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

Giuseppe De Giacomo
Dip. Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
degiacomo@dis.uniroma1.it

Yves Lespérance
Dept. of Computer Science
York University
Toronto, ON, M3J 1P3, Canada
lesperan@cs.yorku.ca

Hector J. Levesque and Sebastian Sardiña
Dept. of Computer Science
University of Toronto
Toronto, ON, M5S 3G4, Canada
{hector,ssardina}@ai.toronto.edu

Abstract

In this paper, 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 IndiGolog agent language in terms of it. We also show 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. We also discuss implementation issues and execution monitoring and replanning.

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 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 programs in the Golog family of high-level programming languages [Levesque et al., 1997, De Giacomo et al., 2000, Reiter, 2001a]. Those 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 sub-tasks to be handled by an automatic planner. In that way, a high-level program serves as a "guide" heavily restricting the search space. By a high-level program, we mean one whose primitive instructions are domain-dependent actions of the robot, whose tests involve domain-dependent fluents affected by these actions, and whose 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 [Levesque, 1996] 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 [Peot and Smith, 1992, Smith et al., 1998, Lakemeyer, 1999]. 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 [De Giacomo and Levesque, 1999a] 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.

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 a basic property they have w.r.t. 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 future and related 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 [De Giacomo and Levesque, 1999a, De Giacomo et al., 2000]. The starting point in the definition is the situation calculus [McCarthy and Hayes, 1979]. 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. To deal with knowledge and sensing, we follow [Moore, 1985, Scherl and Levesque, 1993, Levesque, 1996] and use a fluent $K(s', s)$ used to represent what situations s' are considered epistemically possible by the agent in situation s. **Know**$(\phi(now), s)$ is then taken to be an abbreviation for the formula $\forall s'.K(s', s) \supset \phi(now/s')$. In this paper, we only deal explicitly with sensing actions with binary outcomes as in [Levesque, 1996]. However, the results presented here can be easily generalized to sensors with multiple outcomes. To represent the information provided by a sensing action, we use a predicate $SF(a, s)$, which holds if action a returns the binary sensing result 1 in situation s. 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 use $[SF(a, s) \equiv True]$.

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 of the following form [Reiter, 1991, 2001a]:

- Axioms describing the initial situation, S_0.
- Action precondition axioms, one for each primitive action a, characterizing $Poss(a, s)$.
- 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.
- Sensed fluent axioms, one for each primitive action a of the form $SF(a, s) \equiv \phi_a(s)$, characterizing SF [Levesque, 1996].
- The following successor state axiom for the knowledge fluent K [Scherl and Levesque, 1993]:

$$\begin{aligned} &K(s'', do(a, s)) \equiv \\ &\quad \exists s'.s' = do(a, s') \wedge K(s', s) \wedge Poss(a, s') \ \wedge \\ &\qquad [SF(a, s') \equiv SF(a, s)] \end{aligned}$$

- Unique names axioms for the primitive actions.
- Some foundational, domain independent axioms [Lakemeyer and Levesque, 1998, Reiter, 2001a].

To describe a run which includes both actions and their sensing results, we use the notion of a *history*, i.e., a sequence of pairs (a, x) where a is a primitive action and x is 1 or 0, a sensing result. Intuitively, the history $(a_1, x_1) \cdot \ldots \cdot (a_n, x_n)$ 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 $x_i = 1$. 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: $end[\epsilon] = S_0$; and inductively, $end[\sigma \cdot (a, x)] = do(a, end[\sigma])$.

We also use $Sensed[\sigma]$ as an abbreviation for a formula of the situation calculus, the *sensing results* of a history, and defined by: $Sensed[\epsilon] = True$; 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.

Next we turn to programs. The programs we consider here are based on the ConGolog language defined in [De Giacomo et al., 2000], 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[1]

Among these constructs, we notice the presence of 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 [De Giacomo et al., 2000] for a detailed account of ConGolog.

In [De Giacomo et al., 2000], a single step transition semantics in the style of [Plotkin, 1981] 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.[2]

Offline executions of programs, which are the kind of executions originally proposed for Golog and ConGolog [Levesque et al., 1997, De Giacomo et al., 2000], are characterized using the $Do(p, s, s')$ predicate, which means that there is an execution of program p 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 $Trans$. An offline execution of program p from situation s is a sequence of actions $a_1, \ldots, a_n$ such that:

$$Axioms \models Do(p, s, do(a_n, \ldots, do(a_1, s))).$$

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 [De Giacomo et al., 2000] for more details.

In [De Giacomo and Levesque, 1999a], IndiGolog, an extension of ConGolog that deals with online executions with sensing is developed. The semantics defines an *online execution* of an IndiGolog program p starting from a history σ, as a sequence of *online configurations* $(p_0 = p, \sigma_0 = \sigma), \ldots, (p_n, \sigma_n)$ such that for $i = 0, \ldots, n{-}1$:

$$\begin{array}{l} Axioms \cup \{Sensed[\sigma_i]\} \models \\ \qquad\qquad Trans(p_i, end[\sigma_i], p_{i+1}, end[\sigma_{i+1}]), \end{array}$$

$$\sigma_{i+1} = \begin{cases} \sigma_i & \text{if} \quad end[\sigma_{i+1}] = end[\sigma_i], \\ \sigma_i \cdot (a, x) & \text{if} \quad end[\sigma_{i+1}] = do(a, end[\sigma_i]) \\ & \qquad \text{and } a \text{ returns } x. \end{cases}$$

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

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

$$\begin{array}{l} Trans([p_1; p_2], s, p', s') \equiv \\ \quad Final(p_1, s) \wedge Trans(p_2, s, p', s') \vee \\ \quad\quad \exists q'.\, Trans(p_1, s, q', s') \wedge p' = (q'; p_2) \end{array}$$

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.

An *online execution successfully terminates* if

$$Axioms \cup \{Sensed[\sigma_n]\} \models Final(p_n, end[\sigma_n]).$$

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.,

$$\begin{aligned} & Trans(\Sigma(p), s, p', s') \equiv \\ & \quad \exists q', s_f.\, p' = \Sigma(q') \wedge \\ & \qquad Trans(p, s, q', s') \wedge Do(q', s', s_f). \end{aligned}$$

This semantics means that $Axioms \cup \{Sensed[\sigma]\} \models Trans(\Sigma(p), end[\sigma], \Sigma(p'), s')$ iff $Axioms \cup \{Sensed[\sigma]\} \models Trans(p, end[\sigma], p', s')$ and $Axioms \cup \{Sensed[\sigma]\} \models \exists s_f. Do(p', s', s_f)$. 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 $(a; \textbf{if } \phi \textbf{ then } b \textbf{ else } c) \mid d$, 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.

3 EPISTEMICALLY ACCURATE THEORIES

In this paper we are going to look at theories that are *epistemically accurate*, meaning that what is known accurately reflects what the theory says about the dynamic system.[3] Formally, epistemically accurate theories are theories as introduced earlier, but with two additional constraints:

- The initial situation is characterized by an axiom of the form **Know**(ϕ, S_0) where ϕ is an *objective formula*, i.e., a formula where the knowledge fluent K does not appear, which describes the initial situation, S_0. Note that there can be fluents about which nothing is known in the initial situation.
- There is an axiom stating that the accessibility relation K is reflexive in the initial situation, which is then propagated to all situations by the successor state axiom for K [Scherl and Levesque, 1993].

[3] In [Reiter, 2001b] a similar notion is used to deal with knowledge-based programs and reduce knowledge to provability.

For epistemically accurate theories we have established the following result:

Theorem 1 *For any objective sentence about situation s, $\phi(s)$ ($Trans$ and $Final$ may appear in $\phi(s)$), $Axioms \cup \{Sensed[\sigma]\} \models \phi(end[\sigma])$ if and only if $Axioms \cup \{Sensed[\sigma]\} \models$ **Know**$(\phi, end[\sigma])$.*

Proof Sketch: $\Leftarrow$ Follows trivially from the reflexivity of K in the initial situation, and the fact that it is preserved by the successor state axiom for K.

$\Rightarrow$ Suppose the thesis does not hold, i.e., there exists a model M of $Axioms \cup \{Sensed[\sigma]\}$ such that for some s', $M \models K(s', end[\sigma])$ and $M \models \neg\phi(s')$.

Then take the structure M' obtained from M by intersecting the objects of sort situation with those that are in the situation tree rooted in the initial ancestor situation of s', say s'_0. M' satisfies all the axioms in $Axioms$ except the reflexivity and the successor state axiom for K, and the initial state axiom, which is of the form **Know**$(\Psi(now), S_0)$ (the other axioms involve neither K nor S_0). Observe that $Trans$ and $Final$ for the situations in the tree are defined by considering relations involving only situations in the same tree.

Now consider the M'' obtained from M' by adding the constant S_0 and making it denote s'_0. Although M' and M'' do not satisfy **Know**$(\Psi(now), S_0)$, we have that $M'' \models \Psi(S_0)$. Moreover, the successor state axiom for K implies

$$\begin{aligned} & Axioms \cup \{Sensed[\sigma'] \cdot (a, 1)\} \models \\ & \qquad \textbf{Know}(SF(a, now), end[\sigma' \cdot (a, 1)]) \\ & Axioms \cup \{Sensed[\sigma'] \cdot (a, 0)\} \models \\ & \qquad \textbf{Know}(\neg SF(a, now), end[\sigma' \cdot (a, 0)]) \end{aligned}$$

and the fact that the successor state axiom for K holds in M ensures that all predecessors of s' are K accessible from predecessors of $end[\sigma]$ in M. These imply that $M'' \models Sensed[\sigma]$.

Finally let us define M''' by adding to M'' the predicate K and making it denote the identity relation on situations. Then $M''' \models Axioms \cup \{Sensed[\sigma]\}$. On the other hand since $M' \models \neg\phi(s')$, so does M''', a contradiction. ■

This means that if some objective property of the system is entailed, then it is also known and vice-versa.

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 to try to find a deterministic program that is guaranteed to be "executable" and constitutes a way to execute the program provided, in the sense that it always leads to terminating situations of the given program. Another way to look at this is that the deliberator tries to identify a "strategy" for reaching a final situation of the supplied 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 is required. 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. Note that our approach is similar to that of [Levesque, 1996]; 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:

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

$$
\begin{array}{l}
LEFDP(dp,s) \stackrel{\text{def}}{=} \\
\quad \mathbf{Know}(Final(dp,now),s) \lor \\
\quad \exists dp'.\mathbf{Know}(UTrans(dp,now,dp',now),s) \lor \\
\quad \exists dp',a.\mathbf{Know}(UTrans(dp,now,dp',do(a,now)),s)
\end{array}
$$

$$
\begin{array}{l}
UTrans(dp,s,dp',s') \stackrel{\text{def}}{=} \\
\quad Trans(dp,s,dp',s') \ \land \\
\quad \forall dp'',s''.Trans(dp,s,dp'',s'') \supset dp''=dp' \land s''=s'
\end{array}
$$

Thus to be an $EFDP$, a program must be such that all configurations reachable from the initial program and situation involve a locally epistemically feasible deterministic program ($LEFDP$). A program, is an $LEFDP$ in a situation if the agent knows that it is currently $Final$ or knows what unique transition (with or without an action) it can perform next.

Observe that an epistemically feasible deterministic program is not required to terminate. However, since the agent is guaranteed to know what to do next at every step in its execution, it follows that if it is entailed that the program can reach a final situation, then it can be successfully executed online whatever the sensing outcomes may be:

Theorem 2 *Let dp be such that* $Axioms \cup \{Sensed[\sigma]\} \models EFDP(dp, end[\sigma])$. *Then,* $Axioms \cup \{Sensed[\sigma]\} \models \exists s_f.Do(dp, end[\sigma], s_f)$ *if and only if all online executions of* (dp, σ) *are terminating.*

Proof Sketch: First of all we observe that dp is a deterministic program and its possible online executions from σ are completely determined by the sensing outcomes. We also observe that in each model there will be a single execution of dp, since the sensing outcomes are fully determined in the model.

$\Rightarrow$ If $Axioms \cup \{Sensed[\sigma]\} \models \exists s_f.Do(dp, end[\sigma], s_f)$ then in every model of $Axioms \cup \{Sensed[\sigma]\}$ the only execution of dp from $end[\sigma]$ terminates. Now since offline executions of dp terminate in all models and these models cover all possible sensing outcomes, an online execution must either successfully terminate or get stuck in an online configuration where neither $Final$ nor $Trans$ to some subsequent configuration is entailed. Suppose that there is such an online configuration (dp_i, σ_i) where the agent is stuck. Since in all models of $Axioms \cup \{Sensed[\sigma]\}$ with sensing outcomes as determined by σ_i, $LEFDP(dp_i, end[\sigma_i])$ holds, then either the agent knows that the remaining program is final or knows what the unique next transition is. By reflexivity of K, the agent is correct about this, so $Axioms \cup \{Sensed[\sigma_i]\}$ either entails that dp_i is final or entails that some next transition can be made. If the latter the next transition from (dp_i, σ_i) must be the same in all models of $Axioms \cup \{Sensed[\sigma_i]\}$. Indeed if there were models of $Axioms \cup \{Sensed[\sigma_i]\}$ that had different next transition for $(dp_i, end[\sigma_i])$ then there would be a model where there are distinct epistemic alternatives corresponding to these different models and so the agent would not know what the next transition is in this model. Hence, either way, the agent is not stuck in (dp_i, σ_i), thus getting a contradiction.

$\Leftarrow$ If an online execution of dp from σ terminates it means that the program dp, from $end[\sigma]$, terminates in all models of $Axioms \cup \{Sensed[\sigma]\}$ with the sensing outcomes as in the online execution. Since by hypothesis all online executions terminate, thus covering all possible sensing outcomes, then dp, from $end[\sigma]$, terminates in all models. ■

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:

$$\begin{array}{l} Trans(\Delta_e(p), s, dp', s') \equiv \\ \quad \exists dp. EFDP(dp, s) \wedge \\ \qquad \exists s_f. Trans(dp, s, dp', s') \wedge \\ \qquad\quad Do(dp', s', s_f) \wedge Do(p, s, s_f). \\ Final(\Delta_e(p), s) \equiv Final(p, s). \end{array}$$

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.[4] 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 3 *If* $Axioms \cup \{Sensed[\sigma]\} \models Trans(\Delta_e(p), end[\sigma], p', s')$, *then*

1. $Axioms \cup \{Sensed[\sigma]\} \models \exists s_f. Do(p, end[\sigma], s_f)$
2. $Axioms \cup \{Sensed[\sigma]\} \models \exists s_f. Do(\Delta_e(p), end[\sigma], s_f)$
3. *All online executions from* $(\Delta_e(p), \sigma)$ *terminate.*

Proof Sketch: 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 $Axioms \cup \{Sensed[\sigma]\} \models EFDP(dp, end[\sigma]) \wedge \exists s_f, p', s'. Trans(dp, end[\sigma], p', s') \wedge Do(p', s', s_f)$. The conditions of Theorem 2 are satisfied, thus we have that all online executions from (dp, σ) are terminating. Since these include all online executions from (p', σ') with $end[\sigma'] = s'$, all online executions from (p', σ') must also be terminating. Hence the thesis follows. ■

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

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[5] (see e.g., [Smith and Weld, 1998]), 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., [Smith et al., 1998]). 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 for instance, one is designing a deliberator/planner, one might want to only consider programs from these classes.

5.1 TREE PROGRAMS

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

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

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 (or trivial tests — these are introduced only for technical reasons). Whenever such a program is executable, it is also epistemically feasible — the agent always knows what to do next:

Theorem 4 *Let dpt be a tree program, i.e., dpt* $\in TREE$. *Then, for all histories* σ, *if* $Axioms \cup \{Sensed[\sigma]\} \models \exists s_f. Do(dpt, end[\sigma], s_f)$ *then* $Axioms \cup \{Sensed[\sigma]\} \models EFDP(dpt, end[\sigma])$.

Proof Sketch: By induction on the structure of dpt.

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

[5] 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 $Axioms \cup \{Sensed[\sigma]\} \models \exists s_f.Do(dpt, end[\sigma], s_f)$.

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

$$\begin{aligned}&Axioms \cup \{Sensed[\sigma]\} \models\\ &\quad \mathbf{Know}(Trans(a; dpt_1, now, dpt_1, do(a, now)), end[\sigma])\end{aligned}$$

It is also known that this is the only transition possible for $a; dpt_1$, So $Axioms \cup \{Sensed[\sigma]\} \models LEFDP(a; dpt_1, end[\sigma])$. Therefore,

$$Axioms \cup \{Sensed[\sigma]\} \models EFDP(a; dpt_1, end[\sigma]).$$

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 $\sigma_1 = \sigma \cdot (sense_\phi, 1)$. The initial assumption that $Axioms \cup \{Sensed[\sigma]\}$ entails $\exists s_f.Do(dpt, end[\sigma], s_f)$ implies that $Axioms \cup \{Sensed[\sigma_1]\} \models \exists s_f. Do(dpt_1, end[\sigma_1], s_f)$. Thus, by the induction hypothesis, we have $Axioms \cup \{Sensed[\sigma_1)]\} \models EFDP(dpt_1, end[\sigma_1])$. It follows that

$$\begin{aligned}&Axioms \cup \{Sensed[\sigma]\} \models\\ &\quad \phi(do(sense_\phi, end[\sigma])) \supset\\ &\qquad EFDP(dpt_1, do(sense_\phi, end[\sigma])).\end{aligned}$$

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

$$\begin{aligned}&Axioms \cup \{Sensed[\sigma]\} \models\\ &\quad \neg\phi(do(sense_\phi, end[\sigma])) \supset\\ &\qquad EFDP(dpt_2, do(sense_\phi, end[\sigma])).\end{aligned}$$

The initial assumption $Axioms \cup \{Sensed[\sigma]\} \models \exists s_f.Do(dpt, end[\sigma], s_f)$ also implies that $Axioms \cup \{Sensed[\sigma]\} \models Poss(sense_\phi, end[\sigma])$ and this must be known by Theorem 1, i.e., $Axioms \cup \{Sensed[\sigma]\} \models$ **Know**$(Poss(sense_\phi, now), end[\sigma])$. Thus, we have that

$$\begin{aligned}&Axioms \cup \{Sensed[\sigma]\} \models\\ &\quad \mathbf{Know}(Trans(dpt, now, \mathbf{if}\ \phi\ \mathbf{then}\ dpt_1\\ &\qquad \mathbf{else}\ dpt_2, do(sense_\phi, now)), end[\sigma]).\end{aligned}$$

It is also known that this is the only transition possible for dpt, so $Axioms \cup \{Sensed[\sigma]\} \models LEFDP(dpt, end[\sigma])$. Thus, $Axioms \cup \{Sensed[\sigma]\} \models EFDP(dpt, end[\sigma])$. ■

By Theorem 2, we also have that under the conditions of the above theorem, all online executions of (dpt, σ) are terminating. 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 [Peot and Smith, 1992, Smith et al., 1998].

Next, we show that 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 5 *For any program dp that is*

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

$$\begin{aligned}&Axioms \cup \{Sensed[\sigma]\} \models\\ &\quad \exists p', s', k.k \leq n \wedge Trans^k(dp, end[\sigma], p', s') \wedge\\ &\qquad Final(p', s')\end{aligned}$$

there exists a tree program $dpt \in TREE$ *such that* $Axioms \cup \{Sensed[\sigma]\} \models \forall s_f.Do(dp, end[\sigma], s_f) \equiv Do(dpt, end[\sigma], s_f)$.

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

- $m(dp, \sigma) = False?$ iff $Axioms \cup \{Sensed[\sigma]\}$ is inconsistent, otherwise
- $m(dp, \sigma) = nil$ iff $Axioms \cup \{Sensed[\sigma]\} \models Final(dp, end[\sigma])$, otherwise
- $m(dp, \sigma) = a; m(dp', \sigma \cdot (a, 1))$ iff

  $$\begin{aligned}&Axioms \cup \{Sensed[\sigma]\} \models\\ &\quad Trans(dp, end[\sigma], dp', do(a, end[\sigma])\end{aligned}$$

  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

$$Axioms \cup \{Sensed[\sigma]\} \models Trans(dp, end[\sigma], dp', do(sense_\phi, end[\sigma])$$

for some sensing action $sense_\phi$,
- $m(dp, \sigma) = True?; m(dp', \sigma)$ iff

$$Axioms \cup \{Sensed[\sigma]\} \models Trans(dp, end[\sigma], dp', end[\sigma]).$$

Let us show that

$$Axioms \cup \{Sensed[\sigma]\} \models Do(dp, end[\sigma], s_f) \equiv Do(m(dp, \sigma), end[\sigma], s_f).$$

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

- $Axioms \cup \{Sensed[\sigma]\} \models Final(dp, end[\sigma])$ iff $Axioms \cup \{Sensed[\sigma]\} \models Final(m(dp, \sigma), end[\sigma])$,
- for all dp', σ' if $Axioms \cup \{Sensed[\sigma]\} \models Trans(dp, end[\sigma], dp', end[\sigma'])$ with the set $Axioms \cup \{Sensed[\sigma']\}$ being consistent, then

$$Axioms \cup \{Sensed[\sigma]\} \models Trans(m(dp, \sigma), end[\sigma], m(dp', \sigma'), end[\sigma'])$$

and $[(dp', \sigma'), (m(dp', \sigma'), \sigma')]$,
- for all dp', σ' if $Axioms \cup \{Sensed[\sigma]\} \models Trans(m(dp, \sigma), end[\sigma], m(dp', \sigma'), end[\sigma'])$ with $Axioms \cup \{Sensed[\sigma']\}$ consistent, then

$$Axioms \cup \{Sensed[\sigma]\} \models Trans(dp, end[\sigma], dp', end[\sigma'])$$

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

Now, assume that $Axioms \cup \{Sensed[\sigma]\}$ entails $\exists s_f.Do(dp, end[\sigma], s_f)$. Then since dp is an $EFDP$, by Theorem 2 all online execution from (dp, σ) terminate. Hence since (dp, σ) and $(m(dp, \sigma), \sigma)$ are bisimilar, $(m(dp, \sigma), \sigma)$ has the same online executions (apart from the program appearing in the configurations).

Next, observe that given an online execution of (dp, σ) terminating in (dp_f, σ_f), in all models of $Axioms \cup \{Sensed[\sigma]\}$ with sensing outcomes as in σ_f both the program dp and $m(dp, \sigma)$ reach the same situation $end[\sigma_f]$. Since there are terminating online executions for all possible sensing outcomes, the thesis follows. ■

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 6 *Let dpl be a linear program, i.e., dpl* $\in LINE$*. Then, for all histories* σ*, if* $Axioms \cup \{Sensed[\sigma]\} \models \exists s_f.Do(dpl, end[\sigma], s_f)$ *then* $Axioms \cup \{Sensed[\sigma]\} \models EFDP(dpl, end[\sigma])$*.*

Proof Sketch: This is a corollary of Theorem 4 for tree programs. Since linear programs are tree programs, the thesis follows immediately from this theorem. ■

By Theorem 2, we also have that under the conditions of the above theorem, all online executions of (dpl, σ) are terminating. Since the agent may have incomplete knowledge, 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 [Smith and Weld, 1998].

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

Theorem 7 *For any dp that does not include sensing actions, such that* $Axioms \cup \{Sensed[\sigma]\} \models EFDP(dp, end[\sigma])$*, there exists a linear program dpl such that* $Axioms \cup \{Sensed[\sigma]\} \models \forall s_f.Do(dp, end[\sigma], s_f) \equiv Do(dpl, end[\sigma], s_f)$*.*

Proof Sketch: We show this using the same approach as for Theorem 5 for tree programs. Since dp cannot contain sensing actions, the construction method used in the proof of Theorem 5 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):

$$\begin{aligned}&Trans(\Delta_l(p), s, dpl', s') \equiv\\&\quad \exists dpl.LINE(dpl) \land\\&\qquad \exists s_f.Trans(dpl, s, dpl', s') \ \land\\&\qquad\quad Do(dpl', s', s_f) \land Do(p, s, s_f).\end{aligned}$$

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 8 *There exists a situation s_f such that*

$$Axioms \cup \{Sensed[\sigma]\} \models Do(p, end[\sigma], s_f)$$

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

$$Axioms \cup \{Sensed[\sigma]\} \models Trans(\Delta_l(p), end[\sigma], dpl, s')$$

Proof Sketch: $\Leftarrow$ By hypothesis there exists a dpl that is a $LINE$. If $s' = end[\sigma]$ and 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 $Axioms \cup \{Sensed[\sigma]\} \models Do(p, end[\sigma], s_f)$.

$\Rightarrow$ If for some s_f we have $Axioms \cup \{Sensed[\sigma]\} \models Do(p, end[\sigma], s_f)$ 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 the linear program can be simply nil. ■

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.

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):-   /* A is not   */
  trans(P,H,P1,[(A,1)|H]), /* a sensing  */
  buildLine(P1,DPL,[(A,1)|H]). /* action */
```

Instead of situations, this code uses histories, which are essentially lists of pairs of actions and sensing outcomes since the initial situation. The `buildLine(P,DPL,H)` predicate basically looks for a sequence of transitions that the program can perform and that that is guaranteed to lead to a final configuration without performing sensing (sensing outcomes for non-sensing actions are assumed to be 1). This approach to implementing deliberation is essentially that used in [De Giacomo et al., 1998, Lespérance and Ng, 2000, De Giacomo et al., 2001], 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]):- inconsistent(H) ;
     senses(A,F), holds(neg(F),H).
inconsistent([(A,0)|H]):- inconsistent(H) ;
     senses(A,F), holds(F,H).
```

A transition is performed on a program `search_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 [De Giacomo and Levesque, 1999a]. Both of the above implementations are sound but not complete.[6]

[6]The incompleteness comes from the fact that they stick to

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 [Lespérance and Ng, 2000] 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 [De Giacomo et al., 1998], 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:

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

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. 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.

$$\begin{array}{l} perturbed(mnt(dp, s_e, p_i, s_i), s) \equiv \\ \quad s_e \neq s \land \neg\exists s_f.[Do(dp, s, s_f) \land Do(p_i \parallel p_{ex}, s_i, s_f)] \end{array}$$

Notice that we make use of the special program $p_{ex} \stackrel{\text{def}}{=} (\pi a. Exo(a)?; a)^*$, see [De Giacomo et al., 2000], to allow for a legal sequence of exogenous actions. 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:

$$\begin{array}{l} recover(mnt(dp, s_e, p_i, s_i), s, dp_r) \equiv \\ \quad \exists p_i'. Trans^*(p_i \parallel p_{ex}, s_i, p_i' \parallel p_{ex}, s) \;\land \\ \quad EFDP(dp_r, s) \;\land\; \exists s_f. Do(dp_r, s, s_f) \land Do(p_i', s, s_f). \end{array}$$

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

---

the form of the 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 $True?; True?; a$ is a strategy for executing it, but the implementations fail to find it.

so far, either by the deliberation block or not, since the beginning of the deliberation block.

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

$$
\begin{aligned}
&Trans(mnt(dp, s_e, p_i, s_i), s, p', s') \equiv \\
&\quad [\neg perturbed(mnt(dp, s_e, p_i, s_i), s) \ \wedge \\
&\qquad \exists dp'.Trans(dp, s, dp', s') \ \wedge \\
&\qquad\quad p' = mnt(dp', s', p_i, s_i)] \vee \\
&\quad [perturbed(mnt(dp, s_e, p_i, s_i), s) \ \wedge \\
&\qquad \exists dp_r.recover(mnt(dp, s_e, p_i, s_i), s, dp_r) \ \wedge \\
&\qquad\quad \exists dp'.Trans(dp_r, s, dp', s') \ \wedge \\
&\qquad\qquad p' = mnt(dp', s', p_i, s_i)]
\end{aligned}
$$

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

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.

Summarizing, 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 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 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 the notion of epistemically feasible plan, e.g. Moore [1985], Davis [1994], Lespérance et al. [2000], Levesque [1996], and our accounts builds on this. One of its distinguishing features is that it is integrated with the transition system semantics of our programming language. In Lespérance [2001], a similar approach is used to formalize a notion of epistemic feasibility for multiagent system specifications. In McIlraith and Son [2001], a notion of "self-sufficient program" very similar to $EFDP$s is formalized; but this account is more sensitive to the syntax of the program than ours.

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.[7] 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 impractical without advice from the programmer as to what conditions the plan should branch on [Lakemeyer, 1999, Thielscher, 2001]. In [Sardiña, 2001], 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. There are also more general theories of sensing, such as that of [De Giacomo and Levesque, 1999b] which deals with online sensors that always provide values and situations where the law of inertia is not always applicable. In [De Giacomo et al., 2001], a search operator for such theories is devel-

[7] One can introduce non-binary sensing actions in our framework as in [Scherl and Levesque, 1993].

oped. 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.

References

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

Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121:109–169, 2000.

Giuseppe De Giacomo and Hector J. Levesque. An incremental interpreter for high-level programs with sensing. In Hector J. Levesque and Fiora Pirri, editors, *Logical Foundations for Cognitive Agents*, pages 86–102. Springer-Verlag, 1999a.

Giuseppe De Giacomo and Hector J. Levesque. Progression and regression using sensors. In *Proc. of IJCAI-99*, pages 160–165, 1999b.

Giuseppe De Giacomo, Hector J. Levesque, and Sebastian Sardiña. Incremental execution of guarded theories. *ACM Transactions on Computational Logic*, 2(4):495–525, 2001.

Giuseppe De Giacomo, Raymond Reiter, and Mikhail Soutchanski. Execution monitoring of high-level robot programs. In *Proc. of KR-98*, pages 453–465, 1998.

Gerhard Lakemeyer. On sensing and off-line interpreting in Golog. In H. J. Levesque and F. Pirri, editors, *Logical Foundations for Cognitive Agents*, pages 173–187. Springer-Verlag, 1999.

Gerhard Lakemeyer and Hector J. Levesque. AOL: A logic of acting, sensing, knowing, and only-knowing. In *Proc. of KR-98*, pages 316–327, 1998.

Yves Lespérance. On the epistemic feasibility of plans in multiagent systems specifications. In J.-J. Meyer, M. Tambe, and D. Pynadath, editors, *Intelligent Agents VIII, Agent Theories, Architectures, and Languages, 8th Intl. Workshop, ATAL-2001, Seattle, WA, USA, Aug. 1-3, 2001, Proc.*, LNAI. Springer, 2001.

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

Yves Lespérance and Ho-Kong Ng. Integrating planning into reactive high-level robot programs. In *Proc. of the Second International Cognitive Robotics Workshop*, pages 49–54, 2000.

Hector J. Levesque. What is planning in the presence of sensing? In *Proc. of AAAI-96*, pages 1139–1146, 1996.

Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(59–84), 1997.

John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intellig ence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, 1979.

Sheila McIlraith and Tran Cao Son. Adapting Golog for programming the semantic web. In *Working Notes of the 5th Int. Symposium on Logical Formalizations of Commonsense Reasoning*, pages 195–202, 2001.

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

Mark A. Peot and David E. Smith. Conditional nonlinear planning. In *Proc. of the First International Conference on AI Planning Systems*, pages 189–197, 1992.

Gordon Plotkin. A structural approach to operational semantics. Technical Report DAIMI-FN-19, Computer Science Dept., Aarhus University, Denmark, 1981.

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

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

Raymond Reiter. On knowledge-based programming with sensing in the situation calculus. *ACM Transactions on Computational Logic*, 2(4):433–457, 2001b.

Sebastian Sardiña. Local conditional high-level robot programs. In *Proc. of LPAR-01*, volume 2250 of *LNAI*, pages 110–124, 2001.

Richard B. Scherl and Hector J. Levesque. The frame problem and knowledge-producing actions. In *Proc. of AAAI-93*, pages 689–695. AAAI Press/The MIT Press, 1993.

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

David E. Smith and Daniel S. Weld. Conformant graphplan. In *Proc. of AAAI-98*, pages 889–896, 1998.

Michael Thielscher. Inferring implicit state knowledge and plans with sensing actions. In *Proc. of KI-01*, volume 2174 of *LNAI*, pages 366–380. Springer, 2001.