On Ability to Autonomously Execute Agent Programs with Sensing

Sebastian Sardina[1] and **Giuseppe De Giacomo**[2] and **Yves Lespérance**[3] and **Hector J. Levesque**[4]

Abstract. Most existing work in agent programming assumes an execution model where an agent has a knowledge base (KB) about the current state of the world, and makes decisions about what to do in terms of what is entailed or consistent with this KB. Planning then involves looking ahead and gauging what would be consistent or entailed at various stages by possible future KBs. We show that in the presence of sensing, such a model does not always work properly, and propose an alternative that does. We then discuss how this affects agent programming language design/semantics.

1 INTRODUCTION

There has been considerable work on formal models of deliberation/planning under incomplete information, where an agent can perform sensing actions to acquire additional information. This problem is very important in agent applications such as web information retrieval/management. However, much of the previous work on formal models of deliberation—i.e., models of knowing how, ability, epistemic feasibility, executabiliy, etc. such as [14, 4, 9, 11, 19]—has been set in epistemic logic-based frameworks and is hard to relate to work on agent programming languages (e.g. 3APL [8], AgentSpeak(L) [17]). In this paper, we develop new non-epistemic formalizations of deliberation that are much closer and easier to relate to standard agent programming language semantics based on transition systems.

When doing deliberation/planning under incomplete information, one typically searches over a set of states, each of which is associated with a knowledge base (KB) or theory that represents what is known in the state. To evaluate tests in the program and to determine what transitions/actions are possible, one looks at what is *entailed* by the current KB. To allow for future sensing results, one looks at which of these are *consistent* with the current KB. We call this type of approach to deliberation "entailment and consistency-based" (EC-based). In this paper, we argue that EC-based approaches do not always work, and propose an alternative. Our accounts are formalized within the situation calculus and use a simple programming language based on ConGolog [6] to specify agent programs as described in Section 2, but we claim that the results generalize to most proposed agent programming languages/frameworks. We point out that this paper is mainly concerned with the semantics of the deliberation process and not much with the actual algorithms implementing this process.

We initially focus on deterministic programs/plans and how to formalize when an agent knows how to execute them. For such deterministic programs, what this amounts to is ensuring that the agent will always know what the next step to perform is, and no matter what sensing results are obtained, the agent will eventually get to the point where it knows it can terminate. In Sections 3 and 4, we develop a simple EC-based account of knowing how (*KHow*$_{EC}$). We show that this account gives the wrong results on a simple example involving indefinite iteration. Then, we show that whenever this account says that a deliberation/planning problem is solvable, there is a *conditional plan* (a finite tree program without loops) that is a solution. It follows that this account is limited to problems where the total number of steps needed can be bounded in advance. We claim that this limitation is not specific to the simple account and applies to all EC-based accounts of deliberation.

The source of the problem with the EC-based account is the use of local consistency checks to determine which sensing results are possible. This does not correctly distinguish between the models that satisfy the overall domain specification (for which the plan must work) and those that do not. To get a correct account of deliberation, one must take into account what is true in different models of the domain together with what is true in all of them (what is entailed). In Section 5, we develop such an entailment and truth-based account (*KHow*$_{ET}$), argue that it intuitively does the right thing, and show how it correctly handles our test examples.

We end by reviewing the paper's contributions, discussing the lessons for agent programming language design, and sketching other related results that we have but were left out due to lack of space.

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 [7, 6]. The starting point in the definition is the situation calculus [12]. 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. There is a special predicate $Poss(a, s)$ used to state that action a is executable in situation s. We assume that actions

[1] Dept. of Computer Science, University of Toronto, Toronto, Canada, email: ssardina@cs.toronto.edu
[2] Dip. Informatica e Sistemistica, Univer. di Roma "La Sapienza", Roma, Italy, email: degiacomo@dis.uniroma1.it
[3] Dept. of Computer Science, York University, Toronto, Canada, email: lesperan@cs.yorku.ca
[4] Dept. of Computer Science, University of Toronto, Toronto, Canada, email: hector@cs.toronto.edu

return binary sensing results, and we use the predicate $SF(a, s)$ to characterize what the action tells the agent about its environment. For example, the axiom

$$SF(senseDoor(d), s) \equiv Open(d, s)$$

states that the action $senseDoor(d)$ tells the agent whether the door is open in situation s. For actions with no useful sensing information, we write $SF(a, s) \equiv True$.

Within this language, we can formulate domain theories which describe how the world changes as a result of the available actions. Here, we use basic action theories [18] of the following form:

- A set of foundational, domain independent axioms for situations Σ as in [18].
- Axioms describing the initial situation, S_0.
- Action precondition axioms, one for each primitive action a, characterizing $Poss(a, s)$.
- Successor state axioms for fluents of the form
  $F(\vec{x}, do(a, s)) \equiv \gamma(\vec{x}, a, s)$
  providing the usual solution to the frame problem.
- Sensed fluent axioms, as described above, of the form
  $SF(A(\vec{x}), s) \equiv \phi(\vec{x}, s)$
- Unique names axioms for the primitive actions.

To describe a run of a program which includes both actions and their sensing results, we use the notion of a *history*, i.e., a sequence of pairs (a, μ) where a is a primitive action and μ is 1 or 0, a sensing result. Intuitively, the history $\sigma = (a_1, \mu_1) \cdot \ldots \cdot (a_n, \mu_n)$ is one where actions $a_1, \ldots, a_n$ happen starting in some initial situation, and each action a_i returns sensing value μ_i. We use $end[\sigma]$ to denote the situation term corresponding to the history σ, and $Sensed[\sigma]$ to denote the formula of the situation calculus stating all sensing results of the history σ. Formally,

$end[\epsilon] = S_0$, where ϵ is the empty history; and
$end[\sigma \cdot (a, \mu)] = do(a, end[\sigma])$.

$Sensed[\epsilon] = True$;
$Sensed[\sigma \cdot (a, 1)] = Sensed[\sigma] \wedge SF(a, end[\sigma])$;
$Sensed[\sigma \cdot (a, 0)] = Sensed[\sigma] \wedge \neg SF(a, end[\sigma])$.

Next we turn to programs. We consider a very simple deterministic language with the following constructs:

a	primitive action
$\delta_1; \delta_2$	sequence
if ϕ **then** δ_1 **else** δ_2 **endIf**	conditional
while ϕ **do** δ **endWhile**	while loop

This is a small subset of ConGolog [6] and we use its single step transition semantics in the style of [16]. This semantics introduces two special predicates $Trans$ and $Final$: $Trans(\delta, s, \delta', s')$ means that by executing program δ in situation s, one can get to situation s' in one elementary step with the program δ' remaining to be executed; $Final(\delta, s)$ means that program δ may successfully terminate in situation s.

Offline executions of programs, which are the kind of executions originally proposed for Golog and ConGolog [10, 6], are characterized using the $Do(\delta, s, s')$ predicate, which means that there is an execution of program δ that starts in situation s and terminates in situation s'. This holds if there is a sequence of legal transitions from the initial configuration up to a final configuration:

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

where $Trans^*$ is the reflexive transitive closure of $Trans$. An offline execution of δ from s is a sequence of actions $a, \ldots, a_n$ such that: $\mathcal{D} \cup \mathcal{C} \models Do(\delta, s, do(a_n, \ldots, do(a_1, s) \ldots))$, where $\mathcal{D}$ is an action theory as mentioned above, and $\mathcal{C}$ is a set of axioms defining the predicates $Trans$ and $Final$ and the encoding of programs as first-order terms [6].

Observe that an offline executor has no access to sensing results, available only at runtime. IndiGolog, an extension of ConGolog to deal with online executions with sensing, is proposed in [7]. The semantics defines an *online execution* of a program δ starting from a history σ. We say that a configuration (δ, σ) *may evolve* to configuration (δ', σ') *w.r.t. a model M* (relative to an underlying theory of action $\mathcal{D}$) iff [5] (i) M is a model of $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma]\}$, and (ii)

$$\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma_i]\} \models Trans(\delta, end[\sigma], \delta', end[\sigma'])$$

and (iii)

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

The model M above is only used to represent a possible environment and, hence, it is just used to generate the sensing results of the corresponding environment. Finally, we say that a configuration (δ, σ) is *final* whenever

$$\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma]\} \models Final(\delta, end[\sigma]).$$

Using these two concepts of configuration evolution and final configurations, one can define various notions of online, incremental, executions of programs as a sequence of legal configuration evolutions, possibly terminating in a final configuration.

3 DELIBERATION: EC-BASED ACCOUNT

Perhaps the first approach to come to mind for defining when an agent knows how/is able to execute a deterministic program δ in a history σ goes as follows: the agent must always know what the next action prescribed by the program is and be able to perform it such that no matter what sensing output is obtained as a result of doing the action, she can continue this process with what remains of the program and, eventually, reach a configuration where she knows she can legally terminate. We can formalize this idea as follows.

We define $KHow_{EC}(\delta, \sigma)$ to be the smallest relation $\mathcal{R}(\delta, \sigma)$ such that:

(E1) if (δ, σ) is *final*, then $\mathcal{R}(\delta, \sigma)$;
(E2) if there exists an action a such that (δ, σ) *may evolve* to configuration $(\delta', \sigma \cdot (a, \mu))$ for some δ' and μ *w.r.t. some model* of theory $\mathcal{D}$, and $\mathcal{R}(\delta', \sigma \cdot (a, \mu_i))$ holds for *every* configuration $(\delta', \sigma \cdot (a, \mu_i))$ such that (δ, σ) *may evolve* to *w.r.t. some model* M_i of theory $\mathcal{D}$, then $\mathcal{R}(\delta, \sigma)$.

The first condition states that every terminating configuration is in the relation $KHow_{EC}$. The second condition states that if a configuration performs an action transition and for every consistent sensing result, the resulting configuration is in $KHow_{EC}$, then this configuration is also in $KHow_{EC}$.

[5] This definition is more general than the one in [7], where the sensing results were assumed to come from the actual environment rather than from a model (a model can represent any possible environment).

Note that, here, the agent's lack of complete knowledge in a history σ is modeled by the theory $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma]\}$ being incomplete and having many different models. *KHow*$_{EC}$ uses entailment to ensure that the information available is sufficient to determine which transition should be performed next. *KHow*$_{EC}$ uses consistency to determine which sensing results can occur, for which the agent needs to have a subplan that leads to a final configuration. Due to this, we say that *KHow*$_{EC}$ is an *entailment and consistency-based* (EC-based) account of knowing how.

This EC-based account of knowing how seems quite intuitive and attractive. However it has a fundamental limitation: it fails on programs involving indefinite iteration. The following simple example from [9] shows the problem.

Consider a situation in which an agent wants to cut down a tree. Assume that the agent has a primitive action *chop* to chop at the tree, and also assume that she can always find out whether the tree is down by doing the (binary) sensing action $look$. If the sensing result is 1, then the tree is down; otherwise the tree remains up. There is also a fluent $RemainingChops(s)$, which we assume ranges over the natural numbers $\mathbb{N}$ and whose value is unknown to the agent, and which is meant to represent how many *chop* actions are still required in s to bring the tree down. The agent's goal is to bring the tree down, i.e., bringing about a situation s such that $Down(s)$ holds, where

$$Down(s) \stackrel{\text{def}}{=} RemainingChops(s) = 0$$

The action theory $\mathcal{D}_{tc}$ is the union of:

1. The foundational axioms for situations Σ.
2. $\mathcal{D}_{una} = \{chop \neq look\}$.
3. $\mathcal{D}_{ss}$ contains the following successor state axiom:

$$\begin{array}{l} RemainingChops(do(a,s)) = n \ \equiv \\ \quad (a = chop \wedge RemainingChops(s) = n+1) \ \vee \\ \quad (a \neq chop \wedge RemainingChops(s) = n). \end{array}$$

4. $\mathcal{D}_{ap}$ contains the following two precondition axioms:

$$\begin{array}{l} Poss(chop, s) \equiv (RemainingChops > 0), \\ Poss(look, s) \equiv True. \end{array}$$

5. $\mathcal{D}_{S_0} = \{RemainingChops(S_0) \neq 0\}$.
6. $\mathcal{D}_{sf}$ contains the following two sensing axioms:

$$\begin{array}{l} SF(chop, s) \equiv True, \\ SF(look, s) \equiv (RemainingChops(s) = 0). \end{array}$$

Notice that sentence $\exists n.RemainingChops(S_0) = n$ (where the variable n ranges over $\mathbb{N}$) is trivially entailed by this theory so "infinitely" hard tree trunks are ruled out. Nonetheless, the theory *does not* entail the sentence $RemainingChops(S_0) < k$ for any constant $k \in \mathbb{N}$. Hence, there exists some $n \in \mathbb{N}$, though unknown and unbounded, such that the tree will fall after n chops. Because of this, intuitively, we should have that the agent can bring the tree down, since if the agent keeps chopping, the tree will eventually come down, and the agent can find out whether it has come down by looking. Thus, for the program

$$\delta_{tc} = \textbf{while } \neg Down \textbf{ do } chop; look \textbf{ endWhile}$$

we should have that *KHow*$_{EC}(\delta_{tc}, \epsilon)$ (note that δ_{tc} is deterministic). However, this is not the case:

Theorem 1 *Let δ_{tc} be the above program to bring the tree down. Then, for all $k \in \mathbb{N}$, KHow$_{EC}(\delta_{tc}, [(chop, 1) \cdot (look, 0)]^k)$ does not hold. In particular, when $k = 0$, KHow$_{EC}(\delta_{tc}, \epsilon)$ does not hold.*

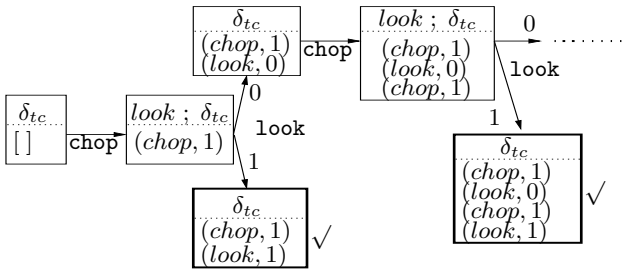

Figure 1. Online execution tree of program δ_{tc}. Each box represents a configuration with the remaining program at the top and the current history at the bottom. Terminating configurations are marked with a check sign.

Thus, the simple EC-based formalization of knowing how gives the wrong result for this example. Why is this so? Intuitively, it is easy to check that if the agent knows how (to execute) the initial configuration, i.e., *KHow*$_{EC}(\delta_{tc}, \epsilon)$ holds, then she knows-how (to execute) *every* possible finite evolution of it, i.e., for all $j \in \mathbb{N}$, *KHow*$_{EC}(\delta_{tc}, [(chop, 1) \cdot (look, 0)]^j)$ and *KHow*$_{EC}((look; \delta_{tc}), [(chop, 1) \cdot (look, 0)]^j \cdot (chop, 1))$. Now consider the hypothetical scenario in which an agent keeps chopping and looking forever, always seeing that the tree is not down. There is no model of $\mathcal{D}_{tc}$ where δ_{tc} yields this scenario, as the tree is guaranteed to come down after a finite number of chops. However, by the above, we see that *KHow*$_{EC}$ is, in some way, taking this case into account in determining whether the agent knows how to execute δ_{tc} (see Figure 1). This happens because every finite prefix of this never-ending execution is indeed *consistent* with $\mathcal{D}_{tc}$. The problem is that the set of *all* of them together is not. This is why *KHow*$_{EC}$ fails, which can also be viewed as a lack of compactness issue. In the next section, we show that *KHow*$_{EC}$'s failure on the tree chopping example is due to a general limitation of the *KHow*$_{EC}$ formalization. Note that Moore's original account of ability [14] is closely related to *KHow*$_{EC}$ and also fails on the tree chopping example [9].

4 *KHow*$_{EC}$ ONLY HANDLES BOUNDED PROBLEMS

In this section, we show that whenever *KHow*$_{EC}(\delta, \sigma)$ holds for some program δ and history σ, there is simple kind of conditional plan, what we call a $TREE$ program, that can be followed to execute δ in σ. Since for $TREE$ programs (and conditional plans), the number of steps they perform can be bounded in advance (there are no loops), it follows that *KHow*$_{EC}$ will never be satisfied for programs whose execution cannot be bounded in advance. Since there are many such programs (for instance, the one for the tree chopping example), it follows that *KHow*$_{EC}$ is fundamentally limited as a formalization of knowing how and can only be used in contexts where attention can be restricted to bounded strategies. As in [19], we define the class of *(sense-branch) tree programs* $TREE$ with the following BNF rule:

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

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. Thus as shown in [19], whenever a $TREE$ program is executable, it is also epistemically feasible, i.e.,

the agent can execute it without ever getting stuck not knowing what transition to perform next. $TREE$ programs are clearly deterministic.

Let us define a relation *KHowBy*$_{EC}$: $Program \times History \times TREE$. The relation is intended to associate a program δ and history σ for which *KHow*$_{EC}$ holds with some $TREE$ program(s) that can be used as a strategy for successfully executing δ in σ.

We define *KHowBy*$_{EC}(\delta, \sigma, \delta^{tp})$ to be the least relation $\mathcal{R}(\delta, \sigma, \delta^{tp})$ such that:

(A) if (δ, σ) is *final*, then $\mathcal{R}(\delta, \sigma, nil)$;
(B) if (δ, σ) *may evolve* to configurations $(\delta', \sigma \cdot (a, 1))$ and $(\delta', \sigma \cdot (a, 0))$ w.r.t. some models M_1 and M_2, respectively, of theory $\mathcal{D}$, and there exist δ_1^{tp} and δ_0^{tp} such that $\mathcal{R}(\delta', \sigma \cdot (a, 1), \delta_1^{tp})$ and $\mathcal{R}(\delta', \sigma \cdot (a, 0), \delta_0^{tp})$ hold, then $\mathcal{R}(\delta, \sigma, (a; \textbf{if } \phi \textbf{ then } \delta_1^{tp} \textbf{ else } \delta_0^{tp} \textbf{ endIf}))$ where ϕ is the condition on the right hand side of the sensed fluent axiom for a (i.e., action a senses the truth value of formula ϕ).
(C) if there exists an action a and a program δ' for which (δ, σ) *may evolve* to configuration $(\delta', \sigma \cdot (a, \mu))$ *only* for some *unique* sensing outcome μ and some model M of theory $\mathcal{D}$, and there exist δ'' such that $\mathcal{R}(\delta', \sigma \cdot (a, \mu), \delta'')$ holds, then $\mathcal{R}(\delta, \sigma, (a; \delta''))$.

Condition (A) deals with the simple case of a terminating configuration; condition (B) handles the case in which the current configuration can perform a step with some (sensing) action a and where both sensing outcomes 1 and 0 are eventually possible/consistent; and condition (C) deals with the simpler cases of a non-sensing action step and a sensing action step for which there is only one consistent sensing outcome.

It is possible to show that whenever *KHowBy*$_{EC}(\delta, \sigma, \delta^{tp})$ holds, then *KHow*$_{EC}(\delta, \sigma)$ and *KHow*$_{EC}(\delta^{dp}, \sigma)$ hold, and the $TREE$ program δ^{tp} is guaranteed to terminate in a $Final$ situation of the given program δ (in all models).

Theorem 2 *For all programs δ, histories σ, and programs δ^{tp}, if* *KHowBy*$_{EC}(\delta, \sigma, \delta^{tp})$ *then we have that*

- *KHow*$_{EC}(\delta, \sigma)$ *and* *KHow*$_{EC}(\delta^{dp}, \sigma)$ *hold; and*
- *There is a common execution for δ^{tp} and δ from $end[\sigma]$:*

$$\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma]\} \models \exists s. Do(\delta^{tp}, end[\sigma], s) \wedge Do(\delta, end[\sigma], s).$$

In addition, every configuration captured in *KHow*$_{EC}$ can be executed using a $TREE$ program.

Theorem 3 *For all programs δ and histories σ, if* *KHow*$_{EC}(\delta, \sigma)$, *then there exists a program δ^{tp} such that* *KHowBy*$_{EC}(\delta, \sigma, \delta^{tp})$.

Since the number of steps a $TREE$ program performs can be bounded in advance, it follows that *KHow*$_{EC}$ will never hold for programs/problems that are solvable, but whose execution requires a number of steps that cannot be bounded in advance, as it is the case with the program in the tree chopping example. Thus *KHow*$_{EC}$ is severely restricted as an account of knowing how; it can only be complete when all possible strategies are bounded.

5 DELIBERATION: ET-BASED ACCOUNT

We saw earlier that the reason *KHow*$_{EC}$ failed on the tree chopping example was that it required the agent to have a choice of action that guaranteed reaching a final configuration even for histories that were inconsistent with the domain specification such as the infinite history corresponding to the hypothetical scenario described in the previous paragraph. There was a branch in the configuration tree that corresponded to that history. This occurred because "local consistency" was used to construct the configuration tree. The consistency check kept switching which model of $\mathcal{D} \cup \mathcal{C}$ (which may be thought as representing the environment) was used to generate the next sensing result, postponing the observation that the tree had come down forever. But in the real world, sensing results come from a fixed environment (even if we don't know which environment this is). It seems reasonable that we could correct the problem by fixing the model of $\mathcal{D} \cup \mathcal{C}$ used in generating possible configurations in our formalization of knowing how. This is what we will now do.

We define when an agent knows how to execute a program δ in a history σ and a model M (which represents the environment), *KHowInM*(δ, σ, M), as the smallest relation $\mathcal{R}(\delta, \sigma)$ such that:

(T1) if (δ, σ) is *final*, then $\mathcal{R}(\delta, \sigma)$;
(T2) if (δ, σ) *may evolve* to $(\delta', \sigma \cdot (a, \mu))$ w.r.t. M and $\mathcal{R}(\delta', \sigma \cdot (a, \mu))$, then $\mathcal{R}(\delta, \sigma)$;

The only difference between this and *KHow*$_{EC}$ is that the sensing results come from the fixed model M. Given this, we obtain the following formalization of when an agent knows how to execute a program δ in a history σ:

KHow$_{ET}(\delta, \sigma)$
iff
for every model M such that $M \models \mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma]\}$,
KHowInM(δ, σ, M) holds.

We call this type of formalization *entailment and truth-based*, since it uses entailment to ensure that the agent knows what transitions she can do, and *truth in a model* to obtain possible sensing results.

We claim that *KHow*$_{ET}$ is actually correct for programs δ that are deterministic. For instance, it handles the tree chopping example correctly:

Proposition 4 *KHow*$_{ET}(\delta_{tc}, \epsilon)$ *holds w.r.t. theory $\mathcal{D}_{tc}$.*

Furthermore, *KHow*$_{ET}$ is strictly more general than *KHow*$_{EC}$. Formally,

Theorem 5 *For any background theory $\mathcal{D}$ and any configuration (δ, σ), if* *KHow*$_{EC}(\delta, \sigma)$ *holds, then* *KHow*$_{ET}(\delta, \sigma)$. *Moreover, there is a background theory $\mathcal{D}^*$ and a configuration (δ^*, σ^*) such that* *KHow*$_{ET}(\delta^*, \sigma^*)$ *holds, but* *KHow*$_{EC}(\delta^*, \sigma^*)$ *does not.*

6 DISCUSSION AND CONCLUSION

In an extended version of this paper, we show how the notion of ability to achieve a goal can be defined in terms of our notions of knowing how to execute a deterministic program. We observe that an EC-based definition of ability inherits the limitations of the EC-based definition of knowing how. Then, we examine knowing how to execute a *nondeterministic* program. We consider two ways of interpreting this: one (angelic knowing how) where the agent does planning/lookahead to make the right choices, and another (demonic knowing how) where the agent makes choices arbitrarily. We discuss EC-based and ET-based formalizations of these notions. Finally, we show how angelic knowing how can be used to specify a powerful planning construct in the IndiGolog agent programming language.

In this paper, we have looked at how to formalize when an agent knows how to execute a program, which in the general case, when the program is nondeterministic and the agent does lookahead and reasons about possible execution strategies, subsumes ability to achieve a goal. First, we have shown that an intuitively reasonable entailment and consistency-based approach to formalizing knowing how, $KHow_{EC}$, fails on examples like our tree chopping case and that, in fact, $KHow_{EC}$ can only handle problems that can be solved in a bounded number of steps, i.e. without indefinite iteration.

The problems of accounts like $KHow_{EC}$ when they are formalized in epistemic logic, such as Moore's [14], had been pointed out before, for instance in [9]. However, the reasons for the problems were not well understood. The results we have presented clarify the source of the problems and show what is needed for their solution. A simple meta-theoretic approach to knowing how fails; one needs to take entailment and truth into account together. (Even if we use a more powerful logical language with an knowledge operator, knowledge and truth must be considered together.)

Our non-epistemic accounts of knowing how are easily related to models of agent programming language semantics and our results have important implications for this area. While most work on agent programming languages (e.g. 3APL [8], AgentSpeak(L) [17], etc.) has focused on reactive execution, sensing is acknowledged to be important and there has been interest in providing mechanisms for run-time planning/deliberation. The semantics of such languages are usually specified as a transition system. For instance in 3APL, configurations are pairs involving a program and a belief base, and a transition relation over such pairs is defined by a set of rules. Evaluating program tests is done by checking whether they are entailed by the belief base. Checking action preconditions is done by querying the agent's belief base update relation, which would typically involve determining entailments over the belief base — the 3APL semantics abstracts over the details of this. Sensing is not dealt with explicitly, although one can suppose that it could be handled by simply updating the belief base (AgentSpeak(L) has events for this kind of thing).

As mentioned, most work in the area only deals with on-line reactive execution, where no deliberation/lookahead is performed; this type of execution just involves repeatedly selecting some transition allowed in the current configuration and performing it. However, one natural view is that *deliberation can simply be taken as a different control regime involving search over the agent program's transition tree*. In this view, a deliberating interpreter could first lookahead and search the program's transition tree to find a sequence of transitions that leads to successful termination and later execute this sequence. This assumes that the agent can chose among all alternative transitions. Clearly, in the presence of sensing, this idea needs to be refined. One must find more than just a path to a final configuration in the transition tree; one needs to find some sort of conditional plan or subtree where the agent has chosen some transition among those allowed, but must have branches for all possible sensing results. The natural way of determining which sensing results are possible is checking their consistency with the current belief base. Thus, what is considered here is essentially an EC-based approach.

Also in work on planning under incomplete information, e.g. [3, 15, 5], a similar sort of setting is typically used, and finding a plan involves searching a (finite) space of knowledge states that are compatible with the planner's knowledge. The underlying models of all these planners are meant to represent only the *current* possible states of the environment, which, in turn, are updated upon the hypothetical execution of an action at planning time. We use models that are dynamic in the sense that they represent the potential responses of the environment for *any* future state. In that way, then, what the above planners are doing is deliberation in the style of $KHow_{EC}$. An interesting case arises with answer set planning/programming, e.g. [2, 20, 21]. There, plans are found by inspecting all models of an underlying logic program and, hence, they seem, in principle, to be more in the lines of our ET-based approach to deliberation. Nonetheless, all these approaches are eventually restricted to propositional languages and, as a result, only bounded problems can be expressed.

Our results show that the ET-based view of deliberation is fundamentally flawed when sensing is present. It produces an account that only handles problems that can be solved in a bounded number of actions. As an approach to implementing deliberation, this may be perfectly fine. But as a semantics or specification, it is wrong. What is required is a much different kind of account, like our ET-based one.

Finally, we point out that even though one might argue that results concerning the indistinguishability of unbounded nondeterminism [13, 1] (e.g., a^*b being observationally indistinguishable from a^*b+a^ω) are a problem for our approach, this is not the case because we are assuming that agents can reason about *all* possible program executions/futures.

REFERENCES

[1] K.R. Apt and E.R. Olderog, *Verifi cation of Sequential and Concurrent Programs*, Springer-Verlag, 1997.

[2] Chitta Baral and Michael Gelfond, *Reasoning Agents in Dynamic Domains*, chapter 12, 257–275, Kluwer, 2000.

[3] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso, 'Planning in nondeterministic domains under partial observability via symbolic model checking', in *Proc. of IJCAI-01*, pp. 473–478, (2001).

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

[5] Giuseppe De Giaccomo, Luca Iocchi, Daniele Nardi, and Riccardo Rosati, 'Planning with sensing for a mobile robot', in *Proc, of ECP-97*, pp. 156–168, (1997).

[6] Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque, 'ConGolog, a concurrent programming language based on the situation calculus', *Artifi cial Intelligence*, **121**, 109–169, (2000).

[7] Giuseppe De Giacomo and Hector J. Levesque, 'An incremental interpreter for high-level programs with sensing', in *Logical Foundations for Cognitive Agents*, eds., Hector J. Levesque and Fiora Pirri, 86–102, Springer-Verlag, (1999).

[8] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J. J. Ch. Meyer, 'A formal semantics for an abstract agent programming language', in *Proc. of ATAL-97*, pp. 215–229, (1998).

[9] 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).

[10] H. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. Scherl, 'GOLOG: A logic programming language for dynamic domains', *Journal of Logic Programming*, **31**, 59–84, (1997).

[11] Fangzhen Lin and Hector J. Levesque, 'What robots can do: Robot programs and effective achievability', *Artifi cial Intelligence*, **101**, 201–226, (1998).

[12] John McCarthy and Patrick Hayes, 'Some philosophical problems from the standpoint of artifi cial intellig ence', in *Machine Intelligence*, eds., B. Meltzer and D. Michie, volume 4, 463–502, Edinburgh University Press, (1979).

[13] Robin Milner, *Communication and Concurrency*, Prentice Hall, 1989.

[14] Robert C. Moore, 'A formal theory of knowledge and action', in *Formal Theories of the Common Sense World*, eds., J. R. Hobbs and Robert C. Moore, 319–358, (1985).

[15] Ron Petrick and Fahiem Bacchus, 'A knowledge-based approach to planning with incomplete information and sensing', in *Proc. of AIPS-02*, pp. 212–221, (2002).

[16] Gordon Plotkin, 'A structural approach to operational semantics', Technical Report DAIMI-FN-19, Computer Science Dept., Aarhus University, Denmark, (1981).

[17] Anand S. Rao, 'AgentSpeak(L): BDI agents speak out in a logica computable language', in *Agents Breaking Away (LNAI)*, eds., W. Vander Velde and J. W. Perram, volume 1038, 42–55, Springer-Verlag, (1996).
[18] Raymond Reiter, *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*, MIT Press, 2001.
[19] Sebastian Sardina, Yves De Giacomo, Giuseppe Lespénce, and Hector Levesque, 'On the semantics of deliberation in IndiGolog – from theory to implementation', *Annals of Mathematics and Artifi cial Intelligence*, **41**(2–4), 259–299, (2004). Previous version appeared in Proc. of KR-2002.
[20] T. Son, C. Baral, and S. McIlraith, 'Extending answer set planning with sequence, conditional, loop, non-deterministic choice, and procedure constructs', in *Proceedings of the AAAI Spring Symposium on Answer Set Programming: Towards Effi cient and Scalable Knowledge Representation and Reasoning*.
[21] Tran Son, Phan Huy Tu, and Chitta Baral, 'Planning with sensing actions and incomplete information using logic programming', in *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR 2004*, Lecture Notes in Computer Science, pp. 261–274, Fort Lauderdale, FL, USA. Springer.