P3, Canada.

HECTOR J. LEVESQUE (hector@ai.toronto.edu) and SEBASTIAN SARDINA[†] [‡] (ssardina@ai.toronto.edu)
Department of Computer Science, University of Toronto, Toronto, ON, M5S 3G4, Canada.

Abstract. We develop an account of the kind of deliberation that an agent that is doing planning or executing high-level programs under incomplete information must be able to perform. The deliberator's job is to produce a kind of plan that does not itself require deliberation to interpret. We characterize these as *epistemically feasible programs*: programs for which the executing agent, at every stage of execution, by virtue of what it knew initially and the subsequent readings of its sensors, always knows what step to take next towards the goal of completing the entire program. We formalize this notion and characterize deliberation in the situation calculus based IndiGolog agent programming language in terms of it. We also show that for certain classes of problems, which correspond to those with bounded solutions and those with solutions without sensing, the search for epistemically feasible programs can be limited to programs of a simple syntactic form. Finally, we discuss implementation issues and execution monitoring and replanning too.

Keywords: agent programming, models of agency, deliberation, planning with incomplete information

AMS (MOS) classification: 68T27, 68T30, 68T37

1. Introduction

While a large amount of work on planning deals with issues of efficiency, a number of representational questions remain. This is especially true in applications where, because of limitations on the information available at plan time, and quite apart from computational concerns, no *straight-line* plan (that is, no linear sequence of actions) can be demonstrated to achieve a goal. In very many cases, it is necessary to supplement what is known at plan time by information that can only be obtained at run time via sensing.

In cases like these, what should we expect a planner to do given a goal? We cannot expect it to return a straight-line plan. We could get it to return a

[†] Corresponding author

[‡] Many thanks to Marcelo Arenas and Pablo Barcelo for very useful technical discussions.

more general *program* of some sort, but we need to be careful: if the program is general enough, it may be as challenging to figure out how to execute it as it was to achieve the goal in the first place.

This is certainly true for situation calculus high-level programming languages in the family of Golog [17, 4, 25]. These logic languages offer an interesting alternative to planning in which the user specifies not just a goal, but also constraints on how it is to be achieved, perhaps leaving small subtasks to be handled by an automatic planner. In that way, a high-level program serves as a "guide" heavily restricting the search space. In these languages primitive instructions are domain-dependent actions of the robot, tests involve domain-dependent fluents affected by these actions, and the code may contain nondeterministic choice points. Instead of looking for a legal sequence of actions achieving some goal, the (planning) task now is to find a sequence that constitutes a legal execution of a high-level program.

At its most basic, planning should be a form of deliberation, whose purpose is to produce a specification of the desired behavior, a specification which should not itself require deliberation to interpret. In [15] it was suggested that a planner's job was to return a *robot program*, a syntactically-defined structure that a robot could follow while consulting its sensors to determine a conditional course of action. Other forms of conditional plans have been proposed, for example, in [21, 30, 11, 1]. What these all have in common, is that they define plans as *syntactically restricted* programs.

In this paper, we consider a different and more abstract version of plans. We propose to treat plans as *epistemically feasible* programs: programs for which the executing agent, at every stage of execution, by virtue of what it knew initially and the subsequent readings of its sensors, always *knows* what step to take next towards the goal of completing the entire program.

This paper will not present algorithms for generating epistemically feasible programs. What we will do, however, is characterize the notion formally, prove that certain cases of syntactically restricted programs are epistemically feasible, and that in some cases where there is an epistemically feasible program, a syntactically restricted one that has the same outcome can also be derived.

To make these concepts precise, it is useful to consider a framework where we can talk about the planning and execution of very general agent programs involving sensing and acting. IndiGolog [5] is a variant of Golog intended to be executed online in an incremental way. Because of this incremental style execution, an agent program is capable of gathering new information from the world during its execution. Most relevant for our purposes is that IndiGolog includes a *search* operator which allows it to only take a step if it can convince itself that the step will allow it to eventually complete some user-specified subprogram. In that way, IndiGolog provides an attractive integrated account of sensing, planning, and action. However, IndiGolog search

does not guarantee that it will not get stuck in a situation where it knows that some step can be performed, but does not know which. It is this search operator that we will generalize here.

Our proposed account of deliberation is important to the area of agent programming languages (e.g. 3APL [10], AgentSpeak(L) [23], etc.). So far most such languages only provide on-line reactive execution, where no planning is performed (notable exceptions are the temporal logic-based Concurrent MetateM [9] and the fluent calculus-based FLUX [32, 33]). But many agent applications would benefit from planning, especially if incomplete knowledge and sensing were handled (e.g. web service composition).

To illustrate the discussion, we will use a simple example taken from [15]: an agent wants to get on a flight at the airport; however, the agent does not know in advance which gate it must go to; it must acquire this information after it has arrived at the airport, and then proceed to the gate. To perform planning to solve this problem, one could give IndiGolog the following program to execute:

$$
\begin{array}{l}
getOnFlight_{Sketchy} \stackrel{\text{def}}{=} \\
\quad \Sigma(achieve(OnPlane(Flight123), True)) \\
\text{where } achieve(Goal, GoodSit) \stackrel{\text{def}}{=} \\
\quad \textbf{while } \neg Goal \textbf{ do} \\
\quad\quad \pi a[a; GoodSit(\mathsf{now})?] \\
\quad \textbf{endWhile}
\end{array}
$$

Here, $achieve(Goal, GoodSit)$ is a completely general nondeterministic program schema that keeps choosing an action a nondeterministically and executing it for as long as the goal does not hold. ($GoodSit$ is a predicate on situations that that can be used to constrain the search, but in our example this is not used.) We use an appropriate instance of this schema, $achieve(OnPlane(Flight123), True)$, set within the scope of the search operator Σ, to direct IndiGolog to search for a plan that is guaranteed to lead to a situation where the program given can successfully terminate, i.e., where the agent is on its flight. This works provided an adequate axiomatization of the airport domain has been given, which we do in the next section.

We can contrast this very sketchy nondeterministic program with the following one that is completely detailed and determinate (we assume that the

airport has only two gates):

$$getOnFlight_{Detailed} \stackrel{\text{def}}{=}$$
$$go(Airport);$$
$$checkDepartures; \quad \% \text{ sensing action}$$
$$\textbf{if } Parked(Flight123, GateA) \textbf{ then}$$
$$\quad go(GateA); board(Flight123)$$
$$\textbf{else}$$
$$\quad go(GateB); board(Flight123)$$
$$\textbf{endIf}$$

This program could have been defined by the user, or it could have been returned by the planner. Note that without the sensing action *checkDepartures*, the plan cannot be executed since it will not be epistemically feasible anymore! One could also use a program that is less specific than the above but more specific than the first, for instance, one that directs the agent to first achieve being at the airport, then achieve knowing what gate the flight is at, and then achieve being on the flight. In such a framework, the programmer gets to control how much search the interpreter must do. We will return to this example later on.

The rest of the paper is organized as follows. First, in Section 2 we set the stage by presenting the situation calculus and high-level programs based on it. In Section 3, since we are going to make a specific use of the knowledge operator for characterizing the program returned by the deliberator, we introduce *epistemically accurate theories* and some of their basic properties with respect to reasoning. In Section 4, we characterize *epistemically feasible deterministic programs*, i.e., the kind of program that we consider suitable results of the deliberation process, and in Section 5, we study two notable subclasses of epistemically feasible deterministic programs, that can be characterized in terms of syntax only. In Section 6, we discuss how some of the abstract notions we have introduced can be readily implemented in practice. In Section 7, we discuss how the deliberated program could be monitored and revised if circumstances require it. Finally, in Section 8, we draw conclusions and discuss related and future work.

2. The Situation Calculus and IndiGolog

The technical machinery we use to define program execution in the presence of sensing is based on that of [5, 4]. The starting point in the definition is the situation calculus [18]. We will not go over the language here except to note the following components: there is a special constant S_0 used to denote the *initial situation*, namely that situation in which no actions have yet occurred;

there is a distinguished binary function symbol do where $do(a,s)$ denotes the successor situation to s resulting from performing the action a; relations whose truth values vary from situation to situation, are called (relational) *fluents*, and are denoted by predicate symbols taking a situation term as their last argument; and there is a special predicate $Poss(a,s)$ used to state that action a is executable in situation s. Actions may be ordinary physical actions though which the agent changes its environment or sensing actions through which he acquires new information. In this paper, we only deal explicitly with sensing actions with binary outcomes as in [15]. However, the results presented here can be easily generalized to sensors with multiple outcomes. We use a predicate $SF(a,s)$ to characterize what the action tells the agent about the environment. For a sensing action $sense_\phi$ that senses the truth value of ϕ, we would have $[SF(sense_\phi,s) \equiv \phi(s)]$, and for any ordinary action a that does not involve sensing, we would have $[SF(a,s) \equiv True]$. We assume that $SF(a,s)$ holds if and only if action a returns the binary sensing result 1 in situation s. When the agent performs a sensing action a in situation s, his knowledge base/theory will be expanded with either $SF(a,s)$ or its negation.

Within this language, we can formulate domain theories which describe how the world changes as the result of the available actions. One possibility is an action theory $\mathcal{D}$ of the following form (see [25] for details):

- $\mathcal{D}_{ap}$ is the set of *action precondition axioms*, one for each primitive action a, characterizing $Poss(a,s)$.
- $\mathcal{D}_{ss}$ is the set of *successor state axioms*, one for each fluent F, stating under what conditions $F(\vec{x},do(a,s))$ holds as a function of what holds in situation s; these take the place of effect axioms, but also provide a solution to the frame problem [24].
- $\mathcal{D}_{sf}$ is the set of *sensed fluent axioms*, one for each primitive action a of the form $SF(a,s) \equiv \phi_a(s)$, characterizing SF [15].
- $\mathcal{D}_{una}$ is the set of unique names axioms for the primitive actions.
- $\mathcal{D}_{S_0}$ is the set of axioms describing the initial situation, S_0.
- Some foundational, domain independent axioms [25].

For instance, for our airport example, we could use the following action theory:[1]

- Precondition axioms:
  $Poss(go(x),s) \equiv x = Airport \vee At(Airport,s)$

[1] We omit here unique name axioms for constants, as well as domain closure axioms, including one saying that gate A and gate B are the only gates.

$Poss(board(p),s) \equiv \exists x.Parked(p,x,s) \wedge At(x,s)$
$Poss(checkDepartures) \equiv \neg At(Home,s)$

– Successor state axioms:
$At(x,do(a,s)) \equiv a = go(x) \vee At(x,s) \wedge \neg \exists y\, a = go(y)$
$OnPlane(p,do(a,s)) \equiv a = board(p) \vee OnPlane(p,s)$
$Parked(p,x,do(a,s)) \equiv Parked(p,x,s)$

– Sensed fluent axioms:
$SF(go(x),s) \equiv TRUE$, $SF(board(p),s) \equiv TRUE$,
$SF(checkDepartures,s) \equiv Parked(Flight123,GateA,s)$

To describe a run which includes both actions and their sensing results, we use the notion of a *history*. A history is a sequence of pairs (a,x) where a is a primitive action and x is 1 or 0, a sensing result. Intuitively, the history $\sigma = (a_1,x_1) \cdot \ldots \cdot (a_n,x_n)$ is one where actions $a_1,\ldots,a_n$ happen starting in some initial situation, and each action a_i returns sensing value x_i. We assume that if a_i is an ordinary action with no sensing, then $x_i = 1$. For example in the airport domain,

$$\sigma_1 = (go(Airport),1) \cdot (checkDepartures,0) \cdot (go(GateB),1)$$

would be a possible history, where the agent first goes to the airport, then senses the departure screen and gets a sensing result of 0, meaning that the flight is not at gate A, and then goes to gate B. Notice that the empty sequence ε is a history.

We use $end[\sigma]$ as an abbreviation for the situation term called the *end situation* of history σ on the initial situation S_0, and defined by: $end[\varepsilon] = S_0$; and inductively, $end[\sigma \cdot (a,x)] = do(a,end[\sigma])$. So for example:

$$end[\sigma_1] = do(go(GateB),do(checkDepartures,do(go(Airport),S_0))).$$

We also use $Sensed[\sigma]$ as an abbreviation for a formula of the situation calculus, the *sensing results* of a history, and defined by: $Sensed[\varepsilon] = True$; and inductively, $Sensed[\sigma \cdot (a,1)] = Sensed[\sigma] \wedge SF(a,end[\sigma])$, and $Sensed[\sigma \cdot (a,0)] = Sensed[\sigma] \wedge \neg SF(a,end[\sigma])$. This formula uses SF to tell us what must be true for the sensing to come out as specified by σ starting in S_0. So for example, $Sensed[\sigma_1]$ stands for:

$SF(go(Airport),S_0) \wedge$
$\neg SF(checkDepartures,do(go(Airport),S_0)) \wedge$
$SF(go(GateB),do(checkDepartures,do(go(Airport),S_0)))$

which is equivalent to $\neg Parked(Flight123,GateA)$.

Next we turn to programs. The programs we consider here are based on the ConGolog language defined in [4], which provides a rich set of programming constructs summarized below:

α,	primitive action
$\phi?$,	wait for a condition
$\delta_1;\delta_2$,	sequence
$\delta_1 \mid \delta_2$,	nondeterministic branch
$\pi x.\delta$,	nondeterministic choice of argument
δ^*,	nondeterministic iteration
if ϕ **then** δ_1 **else** δ_2 **endIf**,	conditional
while ϕ **do** δ **endWhile**,	while loop
$\delta_1 \parallel \delta_2$,	concurrency with equal priority
$\delta_1 \rangle\rangle \delta_2$,	concurrency with δ_1 at a higher priority
$\delta^{\parallel}$,	concurrent iteration
$\langle\, \vec{x} : \phi \rightarrow \delta \,\rangle$,	interrupt
$p(\vec{\theta})$,	procedure call[2]

Among these constructs, we notice the presence of nondeterministic constructs. These include $(\delta_1 \mid \delta_2)$, which nondeterministically chooses between programs δ_1 and δ_2, $\pi x.\delta$, which nondeterministically picks a binding for the variable x and performs the program δ for this binding of x, and δ^*, which performs δ zero or more times. Also notice that ConGolog includes constructs for dealing with concurrency. In particular $(\delta_1 \parallel \delta_2)$ expresses the concurrent execution (interpreted as interleaving) of the programs δ_1 and δ_2. Beside $(\delta_1 \parallel \delta_2)$ ConGolog includes other constructs for dealing with concurrency, such as prioritized concurrency $(\delta_1 \rangle\rangle \delta_2)$, and interrupts $\langle\, \vec{x} : \phi \rightarrow \delta \,\rangle$. We refer the reader to [4] for a detailed account of ConGolog.

In [4], a single step transition semantics in the style of [22] is defined for ConGolog programs. Two special predicates *Trans* and *Final* are introduced. $Trans(p,s,p',s')$ means that by executing program p starting in situation s, one can get to situation s' in one elementary step with the program p' remaining to be executed, that is, there is a possible transition from the configuration (p,s) to the configuration (p',s'). $Final(p,s)$ means that program p may successfully terminate in situation s, i.e., the configuration (p,s) is final.[3]

Offline executions of programs, which are the kind of executions originally proposed for Golog and ConGolog [17, 4], are characterized using the $Do(p,s,s')$ predicate, which means that there is an execution of program p

[2] For the sake of simplicity, we will not consider procedures in this paper.

[3] For example, the transition requirements for sequence are

$$Trans([p_1;p_2],s,p',s') \equiv$$
$$Final(p_1,s) \wedge Trans(p_2,s,p',s') \vee \exists q'.Trans(p_1,s,q',s') \wedge p' = (q';p_2)$$

i.e., to single-step the program $(p_1;p_2)$, either p_1 terminates and we single-step p_2, or we single-step p_1 leaving some q', and $(q';p_2)$ is what is left of the sequence. Note that since *Trans* and *Final* take programs (that include test of formulas) as arguments, this requires encoding formulas and programs as terms; see [4] for the details. For notational simplicity, we suppress this encoding and use programs as terms directly.

that starts in situation s and terminates in situation s':

$$Do(p,s,s') \stackrel{\text{def}}{=} \exists p'.Trans^*(p,s,p',s') \wedge Final(p',s'),$$

where $Trans^*$ is the reflexive transitive closure of $Trans$, i.e.

$$\begin{aligned} Trans^*(\delta,s,\delta',s') \stackrel{\text{def}}{=} \\ \forall T.[\forall \delta_1,s_1.T(\delta,s,\delta,s) \wedge \\ \forall \delta_1,s_1.\delta_2,s_2,\delta_3,s_3(Trans(\delta_1,s_1,\delta_2,s_2) \wedge \\ T(\delta_2,s_2,\delta_3,s_3) \supset T(\delta_1,s_1,\delta_3,s_3)) \\ \supset T(\delta,s,\delta',s')]. \end{aligned}$$

From now on, $\mathcal{D}$ will denote the set of axioms defining an underlying theory of action, $\mathcal{T}$ will denote the set of axioms for $Trans$ and $Final$, and $\mathcal{E}$ will stand for the set of axioms needed for the encoding of programs as first-order terms (see [4]). An offline execution of program p from situation s is a sequence of actions $a_1,\ldots,a_n$ such that:

$$\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \models Do(p,s,do(a_n,\ldots,do(a_1,s)))$$

Observe that an offline executor is in fact similar to a planner that given a program, a starting situation, and a theory describing the domain, produces a sequence of action to execute in the environment. In doing this, it has no access to sensing results, which will only be available at runtime. See [4] for more details.

In [5], IndiGolog, an extension of ConGolog that deals with online executions with sensing is developed. We say that a configuration, this time formed by a program and a history, (p,σ) *may evolve* to configuration (p',σ') w.r.t. a model M of $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$ if and only if[4]

$$\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_i]\} \models Trans(p,end[\sigma],p',end[\sigma'])$$

and

$$\sigma' = \begin{cases} \sigma & \text{if } end[\sigma'] = end[\sigma], \\ \sigma \cdot (a,1) & \text{if } end[\sigma'] = do(a,end[\sigma]) \\ & \quad \text{and } M \models SF(a,end[\sigma]) \\ \sigma \cdot (a,0) & \text{if } end[\sigma'] = do(a,end[\sigma]) \\ & \quad \text{and } M \not\models SF(a,end[\sigma]) \end{cases}$$

Finally, we say that a configuration (p,σ) is *final* whenever

$$\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models Final(p,end[\sigma])$$

[4] This definition is more general than the one in [5], where the sensing results were assumed to come from the actual environment rather than from a model (a model can represent any possible environment). Also, here we deal with non-terminating, i.e., infinite executions.

We now define several kinds of online executions. A *non-terminating online execution* of an IndiGolog program p starting from a history σ w.r.t. a model M of $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$ is an infinite sequence of *online configurations* $(p_0 = p, \sigma_0 = \sigma), (p_1, \sigma_1), \ldots$, such that configuration (p_i, σ_i) may evolve to configuration (p_{i+1}, σ_{i+1}) w.r.t. model M for every $i \geq 0$.

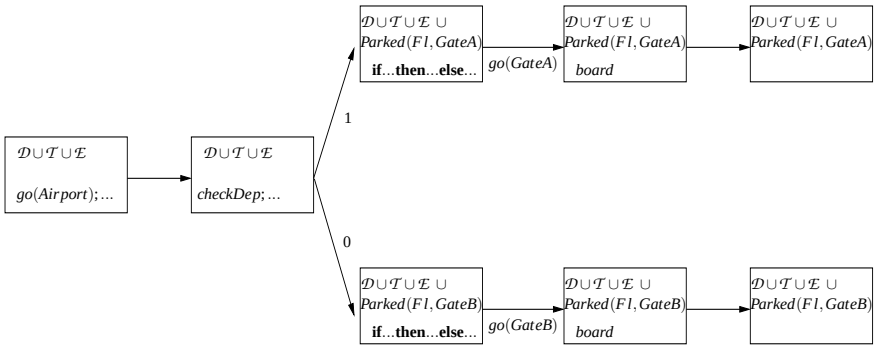
On the other hand, a *terminating online execution* of an IndiGolog program p starting from a history σ w.r.t. a model M of $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$ is a finite sequence of *online configurations* $(p_0 = p, \sigma_0 = \sigma), \ldots, (p_n, \sigma_n)$ such configuration (p_i, σ_i) may evolve to configuration (p_{i+1}, σ_{i+1}) w.r.t. model M for every $0 \leq i \leq n-1$, and either (p_n, σ_n) is a final configuration or (p_n, σ_n) is not a final configuration and there is no configuration (p', σ') to which (p_n, σ_n) may evolve w.r.t. M. In the former case, we say that the online execution *successfully* terminates; in the latter case, we say that the online execution is *stuck* or has reached a *dead-end*. Finally, we say that an online execution is *complete* if it is either a non-terminating or a terminating execution.

The following lemma says that the model used to generate sensing outcomes is always a model of the theory at every step of the online execution.

LEMMA 1. *If $(p_0 = p, \sigma_0 = \sigma), \ldots, (p_n, \sigma_n)$ is an online execution of program p at σ w.r.t. a model M of $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$, then M is a model of $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_i]\}$ for all $i \geq 0$.*

Proof. Trivial since M is a model of $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$ and every sentence $SF(A,S)$ added to such set at every online step, where A is an ground action term and S is a ground action term, is also satisfied by M.

So for the example program $getOnFlight_{Detailed}$, we would have the following tree of online executions:



Depending on the result of the *checkDepartures* sensing action, the theory gets updated differently and different online executions ensue.

There is no automatic lookahead in IndiGolog. Instead, a *search* operator $\Sigma(p)$ is introduced to allow the programmer to specify when lookahead

should be performed. *Final* and *Trans* are defined for the new operator as follows. For *Final*, we simply have that $(\Sigma(p),s)$ is a final configuration of the program if (p,s) itself is, i.e.,

$$Final(\Sigma(p),s) \equiv Final(p,s)$$

For *Trans*, we have that the configuration $(\Sigma(p),s)$ can evolve to $(\Sigma(q'),s')$ provided that (p,s) can evolve to (q',s') and from (q',s') it is possible to reach a final configuration in a finite number of transitions, i.e.,

$$Trans(\Sigma(p),s,p',s') \equiv \exists q',s_f.\ p' = \Sigma(q') \wedge Trans(p,s,q',s') \wedge Do(q',s',s_f)$$

This semantics means that the set of axioms $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$ entails $Trans(\Sigma(p),end[\sigma],\Sigma(p'),s')$ if and only if it entails $Trans(p,end[\sigma],p',s')$ as well as $\exists s_f.Do(p',s',s_f)$. Thus, with this definition, the axioms entail that a step of the program can be performed provided that they entail that this step can be extended into a complete execution (i.e., in all models). This prunes executions that are bound to fail later on. But it does not guarantee that the executor will not get stuck in a situation where it knows that some transition can be performed, but does not know which. For example, consider the program $(a;$ **if** ϕ **then** b **else** $c) \mid d$, where actions *a*, *b*, *c*, and *d* are always possible, but where the agent does not know whether ϕ holds after *a*. There are two possible first steps, *d* which terminates successfully, and *a* after which the executor is stuck. Unfortunately, Σ does not distinguish between the two cases, since even in the latter, there does exist an (unknown) transition to a final state. We address this problem in our account of deliberation of Section 4.

3. Epistemically Accurate Theories

As mentioned in the introduction, our account of deliberation is based on the agent finding a plan which is an epistemically feasible program, a program for which he always knows what step to do next. To formalize this notion, we will use action theories that are extended with a knowledge operator. Our goal in introducing knowledge is only to be able to refer in the language to what the agent knows after a sequence of sensing actions (which we did metatheoretically in the previous section). So we only consider theories that are *epistemically accurate*, meaning that, among other things, what is known accurately reflects what the theory says about the dynamic system.[5]

To represent knowledge in the language, we follow [20, 29, 15] and use a fluent $K(s',s)$ to specify which situations s' are considered epistemically

[5] In [26] and [25], chapter 11, a similar notion is used to deal with knowledge-based programs and reduce knowledge to provability.

possible by the agent in situation s. $\mathbf{Know}(\phi(\mathsf{now}), s)$ is then taken to be an abbreviation for the formula $\forall s'.K(s', s) \supset \phi(\mathsf{now}/s')$.

First we introduce the notion of *objective formula* (cf. [25], chapter 11). Intuitively, an objective formula on a situation $\bar{s}$ is one that only talks about the world (not the knowledge of it) in the situation $\bar{s}$. Formally, objective formulas on situation $\bar{s}$ are inductively defined as follows:[6]

- If $F(\vec{t}, \bar{s})$ is a relational atom, then $F(\vec{t}, \bar{s})$ is an objective formula on $\bar{s}$;
- If t_1, t_2 are terms not of sort situation (that is, object, action or program terms,) then $t_1 = t_2$ is an objective formula on $\bar{s}$;
- If ϕ_1 and ϕ_2 are objective formulas on $\bar{s}$ then so are $\neg\phi$, $\phi_1 \wedge \phi_2$, and $\exists x \phi_1$, where x is a variable not of sort situation.

Note that neither *Trans* nor *Final* can be mentioned in objective formulas.

Epistemically accurate theories are theories as introduced earlier, but with the following additional constraints:

1. The initial situation is characterized by an axiom of the form $\mathbf{Know}(\psi_0(\mathsf{now}), S_0)$, that is, $\mathcal{D}_{S_0} = \{\mathbf{Know}(\psi_0(\mathsf{now}), S_0)\}$, where $\psi_0(\mathsf{now})$ is an *objective formula* on the situation denoted by *now*. Note that there can be fluents about which nothing is known in the initial situation.
2. Every sensing axiom $SF(A(\vec{x}), s) \equiv \psi(\vec{x}, s)$ is such that $\psi(\vec{x}, \mathsf{now})$ is an objective formula.
3. Every precondition axiom $Poss(\vec{x}, s) \equiv \psi(\vec{x}, s)$ is such that $\psi(\vec{x}, \mathsf{now})$ is an objective formula.
4. The set of successor state axioms $\mathcal{D}_{ss}$ includes the following successor state axiom for the knowledge fluent K [29]:

   $$K(s'', do(a, s)) \equiv \exists s'.s' = do(a, s') \wedge K(s', s) \wedge [SF(a, s') \equiv SF(a, s)].$$

   All other successor state axioms $F(\vec{x}, do(a, s)) \equiv \psi(\vec{x}, s)$ are such that $\psi(\vec{x}, \mathsf{now})$ is an objective formula.
5. There is an axiom $\mathcal{K}_{Init}$ stating that the accessibility relation K is, at least, reflexive in the initial situation, which is then propagated to all situations by the successor state axiom for K [29].
6. There are no functional fluents, as before, and no non-fluent relations except for equality, $\sqsubset$ and *Poss*.[7]

[6] Notice that, in contrast to [25], our objective formulas include equality between program terms.

[7] Note that, it is straightforward to represent non-fluent relations using "eternal" relational fluents. Also, functional fluents can also represented using relational fluents.

7. $\mathcal{D}_{una} \cup \psi_0(S_0)$ decides all equality sentences not mentioning any program term or any program variable, that is, for any sentence over the language of the theory whose only predicate symbol is equality and such that β mentions no program term or variable, $\mathcal{D}_{una} \cup \psi_0(S_0) \models \beta$ or $\mathcal{D}_{una} \cup \psi_0(S_0) \models \neg\beta$.

8. There are a finite number of action types $A_1(\vec{x_1}), ..., A_n(\vec{x_n})$, and the agent knows this. Formally,

$$\psi_0(S_0) \models \forall a.[\exists \vec{x_1}.a = A(\vec{x_1}) \vee ... \vee \exists \vec{x_n}.a = A(\vec{x_n})]$$

9. There are domain closure and unique name axioms for objects, and the agent knows this. Formally,

$$\begin{aligned} \psi_0(S_0) &\models \forall x.(\forall R.[R(0) \wedge (\forall y.R(y) \supset R(\varsigma(y))) \supset R(x)]) \\ \psi_0(S_0) &\models \forall x.0 \neq \varsigma(x) \wedge x \neq \varsigma(x) \end{aligned}$$

   This forces the object domain to be isomorphic to the countably infinite set of standard names $0, \varsigma(0), \varsigma(\varsigma(0)), ...$ (see [16]).[8]

Observe that because of assumption 7, whenever we have a program of the form $\pi a.\delta(a)$, where a is an action, we can rewrite it (without loss of generality) as program $\pi\vec{x_1}.\delta(A_1(\vec{x_1}))|...|\pi.\vec{x_n}.\delta(A_n(\vec{x_n}))$ assuming $A_1, ..., A_n$ are all the action types available. This shows that we do not need to deal with existential quantification over action variables, since we can replace nondeterministic choice of action by a nondeterministic branch over all the available action types. It should be clear that any action theory of the form specified in Section 2 that satisfies restrictions 6 to 9 can be transformed into an epistemically accurate theory. Note that we also require that tests appearing in programs be objective formulas that do not mention program terms.

From now on, we will restrict to particular types of theories $\mathcal{D}$ respecting the above assumptions. First we show that every occurrence of *Trans* and *Final* can be substituted by an equivalent objective formula.

THEOREM 1. *For any ConGolog program term $p(\vec{x})$ containing only variables $\vec{x}$ of sort object, there exist objective formulas $\phi_f(\vec{x}, s), \phi_{tt}(\vec{x}, p', s)$, and $\phi_{ta}(\vec{x}, p', s)$ containing no free variables other than the ones listed, not mentioning Trans, nor Final, and such that:*

$$\begin{aligned} \mathcal{D} \cup \mathcal{T} \cup \mathcal{E} &\models Final(p(\vec{x}, s) \equiv \phi_f(\vec{x}, s) \\ \mathcal{D} \cup \mathcal{T} \cup \mathcal{E} &\models Trans(p(\vec{x}), s, p', s) \equiv \phi_{tt}(\vec{x}, p', s) \\ \mathcal{D} \cup \mathcal{T} \cup \mathcal{E} &\models Trans(p(\vec{x}), s, p', do(a, s)) \equiv \phi_{ta}(\vec{x}, p', a, s) \end{aligned}$$

[8] For simplicity, we assumed these sentences to be entailed by $\psi_0(S_0)$, but, since they are situation-independent sentences, they can very well be included in $\mathcal{D}_{S_0}$ and not in $\psi_0(S_0)$.

Proof. Straightforward by induction on the structure of the programs.[9] See appendix.

Next, we show some basic properties of epistemically accurate theories that will be used in the following. The first says that if some objective property of the system is entailed, then it is also known and vice-versa.

THEOREM 2. *Let $\phi(s)$ be an objective formula on situation s. Then,*

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \phi(end[\sigma])$$
if and only if
$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \mathbf{Know}(\phi, end[\sigma])$$

Proof. See appendix.

The next two results tell us that, in some sense, what we know about the agent's knowledge is "complete."

THEOREM 3. *Let $\phi_i(\mathsf{now})$, $i = 1..n$ be objective formulas. Then,*

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \mathbf{Know}(\phi_1(\mathsf{now}), end[\sigma]) \vee ... \vee \mathbf{Know}(\phi_n(\mathsf{now}), end[\sigma])$$

if and only if $\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \mathbf{Know}(\phi_k(\mathsf{now}), end[\sigma])$, for some $1 \leq k \leq n$.

Proof. See appendix.

THEOREM 4. *Let $\phi(\vec{x}, s)$ be an objective formula on situation s with non-situation free variables $\vec{x}$. Then*

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \exists \vec{x}.\mathbf{Know}(\phi(\vec{x}, \mathsf{now}), end[\sigma])$$

if and only if there are ground terms $\vec{t}$ such that

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \mathbf{Know}(\phi(\vec{t}, \mathsf{now}), end[\sigma]).$$

Proof. See appendix.

Finally, under some restrictions on what is known, we can combine the previous two theorems into a single result.

THEOREM 5. *Let $\phi_1(\mathsf{now})$, $\phi_2(\vec{x}, \mathsf{now})$, and $\phi_3(\vec{y}, \mathsf{now})$ be three objective formulas with non-situation free variables $\vec{x}$ and $\vec{y}$. If*

$$\begin{aligned}\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models\ & \mathbf{Know}(\phi_1(\mathsf{now}), end[\sigma]) \vee \\ & \exists \vec{x}.\mathbf{Know}(\neg\phi_1(\mathsf{now}) \wedge \phi_2(\vec{x}, \mathsf{now}) \wedge \forall \vec{y}.\neg\phi_3(\vec{y}, \mathsf{now}), end[\sigma]) \vee \\ & \exists \vec{y}.\mathbf{Know}(\neg\phi_1(\mathsf{now}) \wedge \forall \vec{x}.\neg\phi_2(\vec{x}, \mathsf{now}) \wedge \phi_3(\vec{y}, \mathsf{now}), end[\sigma])\end{aligned}$$

then $\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$ entails one of the following closed formulas:

[9] Remember that we are not allowing for recursive procedures in this paper.

1. $\mathbf{Know}(\phi_1(\mathsf{now}), end[\sigma])$*;*

2. $\mathbf{Know}(\phi_2(\vec{t_2}, \mathsf{now}), end[\sigma])$*, for some ground terms* $\vec{t_2}$*;*

3. $\mathbf{Know}(\phi_3(\vec{t_3}, \mathsf{now}), end[\sigma])$*, for some ground terms* $\vec{t_3}$*.*

Proof. The theorem follows easily as a consequence of Lemma 7 in the Appendix (Section A) and of Theorem 4.

4. Deliberation Program Steps

We are going to introduce and semantically characterize the deliberation steps in the program. The basic idea of the semantics we are going to develop is that the task of the deliberator (that performs search) is, given a possible highly nondeterministic program, to try to find a deterministic program that is guaranteed to be "executable" and constitutes a way to execute the original program, in the sense that it always leads to terminating situations of the original program. Another way to look at this is that the deliberator tries to identify a "strategy" for reaching a final situation of the original program. In such a strategy, all choices must be resolved, i.e., the corresponding program needs to be deterministic, and only information that is available to the executor may be used (e.g., to branch on). In doing this task, the deliberator performs essentially the same task as the offline executor: it compiles the original program into a simpler program that can be executed without any lookahead. The program it produces however, is not just a linear sequence of actions; it can perform sensing, branching, iteration, etc. Moreover, the program is checked to ensure that the executor will always have enough information to continue the execution. Among other things, this addresses the problem raised above concerning the original semantics of search: getting stuck because of lack of knowledge on which transition to perform next. Note that our approach is similar to that of [15]; however, there the strategy was stated in a completely different language (robot programs), here we use ConGolog, i.e., the language used to program the agent itself.

4.1. Epistemically Feasible Deterministic Programs

The first step in developing this approach is formalizing the notion mentioned above of a deterministic program for which an executor will always have enough information to continue the execution, i.e., will always know what the next step to be performed is. We capture this notion formally by defining the class of *epistemically feasible deterministic programs (EFDPs)* as follows:

$$EFDP(dp,s) \stackrel{\text{def}}{=} \forall dp',s'.Trans^*(dp,s,dp',s') \supset LEFDP(dp',s')$$

$$LEFDP(dp,s) \stackrel{\text{def}}{=}$$
$$\mathbf{Know}(Final(dp,\mathsf{now}) \wedge \neg\exists dp',s'.Trans(dp,\mathsf{now},dp',s'),s) \vee$$
$$\exists dp'.\mathbf{Know}(\neg Final(dp,\mathsf{now}) \wedge UTrans(dp,\mathsf{now},dp',\mathsf{now}),s) \vee$$
$$\exists dp',a.\mathbf{Know}(\neg Final(dp,\mathsf{now}) \wedge UTrans(dp,\mathsf{now},dp',do(a,\mathsf{now})),s)$$

$$UTrans(dp,s,dp',s') \stackrel{\text{def}}{=}$$
$$Trans(dp,s,dp',s') \wedge \forall dp'',s''.Trans(dp,s,dp'',s'') \supset dp'' = dp' \wedge s'' = s'$$

Thus to be an *EFDP*, a program must be such that all configurations reachable from the initial program and situation are such that the program is a locally epistemically feasible deterministic one (*LEFDP*). A program is an *LEFDP* in a situation if the agent knows that it is currently *Final* and there are no further transitions possible, or it knows what unique transition (with or without an action) it can perform next.

Our original detailed program for getting on a flight $getOnFlight_{Detailed}$ is an *EFDP*: the agent knows what action it must do first, go to the airport, then it knows what to do next, check the departures screen, which will tell it which gate the flight is at, and then it knows it must go to that gate, board the flight, and then knows that it is done. If we delete the sensing action *checkDepartures* from the program, then we no longer have an *EFDP*; the agent no longer knows what action to do next at the test because it does not know which gate the flight is at and the "then" and "else" branches of the program involve different actions.

First, observe that even though an epistemically feasible deterministic program is not required to terminate, the agent is guaranteed to know what to do next at every step in its execution. As a consequence of that, online executions of an epistemically feasible deterministic program can never get to a configuration where the agent does not know what to do next and the execution is stuck.

THEOREM 6. *Let dp be such that $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models EFDP(dp,end[\sigma])$. Then, for each model M of $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$, there is only one complete online execution of dp from σ w.r.t. M and and this execution is either non-terminating or successfully terminating.*

Proof. First we show, by contradiction, that for all models M of $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$ all online executions of dp from σ w.r.t. M are either non-terminating or successfully terminating.

Suppose there is a model M and an online execution that gets stuck in an online configuration (dp_i,σ_i) where neither *Final* nor *Trans* to some subsequent configuration are entailed. This means that $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_i]\} \not\models Final(dp_i,\sigma_i)$ and there there are no terms dp_{i+1} and σ_{i+1} to which configuration (dp_i,σ_i) can make a transition w.r.t. M.

Now since we have an online execution w.r.t. $\bar{M}$ reaching configuration (dp_i, σ_i), $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models Trans^*(dp, end[\sigma], dp_i, end[\sigma_i])$ follows, hence since by hypothesis $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$ entails $EFDP(dp, end[\sigma])$, we have that $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_i]\}$ entails $LEFDP(dp_i, end[\sigma_i])$ and, thus, by definition of *LEFDP* we have:

$$\begin{array}{l}\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_i]\} \models \\ \mathbf{Know}(Final(dp_i, \mathsf{now}) \wedge \neg \exists dp', s' \, Trans(dp_i, \mathsf{now}, dp', s'), end[\sigma_i]) \vee \\ \exists dp' \mathbf{Know}(\neg Final(dp_i, \mathsf{now}) \wedge UTrans(dp_i, \mathsf{now}, dp', \mathsf{now}), end[\sigma_i]) \vee \\ \exists dp', a \mathbf{Know}(\neg Final(dp_i, \mathsf{now}) \wedge UTrans(dp_i, \mathsf{now}, dp', do(a, \mathsf{now})), end[\sigma_i])\end{array}$$

By Theorem 5, and the fact that it is possible to eliminate all references to *Final* and *Trans* by equivalent objective formulas (Theorem 1), it is possible to show that this implies that one of the logical implications below must hold:

(a) $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_i]\} \models$
$\mathbf{Know}(Final(dp_i, \mathsf{now}) \wedge \neg \exists dp', s' \, Trans(dp_i, \mathsf{now}, dp', s'), end[\sigma_i])$,

(b) $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_i]\} \models$
$\exists dp'.\mathbf{Know}(\neg Final(dp_i, \mathsf{now}) \wedge UTrans(dp_i, \mathsf{now}, dp', \mathsf{now}), end[\sigma_i])$,

(c) $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_i]\} \models$
$\exists dp', a.\mathbf{Know}(\neg Final(dp_i, \mathsf{now}) \wedge$
$UTrans(dp_i, \mathsf{now}, dp', do(a, \mathsf{now})), end[\sigma_i])$.

Taking into account Theorem 4, and again using Theorem 1 to eliminate all references to *Trans* and *Final* predicates, we have one of the following cases:

(a) $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_i]\} \models$
$\mathbf{Know}(Final(dp_i, \mathsf{now}) \wedge \neg \exists dp', s' \, Trans(dp_i, \mathsf{now}, dp', s'), end[\sigma_i])$,

(b) $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_i]\} \models \mathbf{Know}(UTrans(dp_i, \mathsf{now}, \bar{dp'}, \mathsf{now}), end[\sigma_i])$,
for some ground program term $\bar{dp'}$,

(c) $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_i]\} \models \mathbf{Know}(UTrans(dp_i, \mathsf{now}, \bar{dp'}, do(\bar{a}, \mathsf{now})), end[\sigma_i])$, for some ground program term $\bar{dp'}$ and ground action term $\bar{a}$.

Lastly, by reflexivity of K one of the following cases applies:

(a) $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_i]\} \models Final(dp_i, end[\sigma_i])$,

(b) $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_i]\} \models UTrans(dp_i, end[\sigma_i], \bar{dp'}, end[\sigma_i])$,

(c) $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_i]\} \models UTrans(dp_i, end[\sigma_i], \bar{dp'}, do(\bar{a}, end[\sigma_i]))$.

In case (a), the configuration (dp_i, σ_i) is a final one. In case (b), the configuration (dp_i, σ_i) can make a legal non-action online execution step w.r.t. $\bar{M}$ to configuration $(\bar{dp'}, \sigma_i)$. Finally, in case (c), the configuration (dp_i, σ_i) can make a legal online step to $(\bar{dp'}, \sigma_{i+1})$, such that $\sigma_{i+1} = \sigma_i \cdot (\bar{a}, \mu)$, where $\mu = 1$ if $\bar{M} \models SF(\bar{a}, end[\bar{\sigma}])$, and $\mu = 0$, otherwise.

Therefore, in all three cases configuration (dp_i, σ_i) is not stuck, i.e., it is either final or it can evolve to another configuration, thus getting a contradiction.

Next we show, again by contradiction, that for all models M of $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$ there is only one complete execution of dp from σ w.r.t. M.

Assume that there are two *different* complete online executions EX_1 and EX_2 of dp at σ w.r.t. a certain model $\bar{M}$:

$$\begin{aligned} EX_1 &= (dp, \sigma), (dp_1^1, \sigma_1^1), ... \\ EX_2 &= (dp, \sigma), (dp_1^2, \sigma_1^2), ... \end{aligned}$$

As EX_1 is different from Ex_2, then either EX_1 is a prefix execution of EX_2, EX_2 is a prefix execution of EX_1, or for some $i \geq 1$ it is the case that $(dp_j^1, \sigma_j^1) = (dp_j^2, \sigma_j^2)$ for all $j < i$, but $(dp_i^1, \sigma_i^1) \neq (dp_i^2, \sigma_i^2)$. Clearly, EX_1 (EX_2) is not a *complete* execution in the first (second) case, because its last configuration does have a transition and, given that it is a local epistemically feasible configuration, it can never be final. Then, the only possible case is the third one. However, in that case, we must have that:

$$\begin{aligned} \mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_{i-1}]\} &\models Trans(dp_{i-1}^1, end[\sigma_{i-1}^1], dp_i^1, end[\sigma_i^1]) \\ \mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_{i-1}]\} &\models Trans(dp_{i-1}^2, end[\sigma_{i-1}^2], dp_i^2, end[\sigma_i^2]) \end{aligned}$$

where $\sigma_{i-1} = \sigma_{i-1}^1 = \sigma_{i-1}^2$ and $dp_{i-1} = dp_{i-1}^1 = dp_{i-1}^2$. Because $(dp_i^1, \sigma_i^1) \neq (dp_i^2, \sigma_i^2)$, there is no *unique* transition, formally,

$$\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_{i-1}]\} \models \neg \exists dp', s'.UTrans(dp_{i-1}, end[\sigma_{i-1}], dp', s') \quad (1)$$

However, given that program dp is an *EFDP* at history σ, it is the case that $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_{i-1}]\} \models LEFDP(dp_{i-1}, end[\sigma_{i-1}])$. Observe that, by Theorem 2, the agent knows of the possible two transitions for (dp_{i-1}, σ_{i-1}) at σ_{i-1}, and, as dp_{i-1} is a *LEFDP* at σ_{i-1} the agent knows the configuration is *not* a final one. By Theorems 4 and 1 there exists a ground program term $\bar{dp'}$ and a ground situation term $\bar{s'}$ (with $\bar{s'} = end[\sigma_{i-1}]$ or $\bar{s'} = do(\bar{a}, end[\bar{\sigma}_{-1}])$ for some ground action term $\bar{a}$), such that

$$\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_{i-1}]\} \models \mathbf{Know}(UTrans(dp_{i-1}, \mathsf{now}, \bar{dp'}, \bar{s'}), end[\sigma_{i-1}])$$

Then, $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_{i-1}]\} \models UTrans(dp_{i-1}, end[\sigma_{i-1}], \bar{dp'}, \bar{s'})$ follows by the reflexivity of K, which contradicts (1). Thus, the third case is also

not applicable, EX_1 cannot be different from EX_2, and there can only exist one complete execution of dp at σ w.r.t. $\bar{M}$.

The next theorem shows that for epistemically feasible deterministic programs, if it is entailed that the program can reach a final situation, then the program can be successfully executed online whatever the sensing outcomes may be.

THEOREM 7. *Let dp be such that $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$ entails $EFDP(dp, end[\sigma])$. Then, $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \exists s_f.Do(dp, end[\sigma], s_f)$ if and only if for each model M of $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$, the (only) complete online execution of dp from σ w.r.t. M is successfully terminating.*

Proof. $\Rightarrow$ Because of Theorem 6 we know that in any model M, the (only) complete online execution is either non-terminating or successful. We now prove by contradiction that it cannot be non-terminating.

Suppose that, for some model $\bar{M}$ of $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$, there is a *non-terminating* online execution $(dp_0 = dp, \sigma_0 = \sigma), ..., (dp_n, \sigma_n), ...$ Now, for any $i \geq 0$: $\bar{M} \models Trans(dp_i, end[\sigma_i], dp_{i+1}, end[\sigma_{i+1}])$. By Theorem 2, the agent knows in $\bar{M}$ of this transition for (dp_i, σ_i) at σ_i. Moreover given that dpt_i is a *LEFDP* at σ_i in $\bar{M}$ (that is, $\bar{M} \models LEFDP(dp_i, end[\sigma_i])$) the agent must know that such transition is the only possible one and that the configuration is *not* a final one in $\bar{M}$. From there, using reflexivity of K we conclude that

$$\bar{M} \models UTrans(dp_i, end[\sigma_i], dp_{i+1}, end[\sigma_{i+1}]) \wedge \neg Final(dp_i, end[\sigma_i])$$

In words, given that there is a transition from configuration $(dp_i, end[\sigma_i])$ to configuration $(dp_{i+1}, end[\sigma_{i+1}])$ for every $i \geq 0$ plus the fact that each configuration is a *LEFDP* one, then that transition is the *only* one possible in $\bar{M}$ and $(dp_i, end[\sigma_i])$ is not a final in $\bar{M}$. Since this holds for any arbitrary $i \geq 0$,

$$\bar{M} \models \forall dp', s'.Trans^*(dp, end[\sigma], dp', s') \supset \neg Final(dp', s')$$

It follows then that $\bar{M} \models \neg\exists s_f.Do(dp, end[\sigma], s_f)$, which contradicts the initial statement. Thus, the *non-terminating* complete online execution does not exists.

$\Leftarrow$ We proceed again by contradiction. Suppose there is model $\bar{M}$ of $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$ such that $\bar{M} \models \neg\exists s'.Do(dp, end[\sigma], s')$. By Theorem 6, there is only one complete online execution of dp at σ w.r.t. model $\bar{M}$; and, by assumption, such execution is successfully terminating. Then, by Lemma 1, $\bar{M}$ satisfies each step of the online execution, including the final terminating step. Therefore, $\bar{M} \models \exists dp', s'.Trans^*(dp, end[\sigma], dp', s') \wedge Final(dp', s')$, i.e., $\bar{M} \models \exists s'.Do(dp, end[\sigma], s')$ thus getting a contradiction.

4.2. SEMANTICS OF DELIBERATION STEPS

We now give the formal semantics of the deliberation steps. To denote these steps in the program we introduce a *deliberation operator* Δ_e, a new form of the IndiGolog search operator discussed in Section 2.

We define the *Trans* and *Final* predicates for the new deliberation operator as follows:

$$Trans(\Delta_e(p),s,dp',s') \equiv$$
$$\exists dp.EFDP(dp,s) \wedge \exists s_f.Trans(dp,s,dp',s') \wedge Do(dp',s',s_f) \wedge Do(p,s,s_f)$$
$$Final(\Delta_e(p),s) \equiv Final(p,s)$$

Thus, the axioms entail that there is a transition for $\Delta_e(p)$ from a situation s if and only if they entail that there is some epistemically feasible deterministic program dp that reaches a *Final* situation of the original program p no matter how sensing turns out (i.e., in every model of the axioms). Note also that the remaining program after the transition, dp', is what is left of dp; thus, the agent commits to the strategy/*EFDP* found in the initial deliberation and executes it.[10] Note that we do not need to put dp' inside a Δ_e block, since it is deterministic.

The following theorem shows that our semantics for the deliberation operator satisfies some basic requirements: if there is a transition for a deliberation block in a history σ, then (1) the program in the deliberation block can reach a *Final* situation in every model, and (2) so can $\Delta_e(p)$, and moreover (3) $\Delta_e(p)$ can be successfully executed online whatever the sensing results are (thus, the agent will never get to a configuration where it can no longer reach a *Final* situation or does not know what to do next):

THEOREM 8. *If* $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models Trans(\Delta_e(p), end[\sigma], p', s')$, *then*

1. $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \exists s_f.Do(p, end[\sigma], s_f)$
2. $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \exists s_f.Do(\Delta_e(p), end[\sigma], s_f)$
3. *For each model M of* $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$ *all online executions from* $(\Delta_e(p), \sigma)$ *w.r.t. M successfully terminate.*

Proof. 1. and 2. follow immediately from the definition of *Trans* for Δ_e. For 3. consider that by the definition of *Trans* for Δ_e, there exists a dp such that $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$ entails both $EFDP(dp, end[\sigma])$ and $\exists s_f, p', s'.Trans(dp, end[\sigma], p', s') \wedge Do(p', s', s_f)$. The conditions of Theorem 7 are satisfied, and thus we have that all online executions from (dp, σ)

[10] We discuss how this commitment to a given "strategy" can be relaxed when we address execution monitoring in Section 7.

are successfully terminating. Since these include all online executions from (p',σ') with $end[\sigma'] = s'$, all online executions from (p',σ') must also be successfully terminating. Hence the thesis follows.

5. Syntax-Based Accounts of *EFDP*s

In general, deliberating to find a way to execute a high-level program can be very hard because it amounts to doing planning where the class of potential plans is very general. It is thus natural to consider restricted classes of programs. Two particularly interesting such classes are: (i) programs that do not perform sensing, which correspond to conformant plans[11] (see e.g., [31]), and (ii) programs that are guaranteed to terminate in a bounded number of steps (i.e., do not involve any form of cycles), which correspond to conditional plans (see e.g., [30, 1]). We will show that for these two classes, one can restrict one's attention to simple syntactically-defined classes of programs without loss of generality. So if one is designing a deliberator/planner, one might want to only consider programs from these classes.

5.1. TREE PROGRAMS

Let us define the class of *(sense-branch) tree programs TREE* with the following BNF rule:

$$dpt ::= nil \mid False? \mid a; dpt_1 \mid True?; dpt_1 \mid sense_\phi; \textbf{if } \phi \textbf{ then } dpt_1 \textbf{ else } dpt_2$$

where a is any non-sensing action, and dpt_1 and dpt_2 are tree programs.
This class includes conditional programs where one can only test a condition that has just been sensed (trivial tests False? and True? are introduced for technical reasons). As one may expect, whenever such a program is executable, it is also epistemically feasible — the agent always knows what to do next. This is formalized in the next theorem.

THEOREM 9. *Let dpt be a tree program, i.e., $dpt \in TREE$. Then, for all histories σ, if $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \exists s_f.Do(dpt, end[\sigma], s_f)$ then program dpt is an EFDP at history σ, that is, $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models EFDP(dpt, end[\sigma])$.*

Proof. By induction on the structure of dpt.

Base cases. For *nil*, it is known that *nil* is $Final$, so that $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models EFDP(nil, end[\sigma])$ holds; for *False*?, the antecedent is false, so the thesis holds.

[11] We remind the reader that conformant plans are sequences of actions that, even under incomplete information about the domain, are guaranteed to reach the desired goal.

Inductive cases. Assume that the thesis holds for dpt_1 and dpt_2. Assume that $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \exists s_f.Do(dpt, end[\sigma], s_f)$.

For $dpt = a; dpt_1$: $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \exists s_f.Do(a; dpt_1, end[\sigma], s_f)$ implies that $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \exists s_f.Do(dpt_1, do(a, end[\sigma]), s_f)$. Since a is a non-sensing action, $\mathcal{D}_{sf} \models Sensed[\sigma \cdot (a,1)] \equiv Sensed[\sigma]$, so we also have that $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup Sensed[\sigma \cdot (a,1)]$ entails $\exists s_f.Do(dpt_1, end[\sigma \cdot (a,1)], s_f)$. Thus, by the induction hypothesis, we have $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma \cdot (a,1)]\} \models EFDP(dpt_1, end[\sigma \cdot (a,1)])$. It follows then that the set $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$ entails $EFDP(dpt_1, do(a, end[\sigma]))$. The initial assumption that $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$ entails $\exists s_f.Do(a; dpt_1, end[\sigma], s_f)$ also implies that $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models Poss(a, end[\sigma])$ and this must be known by Theorem 2, i.e., $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \mathbf{Know}(Poss(a, \mathsf{now}), end[\sigma])$. Thus, we have that

$$\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \mathbf{Know}(Trans(a; dpt_1, \mathsf{now}, dpt_1, do(a, \mathsf{now})), end[\sigma])$$

It is also known that this is the only transition possible for $a; dpt_1$ and that this program is not final. So $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$ logically entails $LEFDP(a; dpt_1, end[\sigma])$. Then,

$$\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models EFDP(a; dpt_1, end[\sigma])$$

For $dpt = True?; dpt_1$: the argument is similar, but simpler since the test does not change the situation.

For $dpt = sense_\phi$; **if** ϕ **then** dpt_1 **else** dpt_2: Suppose that the sensing action returns 1 and let $\sigma_1 = \sigma \cdot (sense_\phi, 1)$. Given that, the initial assumption that $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$ entails $\exists s_f.Do(dpt, end[\sigma], s_f)$ implies that $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_1]\} \models \exists s_f.Do(dpt_1, end[\sigma_1], s_f)$. Thus, by the induction hypothesis, $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma_1]\} \models EFDP(dpt_1, end[\sigma_1])$ holds. It follows next that

$$\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \phi(do(sense_\phi, end[\sigma])) \supset EFDP(dpt_1, do(sense_\phi, end[\sigma]))$$

By a similar argument, it also follows that we must have that

$$\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \neg\phi(do(sense_\phi, end[\sigma])) \supset EFDP(dpt_2, do(sense_\phi, end[\sigma]))$$

The initial assumption $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \exists s_f.Do(dpt, end[\sigma], s_f)$ also implies that $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models Poss(sense_\phi, end[\sigma])$ and this must be known by Theorem 2, i.e., $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$ logically entails $\mathbf{Know}(Poss(sense_\phi, \mathsf{now}), end[\sigma])$. Thus, we have that

$$\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \mathbf{Know}(Trans(dpt, \mathsf{now}, \textbf{if } \phi \textbf{ then } dpt_1 \textbf{ else } dpt_2, do(sense_\phi, \mathsf{now})), end[\sigma])$$

It is also known that this is the only transition possible for dpt and that dpt is not final, which implies $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models LEFDP(dpt, end[\sigma])$. Thus, $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models EFDP(dpt, end[\sigma])$.

Observe that as a consequence of the theorem above and Theorem 7, the online execution of dpt in σ is successfully terminating for all possible sensing outcomes. It follows that the problem of finding a tree program that yields an execution of a program in a deliberation block is the analogue in our framework of conditional planning (under incomplete information) in the standard setting [21, 30].

Next we show a quite strong result: tree programs are sufficient to express any strategy where there is a known bound on the number of steps it needs to terminate. That is, for any epistemically feasible deterministic program for which this condition holds, there is a tree program that produces the same executions:

THEOREM 10. *For any program dp that is*

1. *an epistemically feasible deterministic program, i.e., $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models EFDP(dp, end[\sigma])$ and*

2. *such that there is a known bound on the number of steps it needs to terminate, i.e., where there is an n such that*

$$\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \\ \exists p', s', k.k \leq n \wedge Trans^k(dp, end[\sigma], p', s') \wedge Final(p', s')$$

there exists a tree program $dpt \in TREE$ such that for each model M of the set $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$, the complete online execution of dp from σ with respect to M and the complete online execution of dpt from σ with respect to M successfully terminate in the same final history σ_M.

Proof. We construct the tree program $dpt = m(dp, \sigma)$ from dp using the following rules:

- $m(dp, \sigma) = False?$ if $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$ is inconsistent, otherwise

- $m(dp, \sigma) = nil$ if $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models Final(dp, end[\sigma])$, otherwise

- $m(dp, \sigma) = a; m(dp', \sigma \cdot (a, 1))$ iff

  $$\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models Trans(dp, end[\sigma], dp', do(a, end[\sigma])$$

  for some non-sensing action a,

– $m(dp,\sigma) = sense_\phi$; **if** ϕ **then** $m(dp',\sigma\cdot(sense_\phi,1))$
else $m(dp',\sigma\cdot(sense_\phi,0))$ iff

$$\mathcal{D}\cup\mathcal{T}\cup\mathcal{E}\cup\{Sensed[\sigma]\}\models Trans(dp,end[\sigma],dp',do(sense_\phi,end[\sigma])$$

for some sensing action $sense_\phi$,

– $m(dp,\sigma) = True?; m(dp',\sigma)$ iff

$$\mathcal{D}\cup\mathcal{T}\cup\mathcal{E}\cup\{Sensed[\sigma]\}\models Trans(dp,end[\sigma],dp',end[\sigma])$$

It turns out that, under the hypothesis of the theorem, for all dp and all σ, (dp,σ) is bisimilar to $(m(dp,\sigma),\sigma)$ with respect to online executions. Indeed, it is easy to check that the relation $[(dp,\sigma),(m(dp,\sigma),\sigma)]$ is a bisimulation, i.e., for all dp and σ, $[(dp,\sigma),(m(dp,\sigma),\sigma)]$ implies that

– $\mathcal{D}\cup\mathcal{T}\cup\mathcal{E}\cup\{Sensed[\sigma]\}\models Final(dp,end[\sigma])$ iff
$\mathcal{D}\cup\mathcal{T}\cup\mathcal{E}\cup\{Sensed[\sigma]\}\models Final(m(dp,\sigma),end[\sigma])$,

– for all dp', σ' if
$\mathcal{D}\cup\mathcal{T}\cup\mathcal{E}\cup\{Sensed[\sigma]\}\models Trans(dp,end[\sigma],dp',end[\sigma'])$ with $\mathcal{D}\cup\mathcal{T}\cup\mathcal{E}\cup\{Sensed[\sigma']\}$ consistent, then

$\mathcal{D}\cup\mathcal{T}\cup\mathcal{E}\cup\{Sensed[\sigma]\}\models Trans(m(dp,\sigma),end[\sigma],m(dp',\sigma'),end[\sigma'])$
and $[(dp',\sigma'),(m(dp',\sigma'),\sigma')]$,

– for all dp', σ' if
$\mathcal{D}\cup\mathcal{T}\cup\mathcal{E}\cup\{Sensed[\sigma]\}\models Trans(m(dp,\sigma),end[\sigma],m(dp',\sigma'),end[\sigma'])$
with $\mathcal{D}\cup\mathcal{T}\cup\mathcal{E}\cup\{Sensed[\sigma']\}$ consistent, then

$$\mathcal{D}\cup\mathcal{T}\cup\mathcal{E}\cup\{Sensed[\sigma]\}\models Trans(dp,end[\sigma],dp',end[\sigma'])$$

and $[(dp',\sigma'),(m(dp',\sigma'),\sigma')]$.

(By the way, note that in this definition, we do not require that histories have sensing values that come from a fixed model of $\mathcal{D}\cup\mathcal{T}\cup\mathcal{E}\cup\{Sensed[\sigma]\}$, only that they remain consistent with $\mathcal{D}\cup\mathcal{T}\cup\mathcal{E}$ and the sensing values already encountered. In fact, bisimulation may hold even w.r.t. sensing outcomes that are not possible w.r.t. $\mathcal{D}\cup\mathcal{T}\cup\mathcal{E}\cup\{Sensed[\sigma]\}$. However, for programs that always terminate in a finite number of steps as assumed, the histories considered will always be such that there is a model of $\mathcal{D}\cup\mathcal{T}\cup\mathcal{E}\cup\{Sensed[\sigma]\}$ that generates them.)

Since by hypothesis $\mathcal{D}\cup\mathcal{T}\cup\mathcal{E}\cup\{Sensed[\sigma]\models\exists s_f.Do(dp,end[\sigma],s_f)$ (in a bounded number of steps, in fact), considering that dp is an *EFDP*, by Theorem 7 for all models M of $\mathcal{D}\cup\mathcal{T}\cup\mathcal{E}\cup\{Sensed[\sigma]\}$ the (unique) online execution of dp from σ w.r.t. M successfully terminates. Hence since (dp,σ)

and $(m(dp,\sigma),\sigma)$ are bisimilar, $m(dp,\sigma)$ has the same online execution from σ w.r.t. M (apart from the program appearing in the configurations) and the two online executions successfully terminate in the same final history σ_M.

This theorem shows that if we restrict our attention to *EFDP*s that terminate in a bounded number of steps, then we can further restrict our attention to programs of a very specific syntactic form, without any loss in generality. This may simplify the task of coming up with a successful strategy for a given deliberation block.

5.2. LINEAR PROGRAMS

Let the class of linear programs *LINE* be defined by the following BNF rule:

$$dpl ::= nil \mid a; dpl_1 \mid True?; dpl_1$$

where a is any non-sensing action, and dpl_1 is a linear program.
This class only includes sequences of actions or trivial tests. So whenever such a plan is executable, then it is also epistemically feasible — the agent always knows what to do next:

THEOREM 11. *Let dpl be a linear program, i.e., $dpl \in LINE$. Then, for all histories σ, if $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \exists s_f.Do(dpl, end[\sigma], s_f)$ then dpl is an EFDP at history σ, i.e., $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models EFDP(dpl, end[\sigma])$.*

Proof. This is a corollary of Theorem 9 for tree programs. Since linear programs are tree programs, the thesis follows immediately from this theorem.

Again as a consequence of the theorem above and Theorem 7, the online execution of (dpl,σ) is successfully terminating for all possible sensing outcomes. Observe that the problem of finding a linear program that yields an execution of a program in a deliberation block is the analogue in our framework of conformant planning in the standard setting [31].

Next, we show that linear programs are sufficient to express any strategy that does not perform sensing.

THEOREM 12. *For any dp that does not include sensing actions, such that $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models EFDP(dp, end[\sigma])$, there exists a linear program dpl such that for each model M of $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$, the complete online execution of dp from σ w.r.t. M and the complete online execution of dpl from σ w.r.t. M successfully terminate in the same history σ_m.*

Proof. We show this using the same approach as for Theorem 10 for tree programs. Since dp cannot contain sensing actions, the construction method used in the proof of Theorem 10 produces a tree program that contains no

branching and is in fact a linear program. Then, by the same argument as used there, the thesis follows.

Observe that this implies that if no sensing is possible — for instance, because there are no sensing actions — then linear programs are sufficient to express every strategy.

Let Δ_l be a deliberation operator that is axiomatized just as Δ_e except that we replace the requirement that dp be an epistemically feasible deterministic program by the requirement that it be a linear program, i.e., where we use the axiom (the *LINE* predicate is defined in the obvious way):

$$Trans(\Delta_l(p),s,dpl',s') \equiv$$
$$\exists dpl.LINE(dpl) \wedge \exists s_f.Trans(dpl,s,dpl',s') \wedge Do(dpl',s',s_f) \wedge Do(p,s,s_f)$$

Then, one can show that a program using this deliberation operator $\Delta_l(p)$ can make a transition in a history if and only if one can identify a sequence of actions that is an execution of p in all models for the history:

THEOREM 13. *There exists a situation s_f such that*

$$\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models Do(p,end[\sigma],s_f)$$

if and only if there is a $dpl \in LINE$ and an s' such that

$$\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models Trans(\Delta_l(p),end[\sigma],dpl,s')$$

Proof. $\Leftarrow$ By hypothesis there exists a dpl that is a *LINE*. If $s' = end[\sigma]$ then $dpl = true?;dpl'$ and if $s' = do(a,end[\sigma])$, for some action a, and then $dpl = a;dpl'$. In both cases dpl' must be a *LINE*. In every model dpl' reaches from s' a final situation of the original program p. Observe that such a situation will be the same in every model since the sequence of actions starting from s' is fixed by dpl'. It follows that the sequence of action done by dpl starting from s reaches a situation s_f such that $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models Do(p,end[\sigma],s_f)$.

$\Rightarrow$ If for some s_f we have $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models Do(p,end[\sigma],s_f)$ then the sequence of actions from $end[\sigma]$ to s_f is a *LINE* program, which trivially satisfies the left-hand-side of the axiom for Δ_l. Observe that if $s_f = end[\sigma]$ then we can simply use the linear program $True?;nil$ to satisfy the left-hand side of the axiom for Δ_l.

This provides the basis for a simple implementation.

6. Implementation

Let us now examine how the deliberation construct can be implemented according to the specification given above, i.e., by having the interpreter look for an epistemically feasible deterministic program of a certain type, linear, tree, etc. We also relate these implementations to earlier implementation proposals for IndiGolog.

Instead of situations, our implementations use histories, which are lists of pairs of actions and sensing outcomes since the initial situation. We assume the following code is already available: (a) `holds(P,H)` implements the evaluation procedure and states that formula `P` is true at history `H`; (b) `trans/4` and `final/2` implement relations *Trans* and *Final* respectively; and (c) `senses(A,F)` states that action `A` senses the truth value of fluent `F`.

The simplest type of implementation is one that only considers linear programs as potential strategies for executing the program in the deliberation block, as in the specification of Δ_l above. This will work if there is a solution that does not do sensing. Here is the code in Prolog:

```
/* implementation using linear programs  */
trans(delib_l(P),H,DPL1,H1)  :-
  buildLine(P,DPL,H), trans(DPL,H,DPL1,H1).
buildLine(P,[],H) :- final(P,H).
buildLine(P,[(true)?|DPL],H) :-
  trans(P,H,P1,H), buildLine(P1,DPL,H).
buildLine(P,[A|DPL],H) :-
  trans(P,H,P1,[(A,1)|H]),
  not senses(A,_),           /* A is not a sensing action */
  buildLine(P1,DPL,[(A,1)|H]).
```

The `buildLine(P,DPL,H)` predicate basically looks for a sequence of transitions that the given program `P` can perform in history `H` which is guaranteed to lead to a final configuration; the transitions must not involve sensing actions, which would be useless without branching (sensing outcomes for non-sensing actions are assumed to be 1); the sequence of transitions found is returned as a linear program `DPL`. This approach to implementing deliberation is essentially that used in [8, 14, 7], as these assume that deliberation blocks do not contain sensing actions.

A more general type of implementation is one that considers tree programs as potential strategies for executing the program in the deliberation block, assuming that binary sensing actions are available. This can be implemented by generalizing the above as follows:

```
/* implementation using tree programs    */
trans(delib_t(P),H,DPT1,H1) :-
```

```
  buildTree(P,DPT,H), trans(DPT,H,DPT1,H1).
buildTree(P,[],H) :- final(P,H).
buildTree(P,[(true)?|DPT],H) :-
  trans(P,H,P1,H), buildTree(P1,DPT,H).
buildTree(P,[A,if(F,DPT1,DPT2)]) :-
  trans(P,H,P1,[(A,_)|H]), senses(A,F),
  buildTree(P1,DPT1,[(A,1)|H]),
  buildTree(P1,DPT2,[(A,0)|H]).
buildTree(P,[A|DPT],H) :-
  trans(P,H,P1,[(A,_)|H]), not senses(A,_),
  buildTree(P1,DPT,[(A,1)|H]).
buildTree(P,(false)?,H) :- inconsistent(H).

inconsistent([(A,1)|H]) :-
  senses(A,F), holds(neg(F),H) ; inconsistent(H).
inconsistent([(A,0)|H]) :-
  senses(A,F), holds(F,H) ; inconsistent(H).
```

A transition is performed on a program `delib_t(P)` only if it is always possible to extend it into a complete execution of `P`. To ensure this, whenever a binary sensing action is encountered, the code verifies the existence of complete executions for both potential sensing outcomes `0` and `1` (3rd clause of `buildTree`). For non-sensing actions, the sensing outcome is assumed to be `1`, and the existence of an execution is verified in this single case (4th clause of `buildTree`). This implementation is similar to that of [5]. Both of the above implementations are sound (see [4, 7] on techniques to prove this), but not complete even assuming soundness and completeness of `holds/2`. The incompleteness comes from the fact that they stick to the form of the original program while the semantics does not. One example that brings this out is: $\phi?;\psi?;a \mid \neg\phi?;\neg\psi?;a$, where it is known that $\phi \equiv \psi$. For our semantics, the LINE program $\mathit{True}?;\mathit{True}?;a$ is a strategy for executing it, but the implementations fail to find it.

7. Deliberation with Execution Monitoring

So far, we have provided a formal account of plans that are suitable for an agent capable of sensing the environment during the execution of a high-level program. We have not addressed, though, another important feature of complex environments with which a realistic agent needs to cope as well: *exogenous actions*. Intuitively, an exogenous action is an action outside the control of the agent, perhaps a natural event or an action performed by another agent. Technically, these are primitive actions that may occur without being

part of the user-specified program. It is not hard to imagine how one would slightly alter the definition of *online execution* of Section 2 so as to allow for the occurrence of exogenous actions after each legal transition. Nonetheless, an exogenous action can potentially compromise the online execution of a deliberation block. This is due to the fact that Δ_e commits to a particular *EFDP* which can turn out to be impossible to execute after the occurrence of some interfering outside action. If there is another *EFDP* that could be used instead to complete the execution of the deliberation block, we would like the agent to switch to it.

To address this problem, the search operator defined in [14] implements an execution monitoring mechanism. The idea is to recompute a search block whenever the current plan has become invalid due to the occurrence of exogenous actions during the incremental execution. The new search starts from the original program and situation (this is important because often commitments are made early on in the program's execution, and these may have to be revised when an exogenous change occurs) and ensures that the plan produced is compatible with the already performed actions.

Based on [8, 14], one can come up with a clean and abstract formalization of execution monitoring and replanning for our epistemic version of deliberation described in Section 4.2. The idea is to avoid permanently committing to a particular *EFDP*. Instead, we define a deliberation operator Δ_{em} that monitors the execution of the selected *EFDP* and replans when necessary, possibly selecting an alternative *EFDP* to follow. The semantics of this *monitored deliberation* construct goes as follows:

$$\begin{aligned}
&Trans(\Delta_{em}(p), s, p', s') \equiv \\
&\quad \exists dp, dp'.EFDP(dp, s) \wedge p' = mnt(dp', s', p, s) \wedge \\
&\qquad \exists s_f.Trans(dp, s, dp', s') \wedge Do(dp', s', s_f) \wedge Do(p, s, s_f) \\
&Final(\Delta_{em}(p), s) \equiv Final(p, s)
\end{aligned}$$

The main difference is in the remaining program which contains not only the epistemically feasible strategy chosen, but also the original program p, original situation s, and next expected situation s'. These components are packaged using a new language construct *mnt*, which basically means that the agent should *monitor* the execution of the selected strategy dp using the original program and situation to replan when necessary.

The next step, then, is to define the semantics for the new "monitoring" construct *mnt*. With that objective, we first introduce two auxiliary relations. Relation $perturbed(mnt(dp, s_e, p_i, s_i), s)$ states whether the strategy dp has just been perturbed in situation s by some exogenous action; p_i and s_i represent the initial program and initial situation from where dp comes from, whereas s_e represent the situation in which program dp is expected to execute in. There are obviously several ways to define when a strategy has been perturbed. A sensible one is the following: a strategy has been perturbed if

the exogenous actions that just occurred rule out a successful execution for both the strategy and the original program of the deliberation block.

$$perturbed(mnt(dp, s_e, p_i, s_i), s) \equiv$$
$$s_e \neq s \wedge \neg\exists s_f.[Do(dp, s, s_f) \wedge Do(p_i \parallel p_{ex}, s_i, s_f)]$$

Above we make use of the special program $p_{ex} \stackrel{\text{def}}{=} (\pi a.Exo(a)?; a)^*$ to allow for a legal sequence of exogenous actions (see [4]). Also, observe that a strategy can be perturbed *only* if an action outside the strategy occurred, in which case the actual situation s would differ from the expected situation s_e. Thus in practice, there is no need to check for perturbation unless an exogenous action or an action other than that performed by the chosen strategy occurs.

The next auxiliary relation is used to calculate a *recovered* strategy dp_r when the current one dp was perturbed in situation s. A sensible definition for it is:

$$recover(mnt(dp, s_e, p_i, s_i), s, dp_r) \equiv$$
$$\exists p_i'.Trans^*(p_i \parallel p_{ex}, s_i, p_i' \parallel p_{ex}, s) \wedge$$
$$EFDP(dp_r, s) \wedge \exists s_f.Do(dp_r, s, s_f) \wedge Do(p_i', s, s_f)$$

Observe that the above definition may end up choosing an *alternative* epistemically feasible strategy than the one chosen before. In a nutshell, a new *recovered* strategy is an epistemically feasible one that is able to "solve" the original program p_i while accounting for *every* action executed so far, either by the deliberation block or exogenous, since the beginning of the deliberation block.

We now have all the machinery needed to define the semantics for the monitoring construct *mnt*:

$$Trans(mnt(dp, s_e, p_i, s_i), s, p', s') \equiv$$
$$[\neg perturbed(mnt(dp, s_e, p_i, s_i), s) \wedge$$
$$\exists dp'.Trans(dp, s, dp', s') \wedge p' = mnt(dp', s', p_i, s_i)] \vee$$
$$[perturbed(mnt(dp, s_e, p_i, s_i), s) \wedge$$
$$\exists dp_r.recover(mnt(dp, s_e, p_i, s_i), s, dp_r) \wedge$$
$$\exists dp'.Trans(dp_r, s, dp', s') \wedge p' = mnt(dp', s', p_i, s_i)]$$

$$Final(mnt(dp, s_e, p_i, s_i), s) \equiv$$
$$[\neg perturbed(mnt(dp, s_e, p_i, s_i), s) \wedge Final(dp, s)] \vee$$
$$[perturbed(mnt(dp, s_e, p_i, s_i), s) \wedge Do(p_i \parallel p_{ex}, s_i, s)]$$

For *Trans*, we have two possibilities: (i) if the strategy has not been perturbed, then we continue its execution by performing one step and updating the next expected situation; (ii) if the strategy has just been perturbed, a recovered new strategy dp_r is computed and the execution continues with respect to this alternative strategy. It is important to note that the original program and

situation are always kept throughout the whole execution of a deliberation block. In that way, the recovery process can be as general as possible. The case for *Final* is simpler: (i) if the strategy has not been perturbed, then we check whether the strategy is final in the actual situation; (ii) if the strategy has been perturbed, then there is a chance that the original program might be terminating in the current situation and we check for this.

In summary, deliberation can be naturally integrated with execution monitoring in order to cope with exogenous actions that make the chosen strategy unsuitable.

8. Conclusion

In this paper, we developed an account of the kind of deliberation that an agent that is doing planning or executing high-level programs under incomplete information must be able to perform. The deliberator's job is to produce a kind of plan that does not itself require deliberation to interpret. We characterized these as *epistemically feasible* programs: programs for which the executing agent, at every stage of execution, by virtue of what it knew initially and the subsequent readings of its sensors, always *knows* what step to take next towards the goal of completing the entire program. We formalized this notion and characterized deliberation in the IndiGolog agent programming language in terms of it. We have also shown that for certain classes of problems, which correspond to conformant planning and conditional planning, the search for epistemically feasible programs can be limited to programs of a simple syntactic form.

There has been a lot of work in the past on formalizing notions of epistemically feasible plan and achievability of goals, e.g. [20, 2, 13, 15], and our accounts builds on this. However, our account differs from previous work on several aspects. First, we model the plans that are the result of deliberation as ordinary programs that satisfy certain semantic criteria, i.e. are epistemically feasible deterministic programs. This means that after deliberation, such plans can be handled by the existing on-line executor for the language. They do not belong to a different "plan language" as in [15], and are not syntactically restricted as in most work on planning with incomplete information. Our proposal differs from that in [2], in which there is no characterization of the result of deliberation other than as a semantic object (a relation over situations), and this also applies to [13] and most other accounts of goal achievability. Secondly, we have shown how deliberation can be viewed as a part of agent program execution, and our semantics for deliberation is integrated within the transition system semantics of our programming language. Thirdly, we show how one can also incorporate execution monitoring and replanning to cope with a changing environment. Many agent applications re-

quire planning, and often involve incomplete information and sensing. In this work, we try to show how one can develop an agent programming language, IndiGolog, that is a convenient tool for this.

As far as we know, the only other agent programming language that attempts to support planning under incomplete information is FLUX [32, 33]. Thielscher's FLUX agent programming framework supports online execution, sensing, and planning for agents with open world knowledge bases with disjunctive formulas, with the restriction that only finitely many facts are known to be true. It is implemented using constraint logic programming techniques which, together with the Fluent Calculus state-based approach, yields good computational properties in terms of execution time. FLUX though, only does a restricted form of conditional planning and no results are proven regarding the correctness of the outcome of its deliberation mechanism. Also, the programming framework is defined somewhat informally and it is not clear exactly what range of planning problems can be handled.

McIlraith and Son [19] have used Golog to model web services and perform service customization and composition. They also formalize a notion of "self-sufficient program" that is similar to that of an *EFDP*s; however their account is incomplete for programs that involve indefinite iteration (such as the tree chopping example of [13]) and more sensitive to the program's syntax than ours. It would be interesting to evaluate the effectiveness of IndiGolog's planning capabilities in such applications.

Many problems and lines of research remain open. In this paper, we have only dealt with binary sensing actions. However, the account of deliberation developed in Section 4 and its extension to provide execution monitoring in Section 7 do not rely on this restriction and apply unchanged to theories with sensing actions that have even an infinite number of possible sensing outcomes.[12] This comes from the fact that our characterization of "good execution strategies" through the notion of *EFDP* is not syntactic, only requiring the agent to know what action to do next at every step. The results of Section 5.1 showing that tree programs are sufficient to solve any planning/deliberation problem where there is some strategy that solves the problem in a bounded number of steps also generalize to domains involving sensing actions with non-binary but finitely many outcomes; this is easy to see given that any such sensing action can be encoded as a sequence binary sensing actions that read the outcome one bit at a time (one could of course extend the class of tree programs with a non-binary branching structure to avoid the need for such an encoding). Whether a similar characterization can be obtained for sensing actions with an infinite number of possible outcomes is an open problem. While the above holds in principle, as soon as the number of sensing outcomes is more than a few, conditional planning becomes

[12] One can introduce non-binary sensing actions in our framework as in [29].

impractical without advice from the programmer as to what conditions the plan should branch on [11, 32]. In [27], a search construct for IndiGolog that generates conditional plans involving non-binary sensing actions by relying on such programmer advice is developed. This approach seems very compatible with ours and it would be interesting to formalize it as a special case of our account of deliberation. In [28], the search operator is combined with declarative goals to provide a planning account which mixes both procedural and declarative notions of action. Roughly speaking, the new (rational) search operator looks for the "best" *EFDP* possible w.r.t. some set of (prioritized) goals. There are also more general theories of sensing, such as that of [6] which deals with online sensors that always provide values and situations where the law of inertia is not always applicable. In [7], a search operator for such theories is developed. It would be worthwhile examining whether this setting could also be handled within our account of deliberation. As well, one could look for syntactic characterizations for certain classes of epistemically feasible deterministic programs in this setting.

Also related to the work presented here is [12], where a similar approach is used to develop an account of epistemic feasibility for multiagent system specifications, expressed in a version of ConGolog extended with knowledge and goal attitudes. In [3], we investigate a non-epistemic account of deliberation that is more easily related to previous work on agent programming languages and draw some lessons.

References

[1] Bertoli, P., A. Cimatti, M. Roveri, and P. Traverso: 2001, 'Planning in Nondeterministic Domains under Partial Observability via Symbolic Model Checking'. In: *Proc. of IJCAI'01*. Seattle, pp. 473–478.

[2] Davis, E.: 1994, 'Knowledge Preconditions for Plans'. *Journal of Logic and Computation* **4**(5), 721–766.

[3] De Giacomo, G., Y. Lespérance, H. Levesque, and S. Sardiña: 2003, 'On Deliberation under Incomplete Information and the Inadequacy of entailment and Consistency-Based Formalizations'. In: *Proceedings of the 1st Programming Multiagent Systems Languages, Frameworks, Techniques and Tools workshop (PROMAS-2003)*. Melbourne, Australia.

[4] De Giacomo, G., Y. Lespérance, and H. J. Levesque: 2000, 'ConGolog, a Concurrent Programming Language Based on the Situation Calculus'. *Artifi cial Intelligence* **121**, 109–169.

[5] De Giacomo, G. and H. J. Levesque: 1999a, 'An Incremental Interpreter for High-Level Programs with Sensing'. In: H. J. Levesque and F. Pirri (eds.): *Logical Foundations for Cognitive Agents*. Springer-Verlag, pp. 86–102.

[6] De Giacomo, G. and H. J. Levesque: 1999b, 'Progression and Regression Using Sensors'. In: *Proc. of IJCAI-99*. pp. 160–165.

[7] De Giacomo, G., H. J. Levesque, and S. Sardiña: 2001, 'Incremental execution of Guarded Theories'. *ACM Transactions on Computational Logic* **2**(4), 495–525.

[8] De Giacomo, G., R. Reiter, and M. Soutchanski: 1998, 'Execution Monitoring of High-Level Robot Programs'. In: *Proc. of KR-98*. pp. 453–465.
[9] Fisher, M.: 1995, 'Towards a Semantics for Concurrent METATEM'. In: M. Fisher and R. Owens (eds.): *Executable Modal and Temporal Logics (LNAI Volume 896)*.
[10] Hindriks, K. V., F. S. de Boer, W. van der Hoek, and J.-J. C. Meyer: 1998, 'A formal semantics for an abstract agent programming language'. In: M. Singh, A. Rao, and M. Wooldridridge (eds.): *Intelligent Agents IV — Proc. of ATAL'97*. pp. 215–229.
[11] Lakemeyer, G.: 1999, 'On Sensing and Off-Line Interpreting in Golog'. In: H. J. Levesque and F. Pirri (eds.): *Logical Foundations for Cognitive Agents*. Springer-Verlag, pp. 173–187.
[12] Lespérance, Y.: 2001, 'On the Epistemic Feasibility of Plans in Multiagent Systems Specifi cations'. In: J.-J. C. Meyer and M. Tambe (eds.): *Intelligent Agents VIII, Agent Theories, Architectures, and Languages, 8th Intl. Workshop, ATAL-2001, Seattle, WA, USA, Aug. 1-3, 2001, Revised Papers*, Vol. 2333 of *LNAI*. pp. 69–85.
[13] Lespérance, Y., H. J. Levesque, F. Lin, and R. B. Scherl: 2000, 'Ability and Knowing How in the Situation Calculus'. *Studia Logica* **66**(1), 165–186.
[14] Lespérance, Y. and H.-K. Ng: 2000, 'Integrating Planning into Reactive High-Level Robot Programs'. In: *Proc. of the Second International Cognitive Robotics Workshop*. pp. 49–54.
[15] Levesque, H. J.: 1996, 'What is Planning in the Presence of Sensing?'. In: *Proc. of AAAI-96*. pp. 1139–1146.
[16] Levesque, H. J. and G. Lakemeyer: 2001, *The Logic of Knowledge Bases*. MIT Press.
[17] Levesque, H. J., R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl: 1997, 'GOLOG: A Logic Programming Language for Dynamic Domains'. *Journal of Logic Programming* **31**(59–84).
[18] McCarthy, J. and P. Hayes: 1979, 'Some Philosophical Problems from the Standpoint of Artifi cial Intellig ence'. In: B. Meltzer and D. Michie (eds.): *Machine Intelligence*, Vol. 4. Edinburgh University Press, pp. 463–502.
[19] McIlraith, S. and T. C. Son: 2002, 'Adapting Golog for Programming the Semantic Web'. In: *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002)*. Toulouse, France, pp. 482–493.
[20] Moore, R. C.: 1985, 'A Formal Theory of Knowledge and Action'. In: J. R. Hobbs and R. C. Moore (eds.): *Formal Theories of the Common Sense World*. Norwood, NJ: Ablex Publishing, pp. 319–358.
[21] Peot, M. A. and D. E. Smith: 1992, 'Conditional Nonlinear Planning'. In: *Proc. of the First International Conference on AI Planning Systems*. pp. 189–197.
[22] Plotkin, G.: 1981, 'A structural approach to operational semantics'. Technical Report DAIMI-FN-19, Computer Science Dept., Aarhus University, Denmark.
[23] Rao, A.: 1996, 'AgentSpeak(L): BDI agents speak out in a logical computable language'. In: W. V. der Velde and J. W. Perram (eds.): *Agents Breaking Away*. LNAI 1038, Springer-Verlag, pp. 42–55.
[24] Reiter, R.: 1991, 'The Frame Problem in the Situation Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression'. In: V. Lifschitz (ed.): *Artifi cial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*. Academic Press, pp. 359–380.
[25] Reiter, R.: 2001a, *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.
[26] Reiter, R.: 2001b, 'On Knowledge-Based Programming with Sensing in the Situation Calculus'. *ACM Transactions on Computational Logic* **2**(4), 433–457.
[27] Sardiña, S.: 2001, 'Local Conditional High-Level Robot Programs'. In: *Proc. of LPAR-01*, Vol. 2250 of *LNAI*. pp. 110–124.

[28] Sardiña, S. and S. Shapiro: 2003, 'Rational Action in Agent Programs with Prioritized Goals'. In: *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-03)*. Melbourne, Australia, pp. 417–424.
[29] Scherl, R. B. and H. J. Levesque: 1993, 'The Frame Problem and Knowledge-Producing Actions'. In: *Proc. of AAAI-93*. pp. 689–695.
[30] Smith, D. E., C. R. Anderson, and D. S. Weld: 1998, 'Extending Graphplan to Handle Uncertainty and Sensing Actions'. In: *Proc. of AAAI-98*. pp. 897–904.
[31] Smith, D. E. and D. S. Weld: 1998, 'Conformant Graphplan'. In: *Proc. of AAAI-98*. pp. 889–896.
[32] Thielscher, M.: 2001, 'Inferring Implicit State Knowledge and Plans with Sensing Actions'. In: *Proc. of KI-01*, Vol. 2174 of *LNAI*. pp. 366–380.
[33] Thielscher, M.: 2002, 'Programming of Reasoning and Planning Agents with FLUX'. In: D. Fensel, F. Giunchiglia, D. McGuinness, and M. A. Williams (eds.): *Proceedings of Eighth International Conference in Principles of Knowledge Representation and Reasoning (KR-2002)*. Toulouse, France, pp. 435–336.

Appendix

A. Proofs

Recall from Section 2 that $\mathcal{D}$ denotes the set of axioms defining an underlying theory of action, $\mathcal{T}$ denotes the set of axioms for $Trans$ and $Final$, and $\mathcal{E}$ stands for the set of axioms needed for the encoding of programs as first-order terms (see [4].)

Also, we will be using two functions defined in [25] for performing regression. First, $\rho^1(\phi(\mathsf{now}), A)$ stands for the one-step regression of formula $\phi(\mathsf{now})$ through action A. Second, $\rho(\phi(\mathsf{now}), end[\sigma])$ stands for the full regression of formula $\phi(\mathsf{now})$ through situation $end[\sigma]$.

For our proofs, we will be using some generalized versions of existing results proven by Reiter [25] (Chapter 11). Roughly speaking, we will be adding the set of axioms $\mathcal{E}$ for the encoding of programs as first-order terms into these existing results, and the results in [25] should still be valid as adding $\mathcal{E}$ only produces a conservative extension that defines the new program sort.

We do not provide detailed proofs of these results since the proofs are long, laborious, and of limited interest. But let us point out that the following three results hold because of the following reasons: (i) program terms and variables can *only* be mentioned in objective formulas as arguments of equality terms; (ii) given that every object and every action has a name in the language, it follows (because of $\mathcal{E}$) that every possible program must have a name as well; (iii) $\mathcal{E} \cup \mathcal{D}_{una} \cup \{\psi(S_0)\}$ decides all equalities sentences given that $\mathcal{D}_{una} \cup \{\psi(S_0)\}$ decides all equalities sentences that mention no program term or variable; and (iv) it is possible to obtain a generalized version of the "Regression Theorem with Knowledge" (Theorem 11.6.3 in [25]) in which $\mathcal{E}$ is added to the set of underlying axioms and equality among program terms are permitted in regressable formulas. Point (iv) relies on the fact that any model of $\mathcal{D}$ can be extended to satisfy $\mathcal{E} \cup \mathcal{D}$ by Theorem A.1 in [4], and we only use its instance w.r.t. the initial situation S_0.

LEMMA 2 (Generalization of Lemma 11.7.2 in [25]). *If* $\phi(\mathsf{now})$ *is an objective formula, then*

$$\mathcal{E} \cup \mathcal{D} \models \mathbf{Know}(\phi(\mathsf{now}), S_0) \text{ iff } \mathcal{E} \cup \mathcal{K}_{Init} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \models \mathbf{Know}(\phi(\mathsf{now}), S_0)$$

LEMMA 3 (Generalization of Lemma 11.7.3 in [25]). *Suppose that* $\phi(\mathsf{now})$ *is an objective sentence, and that* $\mathcal{D}_{S_0} = \{\mathbf{Know}(\psi(\mathsf{now}), S_0)\}$ *where* $\psi(\mathsf{now})$ *is objective. Then,*

$$\mathcal{E} \cup \mathcal{D} \models \mathbf{Know}(\phi(\mathsf{now}), S_0) \text{ iff } \mathcal{E} \cup \mathcal{D}_{una} \cup \{\psi(S_0)\} \models \phi(S_0)$$

LEMMA 4 (Generalization of Lemma 11.7.12 in [25]). *Suppose that* $\phi_0(\mathsf{now}), ...,$ $\phi_n(\mathsf{now})$ *are objective sentences, that* $\mathcal{D}_{S_0} = \{\mathbf{Know}(\psi(\mathsf{now}), S_0)\}$ *where* $\psi(\mathsf{now})$ *is objective, and that* $\mathcal{E} \cup \mathcal{D}_{una} \cup \psi(S_0)$ *decides all equality sentences. Suppose further that*

$$\mathcal{E} \cup \mathcal{D} \models \phi_0(S_0) \vee \mathbf{Know}(\phi_1(\mathsf{now}), S_0) \vee ... \vee \mathbf{Know}(\phi_n(\mathsf{now}), S_0)$$

Then for some $0 \leq i \leq n$, $\mathcal{E} \cup \mathcal{D} \models \mathbf{Know}(\phi_i(\mathsf{now}), S_0)$

A.1. Additional Lemmas

LEMMA 5. *Let* $\psi(\mathsf{now})$ *be an objective formula. If* $\mathcal{E} \cup \mathcal{D}_{una} \cup \{\psi(S_0)\}$ *is satisfiable, then it is satisfiable in a model M such that for every object/action/program element d in the object/action/program universe of M, there is an object/action/program term t in the language such that* $[t]^M = d$.

Proof. For objects and actions, the Lemma follows directly from assumptions 9 (domain closure and unique names for objects) and 8 (finitely many action types) in Section 3. Objects are identified with a set of well-defined standard names; actions are build from (finitely many) action functions and the objects.

For programs, it follows directly from the fact that $\mathcal{E}$ has a second order axiom closing the set of programs to be exactly the one constructed from primitive actions and a finite set of language constructs (if, while, pick, etc.).

LEMMA 6 (Base case of Theorem 4). *Let* $\phi(\vec{x}, \mathsf{now})$ *be an objective formula with non-situation free variables* $\vec{x}$ *(that is, object, action, or program variables.)*
Then, $\mathcal{D} \cup \mathcal{E} \models \exists \vec{x}.\mathbf{Know}(\phi(\vec{x}, \mathsf{now}), S_0)$ *if and only if there are ground terms* $\vec{t}$ *such that* $\mathcal{D} \cup \mathcal{E} \models \mathbf{Know}(\phi(\vec{t}, \mathsf{now}), S_0)$

Proof. $\Leftarrow$) This direction is trivial.

$\Rightarrow$) Without loss of generality, we assume $\vec{x} = x$. Let $t_1^o, t_2^o, ...$ be an enumeration of object terms, let $t_1^a, t_2^a, ...$ be an enumeration of action terms, and $t_1^p, t_2^p, ...$ be an enumeration of program terms. In general, all three enumerations will be infinite as there are infinite terms that can be built from one constant and one function.

We can then simplify these enumerations by grouping terms that are seen equal w.r.t. the underlying theory. That is, we can assume $\bar{t}_1^o, \bar{t}_2^o, ...$ is an enumeration of *different* equivalences classes among ground *object terms* such that two object terms t_i^o and t_j^o are in the same equivalence class iff $\mathcal{D}_{una} \cup \psi(S_0) \models t_i = t_j$. Clearly, the whole set of possible object terms is perfectly partitioned because $\mathcal{D}_{una} \cup \psi(S_0)$ decides

equality over object sentences. An analogous argument will lead us to an enumeration $\overline{t}^a_1, \overline{t}^a_2, ...$ is an enumeration of *different* equivalences classes among ground *action terms*. Lastly, the "decides equality" property is automatically lifted to program terms whenever we take into consideration the set of axioms $\mathcal{E}$ defining how program terms are built, and, therefore, we can construct an enumeration $\overline{t}^p_1, \overline{t}^p_2, ...$ of *different* equivalences classes among ground *program terms*.

Assume next that for every $i \geq 1$, $\mathcal{D} \cup \mathcal{E} \not\models \mathbf{Know}(\phi(t_i, \mathsf{now}), S_0)$ where t_i is of the type of variable x (i.e., t_i stands for a term in the object, action, or program enumeration.) By Lemma 3, $\mathcal{E} \cup \mathcal{D}_{una} \cup \psi(S_0) \not\models \{\phi(t_i, S_0)\}$, for every $i \geq 1$ where $\mathcal{D}_{S_0} = \{\mathbf{Know}(\psi(\mathsf{now}), S_0)\}$. Thus, $\mathcal{E} \cup \mathcal{D}_{una} \cup \psi(S_0) \cup \{\neg\phi(t_i, S_0)\}$ is satisfiable for for every $i \geq 1$. Moreover, by Lemma 5, $\mathcal{E} \cup \mathcal{D}_{una} \cup \psi(S_0) \cup \{\neg\phi(t_i, S_0)\}$ is satisfiable in a model M_i where every element in the object, action, and program sorts has a name in the language.

With all this, we can safely assume that, for any M_i, the object sort of M_i is $D_O = \{\overline{t}^o_1, \overline{t}^o_2, ..\}$ and that $[t^o]^{M_i} = \overline{t^o}$. Similarly, we can also assume that the action sort of M_i is $D_A = \{\overline{t}^a_1, \overline{t}^a_2, ..\}$ and that $[t^a]^{M_i} = \overline{t^a}$; and, finally, that the program sort of M_i is $D_P = \{\overline{t}^p_1, \overline{t}^p_2, ..\}$ and that $[t^p]^{M_i} = \overline{t^p}$.

Intuitively, with these assumptions on the form of M_i, all models M_i will coincide exactly on the way they interpret every term, and, therefore, we will be able to amalgamate all them together in a single big model.

Next, we are to show that, based on all these models $M_1, M_2, ...$ (one for each object/action/program term), we can construct an amalgamated model M^* of $\mathcal{E} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ such that the following holds: $M^* \models \neg\exists x.\mathbf{Know}(\phi(x, \mathsf{now}), S_0)$. That would imply that $\mathcal{E} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \cup \mathcal{K}_{Init} \not\models \exists x.\mathbf{Know}(\phi(x, \mathsf{now}), S_0)$ and, by Lemma 2, $\mathcal{D} \cup \mathcal{E} \not\models \exists x.\mathbf{Know}(\phi(x, \mathsf{now}), S_0)$ would follow (i.e., a contradiction).

Let us construct a model M^* as follows:

(a) $S_0, S_1, S_2, ...$ are all initial situations in the sort situation of M^*; $[s_t]^{M^*} = s_t$ for every possible situation term s_t; and $M^* \models K(u', u)$ iff $u = S_0$ and $u' = S_i$ or $u = u'$, for $i \geq 0$;

(b) M^*'s domains for objects, actions, and programs are D_O, D_A, and D_P respectively;

(c) $[t^o]^{M^*} = \overline{t^o}$, $[t^a]^{M^*} = \overline{t^a}$, and $[t^p]^{M^*} = \overline{t^p}$, that is M^* interprets non-situation terms as any M_i does;

(d) for any $i \geq 0$, if $t^1, ..., t^n$ are non-situation domain elements, and P is an n-place relational fluent, then $M^* \models P(t^1, ..., t^n, S_i)$ iff $M_i \models P(t^1, ..., t^n, S_0)$;

(e) assign the rest arbitrarily (for instance, the interpretation of relational fluents in situations other than the initial ones.)

Intuitively, each model M_i is recast as a K-accessible initial situation in model M^*. Notice that point (c) guarantees that for any two non-situation terms t_1 and t_2, $M^* \models t^1 = t^2$ iff $M^i \models t^1 = t^2$ for any arbitrary $i \geq 1$. (*) Similarly, point (d) is well-defined as well as there are no functional fluents and M^* has the same object, action, and program sorts as each M_i.

It is not hard to see that for any objective sentence $\alpha(\mathsf{now})$, $M^* \models \alpha(S_i)$ iff $M_i \models \alpha(S_0)$ (by induction on the structure of $\alpha(s)$ with the base case being atomic formulas, that is, either a relational fluent or an equality term.) (**)

Next, we are to prove that $M^* \models \mathcal{E} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \cup \mathcal{K}_{Init}$. First, given that $M_i \models \psi(S_0)$ for all $i \geq 1$, $M^* \models \psi(S_i)$ by (**). Hence, $M^* \models \forall s.K(s,S_0) \supset \psi(s)$ and $M^* \models \mathcal{D}_{S_0}$. Second, $M^* \models \mathcal{K}_{Init}$ due to point (a) above. Finally, $M^* \models \mathcal{D}_{una}$ because $M_i \models \mathcal{D}_{una}$ for all $i \geq 1$ and point (*) above. Third, $M^* \models \mathcal{E}$ because M^* domain for programs is D_P and the interpretation of all programs in D_P and program terms in the language are exactly the same as in any M_i. Putting all together, $M^* \models \mathcal{E} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \cup \mathcal{K}_{Init}$.

Lastly, let us prove $M^* \models \neg\exists x.\mathbf{Know}(\phi(x,\mathsf{now}),S_0)$. Given that $M_i \models \neg\phi(t_i,S_0)$ and point (d) above, $M^* \models \neg\phi(t_i,S_i)$ is true for every $i \geq 1$. In English, this means that for every possible non-situation term t, there is an accessible world in which $\phi(t,\mathsf{now})$ does not hold. In particular, if x is an object variable, then *for every* object element $\overline{t^o} \in D_O$, there is an initial accessible situation $S_{\overline{t^o}}$ such that $M^* \models \neg\phi(\overline{t^o},S_{\overline{t^o}})$; if x is an action variable, then *for every* action element $\overline{t^a} \in D_A$, there is an initial accessible situation $S_{\overline{t^a}}$ such that $M^* \models \neg\phi(\overline{t^a},S_{\overline{t^a}})$; and if x is a program variable, then *for every* program element $\overline{t^p} \in D_P$, there is an initial accessible situation $S_{\overline{t^p}}$ such that $M^* \models \neg\phi(\overline{t^p},S_{\overline{t^p}})$

Therefore, since x is either an object, action, or program variable, we have that $M^* \models \forall x\exists s.K(s,S_0) \wedge \neg\phi(x,s)$. In other words, $M^* \models \neg\exists x.\mathbf{Know}(\phi(x,\mathsf{now}),S_0)$; and, hence, $\mathcal{E} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \cup \mathcal{K}_{Init} \not\models \exists x.\mathbf{Know}(\phi(x,\mathsf{now}),S_0)$ so that, by Lemma 2, $\mathcal{D} \cup \mathcal{E} \not\models \exists x.\mathbf{Know}(\phi(x,\mathsf{now}),S_0)$ follows. As this contradicts the initial statement, it has to be the case that there exists some (object, action, or program correspondingly) term t such that $\mathcal{D} \cup \mathcal{E} \models \mathbf{Know}(\phi(t,\mathsf{now}),S_0)$.

A.2. PROOFS OF SECTION 3

Proof of Theorem 1. We prove this by induction on the structure of $p(\vec{x})$, taking *nil*, $(\beta)?$, and primitive actions as base cases.

Base Case: Take for instance $p(\vec{x}) = A(\vec{x})$ where A is a primitive action. Then, we take $\phi_f(\vec{x},s) = \phi_{tt}(\vec{x},p',s) = \mathit{False}$ and $\phi_{ta}(\vec{x},p',a,s) = \Pi(\vec{x},s) \wedge p' = nil \wedge a = A(\vec{x})$, where $\Pi(\vec{x},s)$ is the precondition of action type A.

Take next the case where $p(\vec{x}) = (\beta(x))?$, that is a test program. Then, we take $\phi_f(\vec{x},s) = \phi_{ta}(\vec{x},p',a,s) = \mathit{FALSE}$ and $\phi_{tt}(\vec{x},p',s) = \beta(\vec{x},s) \wedge p' = nil$.

Induction Step: We only show the case for sequence, pick for an action, and prioritized concurrency.

Suppose $p(\vec{x}) = p_1(\vec{x}); p_2(\vec{x})$. Then, we take

$$\begin{aligned}
\phi_f(\vec{x},s) &= \phi_f^{p_1}(\vec{x},s) \wedge \phi_f^{p_2}(\vec{x},s) \\
\phi_{tt}(\vec{x},p',s) &= [\exists p''.p' = p''; p_2(\vec{x}) \wedge \phi_{tt}^{p_1}(\vec{x},p'',s)] \vee [\phi_f^{p_1}(\vec{x},s) \wedge \phi_{tt}^{p_2}(\vec{x},p',s)] \\
\phi_{ta}(\vec{x},p',a,s) &= [\exists p''.p' = p''; p_2(\vec{x}) \wedge \phi_{ta}^{p_1}(\vec{x},p'',a,s)] \vee [\phi_f^{p_1}(\vec{x},s) \wedge \phi_{ta}^{p_2}(\vec{x},p',a,s)]
\end{aligned}$$

where $\phi_f^{p_i}(\vec{x},s)$, $\phi_{ta}^{p_i}(\vec{x},p',a,s)$ and $\phi_{tt}^{p_i}(\vec{x},s)$ for $i = 1$ and $i = 2$ come from the induction hypothesis.

Suppose $p(\vec{x}) = \pi a.p_1(a,\vec{x})$. Given that we have a finite set of action types (assumption 6 in Section 3), we can rewrite program $p(\vec{x})$ as $p_1(A_1(\vec{x}),\vec{x})|...|p_1(A_n(\vec{x}),\vec{x})$

assuming $A_1,...,A_n$ are all the action types available. The rest of the proof is similar to the previous case.

Lastly, suppose that $p(\vec{x}) = p_1(\vec{x}) \rangle\rangle\ \ p_1(\vec{x})$. Then, we take

$$\begin{aligned}
\phi_f(\vec{x},s) &= \phi_f^{p_1}(\vec{x},s) \wedge \phi_f^{p_2}(\vec{x},s) \\
\phi_{tt}(\vec{x},p',s) &= [\exists p''.p' = p'' \rangle\rangle\ \ p_2(\vec{x}) \wedge \phi_{tt}^{p_1}(\vec{x},p'',s)] \vee \\
&\quad [\exists p''.p' = p_1(\vec{x}) \rangle\rangle\ \ p'' \wedge \phi_{tt}^{p_2}(\vec{x},p'',s) \wedge \\
&\qquad \neg\exists p'''.\phi_{tt}^{p_1}(\vec{x},p''',s) \vee \exists a\, \phi_{ta}^{p_1}(\vec{x},p''',a,s)] \\
\phi_{ta}(\vec{x},p',a,s) &= [\exists p''.p' = p'' \rangle\rangle\ \ p_2(\vec{x}) \wedge \phi_{ta}^{p_1}(\vec{x},p'',a,s)] \vee \\
&\quad [\exists p''.p' = p_1(\vec{x}) \rangle\rangle\ \ p'' \wedge \phi_{ta}^{p_2}(\vec{x},p'',a,s) \wedge \\
&\qquad \neg\exists p'''.\phi_{tt}^{p_1}(\vec{x},p''',s) \vee \exists a\, \phi_{ta}^{p_1}(\vec{x},p''',a,s)]
\end{aligned}$$

where $\phi_f^{p_i}(\vec{x},s)$ $\phi_{ta}^{p_i}(\vec{x},p',a,s)$, and $\phi_{tt}^{p_i}(\vec{x},s)$ for $i=1$ and $i=2$ come from the induction hypothesis. Note that this relies on the fact that

$$\begin{aligned}
&\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \models \\
&\quad \exists p',s'\, Trans(p,s,p',s') \equiv \exists p'\, Trans(p,s,p',s) \vee \exists p',a\, Trans(p,s,p',do(a,s))
\end{aligned}$$

which follows from $\mathcal{D} \cup \mathcal{T} \cup \mathcal{E} \models Trans(p,s,p',s') \supset s' = s \vee \exists a\, s' = do(a,s)$. The latter is easy to prove by induction on programs.

Proof of Theorem 2.

$\Leftarrow$ This way follows easily from the fact that reflexivity of K propagates through action from the initial situation.

$\Rightarrow$ We prove this direction by induction on the length of σ. For the base case, take σ to be the initial history and suppose $\mathcal{D} \cup \mathcal{E} \models \phi(S_0)$ holds. Then, $\mathcal{D}_{S_0} \cup \mathcal{K}_{Init} \cup \mathcal{E} \cup \mathcal{D}_{una} \models \phi(S_0)$ (i.e., $\mathcal{D}_{ap}, \mathcal{D}_{ss}, \mathcal{D}_{sf}$ and the foundational axioms can all be ignored since the formula in question only talks about S_0). Then, $\mathcal{D}_{S_0} \cup \mathcal{K}_{Init} \cup \mathcal{E} \cup \mathcal{D}_{una} \cup \{\psi(S_0)\} \models \phi(S_0)$, where $\mathcal{D}_{S_0} = \mathbf{Know}(\psi(\mathsf{now}), S_0)$. Finally, since $\phi(S_0)$ is objective we can safely drop both $\mathcal{D}_{S_0}$ and $\mathcal{K}_{Init}$ and, hence, $\mathcal{E} \cup \mathcal{D}_{una} \cup \{\psi(S_0)\} \models \phi(S_0)$. At this point, we can directly appeal to Lemma 3.

Assume next that $\sigma^+ = \sigma \cdot (A,\mu)$ for some ground action A and history σ of length k. Consider the case where $A = sense_\psi(\vec{t})$ and $\mu = 1$. Then $Sensed[\sigma^+] = Sensed[\sigma] \wedge \psi(\vec{t}, end[\sigma])$. We have that

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \phi(end[\sigma^+])$$

By the successor state axioms, it follows that

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \rho^1(\phi(\mathsf{now}), sense_\psi(\vec{t}))[end[\sigma]]$$

Since $Sensed[\sigma^+] = Sensed[\sigma] \wedge \psi(\vec{t}, end[\sigma])$, it follows that

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \psi(\vec{t}, end[\sigma]) \supset \rho^1(\phi(\mathsf{now}), sense_\psi(\vec{t}))[end[\sigma]]$$

By the induction hypothesis, we then have that

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \mathbf{Know}(\psi(\vec{t}, \mathsf{now}) \supset \rho^1(\phi(\mathsf{now}), sense_\psi(\vec{t})), end[\sigma])$$

and thus also that

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \mathbf{Know}(\psi(\vec{t}, \mathsf{now}) \supset \rho^1(\phi(\mathsf{now}), sense_\psi(\vec{t})), end[\sigma])$$

Then by Proposition 11.6.2 in [25], it easy to see that

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \mathbf{Know}(\phi(\mathsf{now}), end[\sigma^+])$$

The case where $A = sense_\psi(\vec{t})$ and $\mu = 0$ is similar.

Now consider the case where A is a non-sensing action. Then $\mu = 1$ and $\mathcal{D}_{sf} \models Sensed[\sigma^+] \equiv Sensed[\sigma]$. We have that

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \phi(end[\sigma^+])$$

By the successor state axioms, it follows that

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \rho^1(\phi(\mathsf{now}), A)[end[\sigma]]$$

Since $\mathcal{D}_{sf} \models Sensed[\sigma^+] \equiv Sensed[\sigma]$, it follows that

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \rho^1(\phi(\mathsf{now}), A)[end[\sigma]]$$

By the induction hypothesis, we then have that

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \mathbf{Know}(\rho^1(\phi(\mathsf{now}), A), end[\sigma])$$

and thus also that

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \mathbf{Know}(\rho^1(\phi(\mathsf{now}), A), end[\sigma])$$

Then by Proposition 11.6.1 in [25], we have that

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \mathbf{Know}(\phi(\mathsf{now}), end[\sigma^+])$$

Proof of Theorem 3. a $\Leftarrow$) This direction is trivial.

$\Rightarrow$) For simplicity, we prove this for two disjuncts, that is for $n = 2$. The proof can be easily extended to an arbitrary n.

The proof goes by induction on the length of the history. The base case, that is, when σ is the initial history, follows from Lemma 4.

Next, assume that the Theorem holds for any history σ of length $\leq k$. Suppose that $\sigma^+ = \sigma \cdot (A, 1)$ for some ground action A and history σ of length k (the case for $\sigma^+ = \sigma \cdot (A, 0)$ is similar.) Suppose further that

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \mathbf{Know}(\phi_1(\mathsf{now}), end[\sigma^+]) \vee \mathbf{Know}(\phi_2(\mathsf{now}), end[\sigma^+])$$

Using Proposition 11.6.2 in [25], the fact that $\sigma^+ = \sigma \cdot (A, 1)$, and the fact that $Sensed[\sigma^+] = Sensed[\sigma] \cup \{\psi(end[\sigma])\}$ we conclude that

$$\begin{array}{l} \mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \\ \quad \mathbf{Know}(\phi_1(\mathsf{now}), end[\sigma^+]) \equiv \mathbf{Know}(\psi(\mathsf{now}) \supset \rho^1(\phi_1(\mathsf{now}), A), end[\sigma]) \end{array} \quad (2)$$

$$\begin{array}{l} \mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \\ \quad \mathbf{Know}(\phi_2(\mathsf{now}), end[\sigma^+]) \equiv \mathbf{Know}(\psi(\mathsf{now}) \supset \rho^1(\phi_2(\mathsf{now}), A), end[\sigma]) \end{array} \quad (3)$$

Therefore,

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \mathbf{Know}(\psi(\mathsf{now}) \supset \rho^1(\phi_1(\mathsf{now}), A), end[\sigma]) \vee \mathbf{Know}(\psi(\mathsf{now}) \supset \rho^1(\phi_2(\mathsf{now}), A), end[\sigma]) \quad (4)$$

At the same time, by Lemma 11.7.10 in [25], there is an objective formula $\psi^*(\mathsf{now})$ ($\psi^*(\mathsf{now}) = \rho(Sensed[\sigma], end[\sigma]) \supset \rho(\psi, end[\sigma])$) such that

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \mathbf{Know}(\psi(\mathsf{now}), end[\sigma]) \equiv \mathbf{Know}(\psi^*(\mathsf{now}), S_0) \quad (5)$$

In other words, coming to know that ψ holds at history σ is equivalent to coming to know that ψ^* holds initially.

Next, we define $\mathcal{D}^*$ to be like $\mathcal{D}$, but with $\psi^*(\mathsf{now})$ added to the the initial database $\mathcal{D}_{S_0}$, that is, if $\mathcal{D}_{S_0} = \{\mathbf{Know}(\psi_0(\mathsf{now}), S_0)\}$ then $\mathcal{D}^*_{S_0} = \{\mathbf{Know}(\psi_0(\mathsf{now}) \wedge \psi^*(\mathsf{now}), S_0)\}$. Let us argue that $\mathcal{D} \cup \mathcal{E} \cup Sensed[\sigma^+]$ and $\mathcal{D}^* \cup \mathcal{E} \cup Sensed[\sigma]$ are equivalent sets of axioms. First, observe that since $\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \psi(end[\sigma])$, then $\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \mathbf{Know}(\psi(\mathsf{now}), end[\sigma])$ by Theorem 2. By (5), $\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \mathbf{Know}(\psi^*(\mathsf{now}), S_0)$. Thus, $\mathcal{D} \cup \mathcal{E} \cup Sensed[\sigma^+]$ logically entails $\mathcal{D}^* \cup \mathcal{E} \cup Sensed[\sigma]$.

Moreover, by Lemma 11.7.10 in [25], we know that the following holds:

$$\mathcal{D}^* \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \mathbf{Know}(\psi(\mathsf{now}), end[\sigma]) \equiv \mathbf{Know}(\psi^*(\mathsf{now}), S_0)$$

and because $\mathcal{D}^* \cup \mathcal{E} \models \mathbf{Know}(\psi^*(\mathsf{now}), S_0)$ we conclude that

$$\mathcal{D}^* \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \mathbf{Know}(\psi(\mathsf{now}), end[\sigma])$$

Using Theorem 2 we get that

$$\mathcal{D}^* \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \psi(end[\sigma])$$

and $\mathcal{D}^* \cup \mathcal{E} \cup Sensed[\sigma] \models \mathcal{D} \cup \mathcal{E} \cup Sensed[\sigma^+]$ applies. Hence, $\mathcal{D} \cup \mathcal{E} \cup Sensed[\sigma^+]$ and $\mathcal{D}^* \cup \mathcal{E} \cup Sensed[\sigma]$ are equivalent sets of axioms.

As a consequence of this and (4) the following holds:

$$\mathcal{D}^* \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \mathbf{Know}(\psi(\mathsf{now}) \supset \rho^1(\phi_1(\mathsf{now}), A), end[\sigma]) \vee \mathbf{Know}(\psi(\mathsf{now}) \supset \rho^1(\phi_2(\mathsf{now}), A), end[\sigma])$$

Given that $\psi^*(\mathsf{now})$, $\psi(\mathsf{now}) \supset \rho^1(\phi_1(\mathsf{now}), A)$, and $\psi(\mathsf{now}) \supset \rho^1(\phi_2(\mathsf{now}), A)$ are objective, and that σ is of length k, we can apply the induction hypothesis: hence, one of the following two cases holds:

(i) $\mathcal{D}^* \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \mathbf{Know}(\psi(\mathsf{now}) \supset \rho^1(\phi_1(\mathsf{now}), A), end[\sigma])$;

(ii) $\mathcal{D}^* \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \mathbf{Know}(\psi(\mathsf{now}) \supset \rho^1(\phi_2(\mathsf{now}), A), end[\sigma])$.

Assume (i) holds. Again, as $\mathcal{D} \cup \mathcal{E} \cup Sensed[\sigma^+]$ and $\mathcal{D}^* \cup \mathcal{E} \cup Sensed[\sigma]$ are equivalent set of axioms, we get

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \mathbf{Know}(\psi(\mathsf{now}) \supset \rho^1(\phi_1(\mathsf{now}), A), end[\sigma])$$

By Proposition 11.6.2. in [25], and (2),

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \mathbf{Know}(\phi_1(\mathsf{now}), end[\sigma^+])$$

The case for (ii) is similar and the theorem follows.

Proof of Theorem 4. $\Leftarrow$) This direction is trivial.

$\Rightarrow$) The proof goes by induction on the length of the history. The base case, that is, when σ is the initial history, corresponds to Lemma 6 above.

Assume that the Theorem holds for any history σ of length $\leq k$. Suppose that $\sigma^+ = \sigma \cdot (A,1)$ for some ground action A and history σ of length k (the case for $\sigma^+ = \sigma \cdot (A,0)$ is analogous.) Suppose further that

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \exists \vec{x}.\mathbf{Know}(\phi(\vec{x}, \mathsf{now}), end[\sigma^+])$$

From Proposition 11.6.2 in [25], the fact that $\sigma^+ = \sigma \cdot (A,1)$, and the fact that $Sensed[\sigma^+] = Sensed[\sigma] \wedge \psi(end[\sigma])$, we have that

$$\begin{array}{l} \mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \\ \quad \exists \vec{x}.\mathbf{Know}(\phi(\vec{x}, \mathsf{now}), end[\sigma^+]) \equiv \exists x.\mathbf{Know}(\psi(\mathsf{now}) \supset \rho^1(\phi(\vec{x}, \mathsf{now}), A), end[\sigma]) \end{array} \quad (6)$$

Therefore,

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \exists x.\mathbf{Know}(\psi(\mathsf{now}) \supset \rho^1(\phi(\vec{x}, \mathsf{now}), A), end[\sigma]) \quad (7)$$

At the same time, by Lemma 11.7.10 in [25], there is some formula $\psi^*(\mathsf{now})$ ($\psi^*(\mathsf{now}) = \rho(Sensed[\sigma], end[\sigma]) \supset \rho(\psi, end[\sigma])$) such that

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \mathbf{Know}(\psi(\mathsf{now}), \sigma) \equiv \mathbf{Know}(\psi^*(\mathsf{now}), S_0) \quad (8)$$

In other words, coming to know that ψ holds at history σ is equivalent to coming to know that ψ^* holds initially.

Next, we define $\mathcal{D}^*$ to be like $\mathcal{D}$, but with $\psi^*(\mathsf{now})$ added to the the initial database $\mathcal{D}_{S_0}$, that is, if $\mathcal{D}_{S_0} = \{\mathbf{Know}(\psi_0(\mathsf{now}), S_0)\}$ then $\mathcal{D}^*_{S_0} = \{\mathbf{Know}(\psi_0(\mathsf{now}) \wedge \psi^*(\mathsf{now}), S_0)\}$. As already done in the proof of Theorem 3, it is possible to demonstrate that $\mathcal{D} \cup \mathcal{E} \cup Sensed[\sigma^+]$ and $\mathcal{D}^* \cup \mathcal{E} \cup Sensed[\sigma]$ are equivalent sets of axioms.

As a consequence of that and equation (7) the following holds:

$$\mathcal{D}^* \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \exists x.\mathbf{Know}(\psi(\mathsf{now}) \supset \rho^1(\phi(\vec{x}, \mathsf{now}), A), end[\sigma])$$

Given that both $\psi^*(\mathsf{now})$ and $\psi(\mathsf{now}) \supset \rho^1(\phi(\vec{x}, \mathsf{now}), A)$ are objective formulas and σ is of length k, we can apply the induction hypothesis; thus, there exist ground terms $\vec{t}$ such that

$$\mathcal{D}^* \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \mathbf{Know}(\psi(\mathsf{now}) \supset \rho^1(\phi(\vec{t}, \mathsf{now}), A), end[\sigma])$$

Again since $\mathcal{D} \cup \mathcal{E} \cup Sensed[\sigma^+]$ and $\mathcal{D}^* \cup \mathcal{E} \cup Sensed[\sigma]$ are equivalent we conclude that

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \mathbf{Know}(\psi(\mathsf{now}) \supset \rho^1(\phi(\vec{t}, \mathsf{now}), A), end[\sigma])$$

From Proposition 11.6.2 in [25], the fact that $\sigma^+ = \sigma \cdot (A,1)$, and the fact that $Sensed[\sigma^+] = Sensed[\sigma] \cup \{\psi(end[\sigma])\}$, we have that

$$\begin{array}{l} \mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \\ \quad \mathbf{Know}(\phi(\vec{t}, \mathsf{now}), end[\sigma^+]) \equiv \mathbf{Know}(\psi(\mathsf{now}) \supset \rho^1(\phi(\vec{t}, \mathsf{now}), A), end[\sigma]) \end{array}$$

and, therefore, we conclude that

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma^+]\} \models \mathbf{Know}(\phi(\vec{t}, \mathsf{now}), end[\sigma^+])$$

LEMMA 7 (Disjunctive and Mutually Exclusive Knowledge with Existentials). *Let* $\phi_1(\mathsf{now})$, $\phi_2(\vec{x}, \mathsf{now})$, *and* $\phi_3(\vec{y}, \mathsf{now})$ *be three objective formulas with non-situation free variables* $\vec{x}$ *and* $\vec{y}$. *If*

$$\begin{aligned}\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models\ & \mathbf{Know}(\phi_1(\mathsf{now}), end[\sigma]) \vee \\ & \exists \vec{x}.\mathbf{Know}(\neg\phi_1(\mathsf{now}) \wedge \phi_2(\vec{x}, \mathsf{now}) \wedge \forall \vec{y}.\neg\phi_3(\vec{y}, \mathsf{now}), end[\sigma]) \vee \\ & \exists \vec{y}.\mathbf{Know}(\neg\phi_1(\mathsf{now}) \wedge \forall \vec{x}.\neg\phi_2(\vec{x}, \mathsf{now}) \wedge \phi_3(\vec{y}, \mathsf{now}), end[\sigma])\end{aligned}$$

then one of the following cases applies:

- $\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \mathbf{Know}(\phi_1(\mathsf{now}), end[\sigma])$*;*
- $\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \exists \vec{x}.\mathbf{Know}(\phi_2(\vec{x}, \mathsf{now}), end[\sigma])$*; or*
- $\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \exists \vec{y}.\mathbf{Know}(\phi_k(\vec{y}, \mathsf{now}), end[\sigma])$.

Proof. First notice that, due to properties of knowledge we can push the existential quantifiers inside the **Know** modality:

$$\begin{aligned}\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models\ & \mathbf{Know}(\phi_1(\mathsf{now}), end[\sigma]) \vee \\ & \mathbf{Know}(\neg\phi_1(\mathsf{now}) \wedge \exists \vec{x}.\phi_2(\vec{x}, \mathsf{now}) \wedge \forall \vec{y}.\neg\phi_3(\vec{y}, \mathsf{now}), end[\sigma]) \vee \\ & \mathbf{Know}(\neg\phi_1(\mathsf{now}) \wedge \forall \vec{x}.\neg\phi_2(\vec{x}, \mathsf{now}) \wedge \exists \vec{y}.\phi_3(\vec{y}, \mathsf{now}), end[\sigma])\end{aligned}$$

By Theorem 3, $\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\}$ logically entails one of the following formulas: (i) $\mathbf{Know}(\phi_1(\mathsf{now}), end[\sigma])$; (ii) $\mathbf{Know}(\neg\phi_1(\mathsf{now}) \wedge \exists \vec{x}.\phi_2(\vec{x}, \mathsf{now}) \wedge \forall \vec{y}.\neg\phi_3(\vec{y}, \mathsf{now}), end[\sigma])$; or (iii) $\mathbf{Know}(\neg\phi_1(\mathsf{now}) \wedge \forall \vec{x}.\neg\phi_2(\vec{x}, \mathsf{now}) \wedge \exists \vec{y}.\phi_3(\vec{y}, \mathsf{now}), end[\sigma])$. In the first case, we are done easily. If (ii) applies, then by properties of knowledge $\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \mathbf{Know}(\neg\phi_1(\mathsf{now}), end[\sigma]) \wedge \mathbf{Know}(\forall \vec{y}.\neg\phi_3(\vec{y}, \mathsf{now}), end[\sigma])$. Hence, by properties of knowledge, we can pull out the universal quantifier from inside the **Know** modality so that $\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \forall \vec{y}.\mathbf{Know}(\neg\phi_3(\vec{y}, \mathsf{now}), end[\sigma])$ holds and, as a result, $\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \forall \vec{y}.\neg\mathbf{Know}(\phi_3(\vec{y}, \mathsf{now}), end[\sigma])$. Then, given the initial assumption, as the first and the third disjunct are ruled out, the following should hold

$$\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \exists \vec{x}.\mathbf{Know}(\neg\phi_1(\mathsf{now}) \wedge \forall \vec{y}.\neg\phi_3(\vec{y}, \mathsf{now}) \wedge \phi_2(\vec{x}, \mathsf{now}), end[\sigma])$$

from which $\mathcal{D} \cup \mathcal{E} \cup \{Sensed[\sigma]\} \models \exists \vec{x}.\mathbf{Know}(\phi_2(\vec{x}, \mathsf{now}), end[\sigma])$ follows directly.

Case (iii) is analogous.