On Ability to Autonomously Execute Agent Programs with Sensing

Sebastian Sardiña*
Dept. of Computer Science
University of Toronto
Toronto, Canada
ssardina@cs.toronto.edu

Giuseppe De Giacomo
Dip. Informatica e Sistemistica
Univer. di Roma "La Sapienza"
Roma, Italy
degiacomo@dis.uniroma1.it

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

Hector J. Levesque
Dept. of Computer Science
University of Toronto
Toronto, Canada
hector@cs.toronto.edu

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. 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, 3, 9, 11, 6]—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 [5] 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 such 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.

Following this, we consider richer notions of deliberation/planning, how they can be formalized, and how they

* First author is a student.

can be exploited in an agent programming language. In Section 6, we discuss the notion of ability to achieve a goal, and show how it 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 in Section 7, 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 in Section 8, we show how angelic knowing how can be used to specify a powerful planning construct in the IndiGolog agent programming language. We end by reviewing the paper's contributions, discussing the lessons for agent programming language design, and discussing future work.

All proofs can be found at the following address:
http://www.cs.toronto.edu/∼ssardina/papers/paamas04.pdf

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, 5]. 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 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 the 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 [5] and we use its single step transition semantics in the style of [16]. This semantics introduces two special predicates $Trans$ and $Final$ are introduced: $Trans(\delta, s, \delta', s')$ 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, 5], 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 [5].

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 [1] (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 say that a configuration (δ, σ) *may evolve* to configuration (δ', σ') *w.r.t. a (background) theory $\mathcal{D}$* if and only (δ, σ) *may evolve* to (δ', σ') *w.r.t. some model M* (relative to $\mathcal{D}$). Note that we now have two notions of "*configuration evolution*," one w.r.t. a particular model (cf. Section 2) and one w.r.t. a theory. Again, the model is used only to obtain the sensing values corresponding to some possible environment and not determine the truth values of formulas. An important point is that this alternative version of configuration evolution appeals to *consistency* in that it considers all evolutions of a configuration in which the sensing outcome (i.e., the environment response) is *consistent* with the underlying theory.

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

(E1) if (δ, σ) is *final*, then $\mathcal{R}(\delta, \sigma)$;

(E2) if (δ, σ) *may evolve* to configurations $(\delta', \sigma \cdot (a, \mu_i))$ *w.r.t. theory* $\mathcal{D}$ with $i = 1..k$ for some $k \geq 1$, and $\mathcal{R}(\delta', \sigma \cdot (a, \mu_i))$ holds for all $i = 1..k$, 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}.

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. For instance, for a conditional program involving different primitive actions a_1 and a_2 in the "then" and "else" branches (i.e., such that $\mathcal{D} \models a_1 \neq a_2$), the agent must know whether the test holds and know how to execute the appropriate branch:

$\mathit{KHow}_{EC}(\textbf{if } \phi \textbf{ then } a_1 \textbf{ else } a_2 \textbf{ endIf}, \sigma)$ iff
$\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma]\} \models \phi(end[\sigma])$ and $\mathit{KHow}_{EC}(a_1, \sigma)$
or
$\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma]\} \models \neg\phi(end[\sigma])$ and $\mathit{KHow}_{EC}(a_2, \sigma)$.

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

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

1 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). Also, here we deal with non-terminating, i.e., infinite executions.

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:

$$RemainingChops(do(a,s)) = n \equiv$$
$$(a = chop \wedge RemainingChops(s) = n+1) \vee$$
$$(a \neq chop \wedge RemainingChops(s) = n).$$

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

$$Poss(chop, s) \equiv (RemainingChops > 0),$$
$$Poss(look, s) \equiv True.$$

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

$$SF(chop, s) \equiv True,$$
$$SF(look, s) \equiv (RemainingChops(s) = 0).$$

Notice that sentence $\exists n.RemainingChop(S_0) = n$ (where the variable n ranges over $\mathbb{N}$) is entailed by this theory so "infinitely" hard tree trunks are ruled out. Nonetheless, the theory *does not* entail the sentence $RemainingChop(S_0) < k$ 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 3.1 *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.*

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}. 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. 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 [6], 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 [6], 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 $(\delta', \sigma \cdot (a, \mu_i))$ w.r.t. theory $\mathcal{D}$, $1 \leq i \leq 2$, and there exist $\delta_i^{tp'}$ such that $\mathcal{R}(\delta', \sigma \cdot (a, \mu_i), \delta_i^{tp'})$, then $\mathcal{R}(\delta, \sigma, (a; \textbf{if } \phi \textbf{ then } \delta_1^{tp} \textbf{ else } \delta_2^{tp} \textbf{ endIf}))$ where ϕ is the condition on the right hand side of the sensed fluent ax-

iom for a, and δ_i^{tp} is nil if $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma \cdot (a, \mu_i)]\}$ is inconsistent, and δ_i^{tp} is $\delta_i^{tp'}$ otherwise.

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 4.1 *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*
- $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma]\} \models \exists s.Do(\delta^{tp}, end[\sigma], s) \wedge Do(\delta, end[\sigma], s)$.

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

Theorem 4.2 *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 executions require 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 in Section 3 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 us the infinite history corresponding to the hypothetical scenario described at the end of Section 3. 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*$(\delta, \sigma, M)$.

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 5.1 *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 5.2 *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. Ability and Planning

Using our two notions of knowing how, we can define related notions of ability to achieve a goal [9, 14, 3]. For both *KHow*$_{EC}$ and *KHow*$_{ET}$, we formally specify when an agent *can achieve a goal ϕ in a history σ*, $Can_X(\phi, \sigma)$ (where $X = EC$ or $X = ET$) iff there exists a deterministic program δ^d such that $KHow_X(\delta^d, \sigma)$ and

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

i.e., the agent knows how to execute the program and the program (always) terminates in a situation where the goal has been achieved. For the tree chopping problem we have that *Can*$_{ET}(Down, S_0)$ holds, but *Can*$_{EC}(Down, S_0)$ does not. Based on Theorem 5.2, we can also show that *Can*$_{ET}$ is a strictly more general account of ability than *Can*$_{EC}$.

One can easily use this to define a new construct $achieve(\phi)$ that does planning to achieve a goal and add it to an agent programming language. But generally, one also want to be able to specify constraints on the search for a plan to achieve the goal, constraints on what sort of plan should be considered. One way to do this is to specify the task as that of executing a nondeterministic program, which we now turn to.

7. Nondeterministic Programs

When it comes to nondeterministic programs, the notion of knowing how needs to be extended. In particular there are two main notions of nondeterministic program execution of interest for agents. The first notion states that choices at program choice points are under the control of the agent, and hence can involve reasoning on part of the agent. In this case, an agent knows how to execute a nondeterministic program if she is able to *make choices* along the execution of the program so that, at each step, the action chosen is known to be executable and such that no matter what sensing output is obtained as a result of doing the action, she can continue this process and eventually terminate successfully. This assumes that the agent does some planning/lookahead to find a strategy for executing the nondeterministic program such that this strategy is guaranteed to succeed.

The second notion of nondeterministic program execution states that choices at its choice points are not under the control of the agent. Hence, no reasoning, planning or lookahead, is involved in choosing the next step, and all choices must be accepted and dealt with by the agent. In this case, the agent knows how to execute a nondeterministic program if every possible way she may execute it would eventually terminate successfully.

Suppose that we enlarge our programming language with the following nondeterministic constructs:

$\phi?$,	wait for a condition
$\delta_1 \mid \delta_2$,	nondeterministic branch
$\pi\, x.\, \delta(x)$,	nondeterministic choice of argument
δ^*,	nondeterministic iteration
$\delta_1 \parallel \delta_2$,	(interleaved) concurrency

The first wait/test construct blocks until its condition becomes true and it is still a deterministic construct which produces transitions involving no action. Such a construct is useful with nondeterministic programs and can be easily accommodated into the already given definitions of KHow_{EC} and KHow_{ET} by just adding one extra condition to their corresponding definitions:

(E3) if (δ, σ) *may evolve* to (δ', σ) *w.r.t. theory* $\mathcal{D}$ and $\mathcal{R}(\delta', \sigma)$, then $\mathcal{R}(\delta, \sigma)$ (for KHow_{EC})

(T3) if (δ, σ) *may evolve* to (δ', σ) w.r.t. M and $\mathcal{R}(\delta', \sigma)$, then $\mathcal{R}(\delta, \sigma)$ (for KHow_{ET})

Let us first focus on knowing how for nondeterministic programs where choices are under the control of the agent, i.e., *angelic* knowing how. We can define EC/ET versions of this notion. We can take $\mathit{KHow}^{Ang}_{ECnd}$ to be just KHow_{EC} assuming (E3) is included. This works just fine for nondeterministic programs, though, it still fails to capture (good) programs with an unbounded number of steps. On the other hand, KHow_{ET}, as defined and assuming (T3) is included, is too weak. Consider the following example. There is a treasure behind one of two doors but the agent does not know which. We want to know if she knows how to execute the program δ_{treas}:

$$[(open1; look) \mid (open2; look)]; AtTreasure?$$

Intuitively, the agent does not know how to execute δ_{treas} because she does not know which door to open to get to the treasure. However, $\mathit{KHow}_{ET}(\delta_{treas}, \epsilon)$ holds. Indeed in a model M_1 where the treasure is behind door 1, the agent can pick the $open1$ action, and then we have $\mathit{KHowInM}((look; AtTreasure?), [(open1, 1)], M_1)$, and thus $\mathit{KHowInM}(\delta_{treas}, \epsilon, M_1)$. Similarly, in a model M_2 where the treasure is behind door 2, she can pick $open2$, and thus $\mathit{KHowInM}(\delta_{treas}, \epsilon, M_2)$.

The problem with KHow_{ET} for nondeterministic programs is that the action chosen need not be the same in different models even if they have generated the same sensing results up to that point and are indistinguishable for the agent. We can solve this problem by requiring that the agent have a common strategy for all models/environments, i.e., that she has a *deterministic* program δ^d that she knows how to execute (in all models of the theory) and knows δ^d will terminate in a *final* situation of the given program δ:

$\mathit{KHow}^{Ang}_{ETnd}(\delta, \sigma)$ iff there is a deterministic program δ^d such that $\mathit{KHow}_{ET}(\delta^d, \sigma)$ and

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

We do not think that it is possible to obtain a much simpler general formalization of knowing how and to avoid the existential quantification over deterministic programs/strategies.

Proposition 7.1 *For any background theory $\mathcal{D}$ and any configuration (δ, σ), if $\mathit{KHow}^{Ang}_{ECnd}(\delta, \sigma)$ holds, then $\mathit{KHow}^{Ang}_{ETnd}(\delta, \sigma)$. Also, there is a background theory $\mathcal{D}^*$ and a configuration (δ^*, σ^*) such that $\mathit{KHow}^{Ang}_{ETnd}(\delta^*, \sigma^*)$ holds, but $\mathit{KHow}^{Ang}_{ECnd}(\delta^*, \sigma^*)$ does not.*

An interesting point is that, now, ability to achieve a goal, as defined in the previous section, can be seen as a special case of knowing how to execute a (nondeterministic) program. Indeed, we (re)define $Can_X(\phi, end[\sigma])$, as follows:

$$Can_X(\phi, \sigma) \text{ iff } KHow^{Ang}_{Xnd}(\textbf{while } \neg\phi \textbf{ do } (\pi a.a) \textbf{ end}, \sigma),$$

i.e., the agent knows how to execute the program that involves repeatedly choosing and executing some action until the goal has been achieved.

Next, we turn our attention to knowing how for nondeterministic programs where choices are not under the control of the agent, i.e., *demonic* knowing how. We start by defining a relation between a program δ_a and another program δ_b that simulates it:

$$Simulates(\delta_a, \delta_b, s) \stackrel{\text{def}}{=} \exists R.(\forall \delta_1, \delta_2, s. R(\delta_1, \delta_2, s) \supset \ldots\ ) \wedge R(\delta_a, \delta_b, s)$$

where the ellipsis stands for the conjunction of:

$$Final(\delta_1, s) \supset Final(\delta_2, s)$$
$$\neg Final(\delta_1, s) \wedge \forall \delta_1', s'.Trans(\delta_1, s, \delta_1', s') \supset$$
$$\exists \delta_2'.Trans(\delta_2, s, \delta_2', s') \wedge R(\delta_1', \delta_2', s)$$

For instance, if $\delta = ((a;b) \mid (a;c))$ and $\delta' = (a;(b \mid c))$, then $Simulates(\delta, \delta', s)$ holds, but $Simulates(\delta', \delta, s)$ does not (provided all actions are possible from situation s).

We say that an agent knows how to execute a *nondeterministic* program in the demonic sense whenever she knows how to execute every possible *deterministic* simulation of it: $\mathit{KHow}^{Dem}_{ETnd}(\delta, \sigma)$ iff $\mathit{KHow}_{ET}(\delta^d, \sigma)$ holds for every deterministic program δ^d such that

$$\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma]\} \models Simulates(\delta^d, \delta, end[\sigma])$$

This account of knowing how can be seen as a specification of nondeterministic programs when choices at choice points are not under the control of the agent. This is relevant for agent programming languages in which nondeterministic programs are used to represent the agent behavior but an online/reactive account of execution of these programs is used (3APL [8], AgentSpeak(L) [17], etc). In those frameworks, the nondeterministic program must work no matter how choice points are resolved.

8. Deliberation in IndiGolog

We can use our formalization of knowing how to provide a better semantics for search/deliberation in agent programming languages. Let's show how this is done for IndiGolog [7]. In IndiGolog, the programmer controls when deliberation occurs. By default, there is no deliberation or lookahead; the interpreter arbitrarily selects a transition and performs it on-line, until it gets to a final configuration. To perform deliberation/lookahead, the programmer uses the *search* operator $\Sigma(\delta)$, where δ is the part of the program that needs to be deliberated over. The idea is that this Σ only allows a transition for δ if there exists a sequence of further transitions that would allow δ to terminate successfully. The original definition for the search operator in [7] failed to ensure that the plan was epistemically feasible, i.e., allowed cases where a sequence of transitions must exist, but where the agent cannot determine what the sequence is; our proposal here corrects this.

We can specify the semantics of this operator by extending our earlier notions of configuration evolution and finalness, used in defining online executions:

A configuration $(\Sigma(\delta), \sigma)$ is *final* if and only if the configuration (δ, σ) is final. A configuration $(\Sigma(\delta), \sigma)$ may evolve to a configuration (δ', σ') w.r.t. a model M if and only if there exists a deterministic program δ^d such that:

- $\mathit{KHow}^{Ang}_{ET}(\delta^d, \sigma)$ holds;
- $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma]\} \models$
  $\exists s.Do(\delta^d, end[\sigma], s) \ \wedge \ Do(\delta, end[\sigma], s)$
  (i.e. δ^d must terminate in a $Final$ situation of the given program δ);
- (δ^d, σ) may evolve to $(\delta^{d'}, \sigma')$ w.r.t. M;
- the program that remains afterwards is $\delta' = \Sigma(\delta^{d'})$.

This semantic is metatheoretic and provides a rather simpler alternative to that proposed in [6], which is based on an epistemic version of the situation calculus.

9. Discussion and Conclusion

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. Then, we developed an alternative entailment and truth-based formalization, KHow_{ET}, that handles indefinite iteration examples correctly. Finally, we proposed accounts of ability and knowing how for nondeterministic programs. 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 metatheoretic 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. [2, 15, 4], 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*$_{EC}$.

Our results show that this 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.

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

Finally, we believe that there is a close relationship between *KHow*$_{ETnd}$ some of the earlier epistemic accounts of knowing how and ability [14, 3, 9, 11, 6]. We hope to get some correspondence results on this soon.

References

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

[2] 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*, pages 473–478, 2001.

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

[4] G. De Giaccomo, L. Iocchi, D. Nardi, and R. Rosati. Planning with sensing for a mobile robot. In *Proc, of ECP-97*, pages 156–168, 1997.

[5] G. De Giacomo, Y. Lespérance, and H. J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artifi cial Intelligence*, 121:109–169, 2000.

[6] G. De Giacomo, Y. Lespérance, H. J. Levesque, and S. Sardiña. On the semantics of deliberation in IndiGolog: From theory to implementation. In *Proc. of KR-02*, pages 603–614, 2002.

[7] G. De Giacomo and H. J. Levesque. An incremental interpreter for high-level programs with sensing. In H. J. Levesque and F. Pirri, editors, *Logical Foundations for Cognitive Agents*, pages 86–102. Springer-Verlag, 1999.

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

[9] Y. Lespérance, H. J. Levesque, F. Lin, and R. 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] F. Lin and H. J. Levesque. What robots can do: Robot programs and effective achievability. *Artifi cial Intelligence*, 101:201–226, 1998.

[12] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artifi cial intellig ence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, 1979.

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

[14] R. C. Moore. A formal theory of knowledge and action. In J. R. Hobbs and R. C. Moore, editors, *Formal Theories of the Common Sense World*, pages 319–358. 1985.

[15] R. Petrick and F. Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *Proc. of AIPS-02*, pages 212–221, 2002.

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

[17] A. S. Rao. AgentSpeak(L): BDI agents speak out in a logica computable language. In W. V. Velde and J. W. Perram, editors, *Agents Breaking Away (LNAI)*, volume 1038, pages 42–55. Springer-Verlag, 1996.

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

A. PROOFS

PROOF OF THEOREM 3.1:
Let $\mathcal{R}$ be the smallest relation defining *KHow*$_{EC}$. Then, $\mathcal{R}$ is the smallest relation satisfying conditions (E1)-(E3).

Assume, by the contrary, that $\mathcal{R}(\delta_{tc}, [chop, (look, 0)]^m)$ for some $m \geq 0$. By Lemma A.1, there exists a binary relation $\mathcal{R}' \subseteq \mathcal{R}$ such that $\mathcal{R}'(\delta_{tc}, [chop, (look, 0)]^k)$ does not holds for any $k \geq 0$ and such that $\mathcal{R}'$ satisfies (E1)-(E3).
This means that $\mathcal{R}'(\delta_{tc}, [chop, (look, 0)]^m)$ does not hold and, therefore, $\mathcal{R}' \subset \mathcal{R}$ (i.e., $\mathcal{R}'$ is a proper subset of $\mathcal{R}$.) Therefore, $\mathcal{R}$ is **not** the smallest relation satisfying (E1)-(E3) which contradicts the initial assumption.

Then, $\neg\mathcal{R}(\delta_{tc}, [chop, (look, 0)]^k)$ for all $k \geq 0$ and the relation *KHow*$_{EC}$ does not contain the pair $(\delta_{tc}, [chop, (look, 0)]^k)$. In particular, (δ_{tc}, ϵ) is not in the *KHow*$_{EC}$ relation when we take $k = 0$.

Lemma A.1
Let δ_{tc} be the program to bring the tree down. This Lemma will prove a more general version of the result by showing that KHow$^{Ang}_{ECnd}(\delta_{tc}, \sigma)$ does not hold. It is obvious to see that KHow$_{EC}$ is always a subset of KHow$^{Ang}_{ECnd}$.

Let $\mathcal{R}$ be a set of pairs program-history (δ, σ) satisfying conditions (E1)-(E3) of the definition of KHow$^{Ang}_{ECnd}$. Then, there exists a set $\mathcal{R}' \subseteq \mathcal{R}$ such that $\mathcal{R}'$ satisfies conditions (E1), (E2) and (E3), and such that $\mathcal{R}'(\delta_{tc}, [chop, (look, 0)]^k)$ does not hold for any $k \geq 0$.

PROOF:

Intuitively, we will prove that we can "safely" remove all program-history pairs of the form $(\delta_{tc}, [chop, (look, 0)]^k)$ from the set $\mathcal{R}$. To do that, let us define $\mathcal{R}' = \mathcal{R} - \beta$, where:

$$\beta = \{(\delta_{tc}, \sigma) : \sigma = [chop, (look, 0)]^k \wedge k \geq 0\} \cup \{((look; \delta_{tc}), \sigma) : \sigma = [chop, (look, 0)]^k \cdot chop \wedge k \geq 0\}$$

Clearly, $\mathcal{R}' \subseteq \mathcal{R}$ and $\neg\mathcal{R}'(\delta_{tc}, \epsilon)$. It remains to show that relation $\mathcal{R}'$ does indeed satisfy conditions (E1), (E2), and (E3).

The set $\mathcal{R}'$ satisfies (E1)

This is trivial since all configurations for which β holds are not final. In concrete, if (δ, σ) is final, then $\mathcal{R}(\delta, \sigma)$ and $\neg\beta(\delta, \sigma)$. Therefore, $\mathcal{R}'(\delta, \sigma)$.

The set $\mathcal{R}'$ satisfies (E2)

Let δ be any program and σ be any history. Let us consider the following three exhaustive and exclusive cases depending on the form of the history σ:

1. *$\sigma \neq [chop, (look, 0)]^k$ and $\sigma \neq [chop, (look, 0)]^k \cdot chop$.*
   First, note that, because of the form of σ, $\neg\beta(\delta, \sigma)$ is true. Assume then that there exist configurations $(\delta_i, \sigma \cdot (a, \mu_i))$, $i \geq 1$, such that (δ, σ) may evolve to w.r.t. theory $\mathcal{D}_{tc}$ and such that $\mathcal{R}'(\delta_i, \sigma \cdot (a, \mu_i))$ apply.
   Since $\mathcal{R}' \subseteq \mathcal{R}$, then $\mathcal{R}(\delta', \sigma \cdot (a, \mu))$ holds, and, given that $\mathcal{R}$ does satisfy condition (E2), $\mathcal{R}(\delta, \sigma)$ is true. Due to the fact that $\neg\beta(\delta, \sigma)$, $\mathcal{R}'(\delta, \sigma)$ holds as well.
2. *$\sigma = [chop, (look, 0)]^k$, for some $k \geq 0$.*
   Assume that there exist configurations $(\delta_i, \sigma \cdot (a, \mu_i))$, $i \geq 1$, such that (δ, σ) may evolve to w.r.t. theory $\mathcal{D}_{tc}$. Let us now consider the following two exhaustive and exclusive cases:
   *(a) $\delta \neq \delta_{tc}$, i.e., $\delta \neq$ **while** $\neg Down$ **do** $chop; look$ **endWhile**.*
   Suppose that $\mathcal{R}'(\delta_i, \sigma \cdot (a, \mu_i))$ apply, for all $i \geq i$. Hence, $\mathcal{R}(\delta', \sigma \cdot (a, \mu))$ holds because $\mathcal{R}' \subseteq \mathcal{R}$; and $\mathcal{R}(\delta, \sigma)$ holds since $\mathcal{R}$ does satisfy (E3). Moreover, because of the form of both δ and σ, it is easy to check that $\neg\beta(\delta, \sigma)$. Thus, $\mathcal{R}'(\delta, \sigma)$ holds.
   *(b) $\delta = \delta_{tc}$, i.e., $\delta =$ **while** $\neg Down$ **do** $chop; look$ **endWhile**.*
   In this case, there could only be one possible configuration (δ', σ') to which (δ, σ) may evolve to w.r.t. theory $\mathcal{D}$, namely, $\delta' = (look; \delta_{tc})$ and $\sigma' = \sigma \cdot chop$. Since $\sigma' = [chop, (look, 0)]^k \cdot chop$, $\beta(\delta', \sigma')$ holds, and, therefore, $\neg\mathcal{R}'(\delta', \sigma')$ applies. Thus, condition (E3) is trivially satisfied by $\mathcal{R}'$.

3. $\sigma = [chop, (look, 0)]^k \cdot chop$, *for some* $k \geq 0$.

Here, let us consider the following two exhaustive and exclusive cases depending on the form of the program δ*:*

(a) $\delta \neq (look; \delta_{tc})$, *i.e.,* $\delta \neq (look;$ **while** $\neg Down$ **do** $chop; look$ **endWhile**$)$.

Suppose that $\mathcal{R}'(\delta_i, \sigma \cdot (a, \mu_i))$ *apply, for all* $i \geq i$. *Hence,* $\mathcal{R}(\delta', \sigma \cdot (a, \mu))$ *holds because* $\mathcal{R}' \subseteq \mathcal{R}$; *and* $\mathcal{R}(\delta, \sigma)$ *holds since* $\mathcal{R}$ *does satisfy (E3). Moreover, because of the form of both* δ *and* σ, *it is easy to check that* $\neg\beta(\delta, \sigma)$. *Thus,* $\mathcal{R}'(\delta, \sigma)$ *holds.*

(b) $\delta = (look; \delta_{tc})$, *i.e.,* $\delta = (look;$ **while** $\neg Down$ **do** $chop; look$ **endWhile**$)$.

Finally, the most interesting case. We can easily verify that (δ, σ) *may evolve to* (δ_{tc}, σ') *w.r.t. theory* $\mathcal{D}$, *where* $\sigma' = \sigma \cdot (look, 0)$. *Informally, there is always a possible evolution of the configuration for which the tree was just sensed to be still up. Technically, there is always a model* M *of* $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma \cdot (look, 0)]\}$.

Next, we observe that $\sigma' = [chop, (look, 0)]^{k+1}$, *and, therefore,* $\beta(\delta_{tc}, \sigma')$ *holds. Thus, by the way we defined* $\mathcal{R}'$ *in terms of* $\mathcal{R}$ *and* β, $\neg\mathcal{R}'(\delta_{tc}, \sigma \cdot (look, 0))$ *applies and the condition (E3) is trivially satisfied in this case.*

In words, there is always a possible and consistent sensing outcome for the only legal action-transition (i.e., a chopping action) such that the resulting configuration is not *on the smallest set. Recall that in order to include the original configuration into the smallest set, we require that no matter how sensing turns out to be, the resulting configuration should be on the smallest set.*

The set $\mathcal{R}'$ *satisfies (E3)*

Let δ *be any program and* σ *be any history. Let us consider the following two exhaustive and exclusive cases depending on the form of the program* δ*:*

(a) $\delta \neq \delta_{tc}$ *and* $\delta \neq look; \delta_{tc}$

First, note that, due to the form of δ, $\neg\beta(\delta, \sigma)$ *is true. Assume then that there exists a program* δ' *such that* (δ, σ) *may evolve to* (δ', σ) *w.r.t.* $\mathcal{D}_{tc}$ *and such that that* $\mathcal{R}'(\delta', \sigma)$ *holds. Since* $\mathcal{R}' \subseteq \mathcal{R}$, *then* $\mathcal{R}(\delta', \sigma)$ *holds as well. Given that* $\mathcal{R}$ *satisfies (E3),* $\mathcal{R}(\delta, \sigma)$. *Finally, because* $\neg\beta(\delta, \sigma)$, *it follows that* $\mathcal{R}'(\delta, \sigma)$ *is true.*

(b) $\delta = \delta_{tc}$ *or* $\delta = look; \delta_{tc}$

In this case, there is no program δ' *such that* (δ, σ) *may evolve to* (δ', σ) *as there can be no transition without an action, either a* $look$ *action or a* $chop$ *one. Therefore, condition (E3) is trivially satisfied.*

PROOF OF PROPOSITION 5.1:
Let M be a model of $\mathcal{D}_{tc} \cup \mathcal{C}$. We are to prove that *KHowInM*(δ, ϵ, M). Let k be the number of chops required initially to get the three down in model M, i.e., $M \models RemainingChops(S_0) = k$. It is not difficult to see that history $\sigma = [(chop, 1) \cdot (look, 1)]^k$ completely executes program δ_{tc} in model M. Moreover, we can check the following points:

- (δ_{tc}, σ) is final and hence, by condition (T1), *KHowInM*(δ_{tc}, σ, M) holds;
- $((look; \delta_{tc}), [(chop, 1) \cdot (look, 0)]^{k-1} \cdot (chop, 1))$ may evolve to (δ_{tc}, σ). By condition (T2), *KHowInM*$((look; \delta_{tc}), [(chop, 1) \cdot (look, 0)]^{k-1}, M)$ holds;
- $(\delta_{tc}, [(chop, 1) \cdot (look, 0)]^{k-1})$ may evolve to $((look; \delta_{tc}), [(chop, 1) \cdot (look, 0)]^{k-1} \cdot (chop, 1))$. By condition (T2), *KHowInM*$((look; \delta_{tc}), [(chop, 1) \cdot (look, 0)]^{k-1}, M)$ holds;
- ...
- $((look; \delta_{tc}), (chop, 1))$ may evolve to $(\delta_{tc}, [(chop, 1) \cdot (look, 0)]^1)$. By condition (T2), *KHowInM*$((look; \delta_{tc}), (chop, 1), M)$ holds;
- (δ_{tc}, ϵ) may evolve to $((look; \delta_{tc}), (chop, 1))$. By condition (T2), *KHowInM*$(\delta_{tc}), \epsilon, M)$ holds.

PROOF OF THEOREM 5.2:

Part (ii) folllows directly from Proposition 5.1 and Theorem 3.1, taking $\mathcal{D}_{tc}$ as the background theory.

Let us prove part (i). Suppose that (δ^*, σ^*) is a configuration such that *KHow*$_{ET}(\delta^*, \sigma^*)$ does not hold. We shall prove that *KHow*$_{EC}(\delta^*, \sigma^*)$ is not true either.

By hypothesis, there is a model M of $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma^*]\}$ such that $KHowInM(\delta^*, \sigma^*, M)$ is false. We define the binary relation $\mathcal{R}$ as follows: for all programs δ and all histories σ, $\mathcal{R}(\delta, \sigma)$ iff $KHowInM(\delta, \sigma, M)$. We shall prove now that $\mathcal{R}$ satisfies conditions (E1) and (E2) in the definition of *KHow*$_{EC}$:

(E1) Suppose that (δ, σ) is final. Then, by condition (T1) in the definition of ***KHowInM***, $KHowSM(\delta, \sigma, M)$ is true and $\mathcal{R}(\delta, \sigma)$ holds.

(E2) Suppose that (δ, σ) *may evolve* to configurations $(\delta', \sigma \cdot (a, \mu_i))$ *w.r.t. theory* $\mathcal{D}$ with $1 \leq i \leq k$, and that $\mathcal{R}(\delta', \sigma \cdot (a, \mu_i))$ for all $1 \leq i \leq k$.

If σ is inconsistent in M (i.e., $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma]\}$ is unsatisfiable,) then configuration (δ, σ) is final (in the general case), $KHowETd(\delta, \sigma, M)$ holds, and, therefore, $\mathcal{R}(\delta, \sigma)$ holds as well.

Assume then that σ is consistent in M. Then, there should be some μ_j, $1 \leq j \leq k$, such that (δ, σ) may evolve to $(\delta', \sigma \cdot (a, \mu_j))$ w.r.t M. Notice that $M \models Trans(\delta, end[\sigma], \delta', do(a, end[\sigma]))$ and that $\mu_j = 1$ iff $M \models SF(a, end[\sigma])$. Since $\mathcal{R}(\delta', \sigma \cdot (a, \mu_j))$ holds, so does ***KHowInM***$(\delta', \sigma \cdot (a, \mu_i), M)$. Then, by condition (T3) in the definition of ***KHowInM***, $KHowInM(\delta, \sigma, M)$ holds and, by the way we have defined $\mathcal{R}$, it follows that $\mathcal{R}(\delta, \sigma)$ holds.

To generalize the result to *KHow*$^{Ang}_{ECnd}$ and *KHow*$^{Ang}_{ETnd}$ we just need to add the following case to the above two:

(E3) Suppose that (δ, σ) *may evolve* to (δ', σ) *w.r.t. theory* $\mathcal{D}$ and that $\mathcal{R}(\delta', \sigma)$ holds. Then, since the history remains the same, it is also true that (δ, σ) *may evolve* to (δ', σ) *w.r.t. model* M (the configuration evolution does not depend on the environment). Moreover, by the way $\mathcal{R}$ was defined, $KHowSM(\delta', \sigma, M)$ holds. By condition (T2) in the definition of ***KHowInM***, it follows that $KHowSM(\delta, \sigma, M)$ and, therefore, $\mathcal{R}(\delta, \sigma)$ holds as well.

PROOF OF PROPOSITION 7.1:
Assume *KHow*$_{EC}(\delta, \sigma)$ holds. By Theorem 4.2, there is a program δ^{tp} such that *KHowBy*$_{EC}(\delta, \sigma, \delta^{tp})$. By Theorem 4.1, δ^{tp} is a tree program and, thus, a deterministic one, and $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma]\} \models \exists s.Do(\delta^{tp}, end[\sigma], s) \wedge Do(\delta, end[\sigma], s)$. By Theorem 5.2, *KHow*$_{ET}(\delta^{tp}, \sigma)$ holds and, putting all together, *KHow*$_{ETnd}(\delta, \sigma)$ applies.

Lemma A.2 (Induction Principle for *KHow*$^{Ang}_{ECnd}$)
For all relations $T(\delta, \sigma)$ over programs and histories, if $T(\delta, \sigma)$ is closed under assertions (E1), (E2), and (E3) of the definition of KHow$^{Ang}_{ECnd}$, then for all programs δ and histories σ, if KHow$^{Ang}_{ECnd}(\delta, \sigma)$, then $T(\delta, \sigma)$.

PROOF:
As for Theorem 3 of [5].

Lemma A.3 (Induction Principle for *KHowBy*$_{EC}$)
For all relations $T(\delta, \sigma, \delta')$ over programs, histories and programs, if $T(\delta, \sigma, \delta')$ is closed under assertions (A), (B), and (C) of the definition of KHowBy$_{EC}$, then for all programs $\delta, delta'$, and histories σ, if KHowBy$_{EC}(\delta, \sigma, \delta')$, then $T(\delta, \sigma, \delta')$.

PROOF:
As for Theorem 3 of [5].

PROOF OF THEOREM 4.1:
We shall prove a more general result w.r.t. *KHow*$^{Ang}_{ECnd}$. To that end, we add the following case to the definition of *KHowBy*$_{EC}$:

(C) if (δ, σ) *may evolve* to (δ', σ) *w.r.t. theory* $\mathcal{D}$ and $\mathcal{R}(\delta', \sigma, \delta^{tp\prime})$, then $\mathcal{R}(\delta, \sigma, True?; \delta^{tp\prime})$

In addition we enrich a bit the notion of $TREE$ programs to include two special test steps:

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

where a is any non-sensing action, and dpt_1 and dpt_2 are tree programs.

The proof goes by induction on *KHowBy*$_{EC}$. The property that we have to show is closed under the assertions in the definition of *KHowBy*$_{EC}$ (and hence be able to apply Lemma A.3) is the following one:

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

Checking that $T(\delta, \sigma, \delta^{tp})$ is closed under assertion (A) of the definition of *KHowBy*$_{EC}$:

Assume that (δ, σ) is a final configuration. Let us show that $T(p, h, nil)$. By definition of $\mathit{KHow}^{Ang}_{ECnd}$, we have $\mathit{KHow}^{Ang}_{ECnd}(\delta, \sigma)$. As well, nil is a $TREE$ program, and we have $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma]\} \models Do(nil, end[\sigma], end[\sigma]) \wedge Do(\delta, end[\sigma], end[\sigma])$. Thus, $T(\delta, \sigma, nil)$.

Checking that $T(\delta, \sigma, \delta^{tp})$ is closed under assertions (B) of the definition of KHowBy_{EC}:

Assume that (δ, σ) may evolve to one or more configurations $(\delta', \sigma \cdot (a, \mu_i))$ (for some action a) w.r.t. theory $\mathcal{D}$ where $1 \leq i \leq 2$ and that there exists δ_i^{tp} such that $T(\delta', \sigma \cdot (a, \mu_i), \delta_i^{tp'})$. This means that:

(a) if $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma \cdot (a, 1)]\}$ is consistent, then $T(\delta', \sigma \cdot (a, 1), \delta_1^{tp'})$, i.e., $\mathit{KHow}^{Ang}_{ECnd}(\delta', \sigma \cdot (a, 1))$ and $\delta_1^{tp'}$ is a $TREE$ program and $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma \cdot (a, 1)]\} \models \exists s. Do(\delta_1^{tp'}, end[\sigma \cdot (a, 1)], s) \wedge Do(\delta', end[\sigma \cdot (a, 1)], s)$.

(b) if $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma \cdot (a, 0)]\}$ is consistent, then $T(\delta', \sigma \cdot (a, 0), \delta_2^{tp'})$, i.e., $\mathit{KHow}^{Ang}_{ECnd}(\delta', \sigma \cdot (a, 0))$ and $\delta_i^{tp'}$ is a $TREE$ program and $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma \cdot (a, 0)]\} \models \exists s. Do(\delta_2^{tp'}, end[\sigma \cdot (a, 0)], s) \wedge Do(\delta', end[\sigma \cdot (a, 0)], s)$.

Observe that since (δ, σ) may evolve to at least one $(\delta', \sigma \cdot (a, \mu)$ for some sensing outcome μ, either $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma \cdot (a, 0)]\}$ or $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma \cdot (a, 1)]\}$ is consistent.

Let us show that $T(\delta, \sigma, (a; \textbf{if } \phi \textbf{ then } \delta_1^{tp} \textbf{ else } \delta_2^{tp} \textbf{ endIf}))$ where ϕ is the condition on the right hand side of the sensed fluent axiom for a, and if $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma \cdot (a, 1)]\}$ is inconsistent then $\delta_1^{tp} = nil$ else $\delta_1^{tp} = \delta_1^{tp'}$ and if $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma \cdot (a, 0)]\}$ is inconsistent then $\delta_2^{tp} = nil$ else $\delta_2^{tp} = \delta_2^{tp'}$.

By the definition of $\mathit{KHow}^{Ang}_{ECnd}$, we have that $\mathit{KHow}^{Ang}_{ECnd}(\delta, \sigma)$ holds (again, notice that (δ, σ) may evolve at least to one configuration of the above form.)

Given the assumptions and the way δ_1^{tp} and δ_2^{tp} are defined, clearly $a; \textbf{if } \phi \textbf{ then } \delta_1^{tp} \textbf{ else } \delta_2^{tp} \textbf{ endIf}$ is a $TREE$ program.

It remains to show that

$$\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma]\} \models \exists s. Do((a; \textbf{if } \phi \textbf{ then } \delta_1^{tp} \textbf{ else } \delta_2^{tp} \textbf{ endIf}), s) \wedge Do(\delta, end[\sigma], s).$$

Pick a model M of $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma]\}$. Suppose M satisfies $SF(a, end[\sigma])$ and thus $\mathcal{D} \cup \mathcal{C} \cup Sensed[\sigma \cdot (a, 1)]$ is true in M. Then, due to (a) above, $Do(\delta', end[\sigma \cdot (a, 1)], s)$ must hold for some s in M. In addition, $Trans(\delta, end[\sigma], \delta', end[\sigma \cdot (a, 1)])$ is true in M, and we thus have $Do(\delta, end[\sigma], s)$ to be true in M. Also since

$$\begin{gathered} Do(\delta_1^{tp}, end[\sigma \cdot (a, 1)], s) \\ Trans(a; \textbf{if } \phi \textbf{ then } \delta_1^{tp} \textbf{ else } \delta_1^{tp'} \textbf{ endIf}, end[\sigma], nil; \textbf{if } \phi \textbf{ then } \delta_1^{tp} \textbf{ else } \delta_1^{tp'} \textbf{ endIf}, end[\sigma \cdot (a, 1)]) \\ \phi(end[\sigma \cdot (a, 1)]) \end{gathered}$$

are all true in M, we must, therefore, have $Do(a; \textbf{if } \phi \textbf{ then } \delta_1^{tp} \textbf{ else } \delta_1^{tp'} \textbf{ endIf}, end[\sigma], s)$ to holds in M.

Suppose, on the other hand, that M *does not* satisfy $SF(a, end[\sigma])$ and thus $\mathcal{D} \cup \mathcal{C} \cup Sensed[\sigma \cdot (a, 0)]$ is true in M. Then, due to (b) above, $Do(\delta', end[\sigma \cdot (a, 0)], s)$ must hold for some s in M. In addition, $Trans(\delta, end[\sigma], \delta', end[\sigma \cdot (a, 0)])$ is true in M, and we thus have $Do(\delta, end[\sigma], s)$ to be true in M. Also since

$$\begin{gathered} Do(\delta_2^{tp}, end[\sigma \cdot (a, 0)], s) \\ Trans(a; \textbf{if } \phi \textbf{ then } \delta_1^{tp} \textbf{ else } \delta_1^{tp'} \textbf{ endIf}, end[\sigma], nil; \textbf{if } \phi \textbf{ then } \delta_1^{tp} \textbf{ else } \delta_1^{tp'} \textbf{ endIf}, end[\sigma \cdot (a, 0)]) \\ \neg\phi(end[\sigma \cdot (a, 0)]) \end{gathered}$$

are all true in M, we must, therefore, have $Do(a; \textbf{if } \phi \textbf{ then } \delta_1^{tp} \textbf{ else } \delta_1^{tp'} \textbf{ endIf}, end[\sigma], s)$ to holds in M.

Therefore, $T(\delta, \sigma, a; \textbf{if } \phi \textbf{ then } \delta_1^{tp} \textbf{ else } \delta_2^{tp} \textbf{ endIf})$ applies.

Checking that $T(\delta, \sigma, \delta^{tp})$ is closed under assertion (C) of the definition of KHowBy_{EC}:

Assume that (δ, σ) may evolve to (δ', σ) w.r.t. the theory and that $T(\delta', \sigma, \delta^{tp'})$, i.e., $\mathit{KHow}^{Ang}_{ECnd}(\delta', \sigma)$ and $\delta^{tp'}$ is a $TREE$ program and $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma]\} \models \exists s. Do(\delta^{tp'}, end[\sigma], s) \wedge Do(\delta', end[\sigma], s)$.

Let us show that $T(\delta, \sigma, (True?; \delta^{tp'}))$. By the definition of $\mathit{KHow}^{Ang}_{ECnd}$, we have $\mathit{KHow}^{Ang}_{ECnd}(\delta, \sigma)$. Since $\delta^{tp'}$ is a $TREE$ program, so must be $True?; \delta^{tp'}$.

It remains to show that $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma]\} \models \exists s. Do((True?; \delta^{tp'}), end[\sigma], s) \wedge Do(\delta', end[\sigma], s)$. Pick a model of $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma]\}$. Since in this model $Do(\delta', end[\sigma], s)$ holds for some s and $Trans(\delta, end[\sigma], \delta', end[\sigma])$ holds too, then we it also holds $Do(\delta, end[\sigma], s)$ in the model in question. As well, since we have $Do(\delta^{tp'}, end[\sigma], s)$

and $Trans((True?;\delta^{tp'}), end[\sigma], nil; \delta^{tp'}, end[\sigma])$, we must also have $Do((True?;\delta^{tp'}), end[\sigma], s)$. Thus $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma]\} \models \exists s.Do((True?;\delta^{tp'}), end[\sigma], s) \wedge Do(\delta', end[\sigma], s)$. Therefore, $T(\delta, \sigma, (True?;\delta^{tp'}))$ applies.

PROOF OF THEOREM 4.2:
We shall prove a more general result w.r.t. *KHow*$^{Ang}_{ECnd}$. To that end, we add the following case to the definition of *KHowBy*$_{EC}$:

(C) if (δ, σ) *may evolve* to (δ', σ) *w.r.t. theory* $\mathcal{D}$ and $\mathcal{R}(\delta', \sigma, \delta^{tp'})$, then $\mathcal{R}(\delta, \sigma, (True?;\delta^{tp'}))$

In addition we enrich a bit the notion of $TREE$ programs to include two special test steps:

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

where a is any non-sensing action, and dpt_1 and dpt_2 are tree programs.

The proof goes by induction on *KHow*$^{Ang}_{ECnd}$ configurations. The property that we have to show is closed under the assertions in the definition of *KHow*$^{Ang}_{ECnd}$ (and hence be able to apply Lemma A.2) is the following one:

$$T(\delta, \sigma) = \text{ there exists } \delta^{tp} \text{ such that } \textit{KHowBy}_{EC}(\delta, \sigma, \delta^{tp}) \text{ holds.}$$

So we shall prove that $T(\delta, \sigma)$ is closed under (E1)-(E3) (that is, take T to be one of the possible $\mathcal{R}$ in the definition of *KHow*$^{Ang}_{ECnd}$.

Checking that $T(\delta, \sigma)$ is closed under assertion (E1) of the definition of *KHow*$^{Ang}_{ECnd}$:

Assume (δ, σ) is final. By definition of *KHowBy*$_{EC}$, case (A), we have *KHowBy*$_{EC}(\delta, \sigma, nil)$. Thus, $T(\delta, \sigma)$ applies.

Checking that $T(\delta, \sigma)$ is closed under assertion (E2) of the definition of *KHow*$^{Ang}_{ECnd}$:

Assume that (δ, σ) may evolve to configurations $(\delta', \sigma \cdot (a, \mu_i))$ (for some action a) w.r.t. theory $\mathcal{D}$ where $1 \leq i \leq 2$ and such that $T(\delta', \sigma \cdot (a, \mu_i))$ holds. Suppose that ϕ is the condition on the right hand side of the sensed fluent axiom for a, that is, $SF(a, s) \equiv \phi(s) \in \mathcal{D}$. Since (δ, σ) must evolve to *at least one* configuration (possibly two), then $\mathcal{D} \cup \mathcal{C}\{Sensed[\sigma]\}$ is consistent and, therefore, either $\mathcal{D} \cup \mathcal{C}\{Sensed[\sigma \cdot (a, 0)]\}$ or $\mathcal{D} \cup \mathcal{C}\{Sensed[\sigma \cdot (a, 1)]\}$ is consistent.

If (δ, σ) may evolve to $(\delta', \sigma \cdot (a, 1))$, then $\mathcal{D} \cup \mathcal{C}\{Sensed[\sigma \cdot (a, 1)]\}$ ought to be consistent, and, hence, $T(\delta', \sigma \cdot (a, 1))$, i.e., there exists $\delta_1^{tp'}$ such that *KHowBy*$_{EC}(\delta', \sigma \cdot (a, 1), \delta_1^{tp'})$.

Simlarly, if (δ, σ) may evolve to $(\delta', \sigma \cdot (a, 0))$, then $\mathcal{D} \cup \mathcal{C}\{Sensed[\sigma \cdot (a, 0)]\}$ ought to be consistent, and, hence, $T(\delta', \sigma \cdot (a, 0))$, i.e. there exists $\delta_2^{tp'}$ such that *KHowBy*$_{EC}(\delta', \sigma \cdot (a, 0), \delta_2^{tp'})$.

Let $\delta^{tp} = (a; \textbf{if } \phi \textbf{ then } \delta_1^{tp} \textbf{ else } \delta_2^{tp} \textbf{ endIf})$, and if $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma \cdot (a, 1)]\}$ is inconsistent then $\delta_1^{tp} = nil$ else $\delta_1^{tp} = \delta_1^{tp'}$ and if $\mathcal{D} \cup \mathcal{C} \cup \{Sensed[\sigma \cdot (a, 0)]\}$ is inconsistent then $\delta_2^{tp} = nil$ else $\delta_2^{tp} = \delta_2^{tp'}$.

By the definition of *KHowBy*$_{EC}$, case (B), we have that *KHowBy*$_{EC}(\delta, \sigma, \delta^{tp})$. Thus, $T(\delta, \sigma)$ applies.

Checking that $T(\delta, \sigma)$ is closed under assertion (E3) of the definition of *KHow*$^{Ang}_{ECnd}$:

Assume that (δ, σ) may evolve to (δ', σ) w.r.t. $\mathcal{D}$ and that $T(\delta', \sigma)$, i.e., there exists $\delta^{tp'}$ such that *KHowBy*$_{EC}(\delta', \sigma, \delta')$. By the definition of *KHowBy*$_{EC}$, case (C), we have *KHowBy*$_{EC}(\delta, \sigma, (True?;\delta^{tp'}))$. Thus, $T(\delta, \sigma)$ applies taking $\delta^{tp} = (True?;\delta^{tp'})$.