Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

Abstraction of Agents Executing Online and their Abilities in the Situation Calculus

Bita Banihashemi[1], **Giuseppe De Giacomo**[2], **Yves Lespérance**[1]
[1] York University
[2] Sapienza Università di Roma
bita@cse.yorku.ca, degiacomo@dis.uniroma1.it, lesperan@cse.yorku.ca

Abstract

We develop a general framework for abstracting online behavior of an agent that may acquire new knowledge during execution (e.g., by sensing), in the situation calculus and ConGolog. We assume that we have both a high-level action theory and a low-level one that represent the agent's behavior at different levels of detail. In this setting, we define ability to perform a task/achieve a goal, and then show that under some reasonable assumptions, if the agent has a strategy by which she is able to achieve a goal at the high level, then we can refine it into a low-level strategy to do so.

1 Introduction

One way to effectively support reasoning about agents with complex behavior is to use *abstraction*. In essence, this involves developing an abstract model of the agent/domain that suppresses less important details. The abstract model allows us to reason more easily about the agent's possible behaviors and to provide high-level explanations of the agent's behavior. In AI, abstraction has been investigated in variety of settings such as to improve efficiency in planning (e.g., [Baier and McIlraith, 2006; Kuter *et al.*, 2007]), facilitate verification (e.g., [Mo *et al.*, 2016]), and capture the social practices of multi-agent systems (e.g., [Dignum *et al.*, 2015]).

[Banihashemi *et al.*, 2017] proposed a general framework for *agent abstraction* based on the situation calculus [McCarthy and Hayes, 1969; Reiter, 2001] and the ConGolog agent programming language [De Giacomo *et al.*, 2000]. They assume that one has a high-level/abstract action theory, a low-level/concrete action theory, both representing the agent's behavior at different levels of detail, and a *refinement mapping* between the two. The mapping associates each high-level primitive action to a ConGolog program defined over the low-level action theory that "implements it". Moreover, it maps each high-level fluent to a state formula in the low-level language that characterizes the concrete conditions under which it holds. They then define a notion of a high-level theory being a *sound abstraction* of a low-level theory under a given refinement mapping. The formalization involves the existence of a suitable bisimulation relation [Milner, 1971; 1989] between models of the low-level and high-level theories. It is shown that sound abstractions have many useful properties that ensure that one can reason about the agent's actions (e.g. executability, projection, and planning) at the abstract level, and refine and concretely execute them at the low level.

In this paper, we study how we can apply this framework in the case where the agent is executing *online* [De Giacomo and Levesque, 1999; Sardiña *et al.*, 2004b], i.e., may acquire new knowledge while executing (e.g., by sensing). This means that the knowledge base that the agent uses in her reasoning needs to be updated during the execution.

To formalize a notion of *sound abstraction in online executions*, we first identify a sufficient property for *sound abstraction to persist in online executions*. We then show that a key result about sound abstractions extends to online executions. We also adapt definitions of strategies and ability to perform a task/achieve a goal [Moore, 1985; Lespérance *et al.*, 2000; 2008] to our model of online execution. Then, we show our main result, namely that under some reasonable assumptions, if we have a sound abstraction and the agent *has a strategy by which she is able to achieve a goal at the high level, then one can refine it into a low-level strategy* by which the agent is able to achieve the refinement of the goal. Moreover, the low-level strategy can be obtained piecewise/incrementally, by finding a refinement of each step of the high-level strategy individually. This makes reasoning about agents' abilities much easier.

Let us briefly discuss related work. [Giunchiglia and Walsh, 1992; Nayak and Levy, 1995] formalize abstraction of static logical theories; instead, our work focuses on abstraction of dynamic domains where the agent may acquire new information as it executes. Other related work focuses on planning, for instance [Gabaldon, 2002], which studies encodings of HTNs in ConGolog with enhanced features like exogenous actions and online executions, and [Baier and McIlraith, 2006], which investigates planning with sensing and complex/macro actions specified in Golog. The former uses a single basic action theory (BAT), while the latter compiles the abstracted actions into a new BAT that includes both the original and abstracted actions. Our approach is more general and allows fluents to be abstracted as well.

In the next section, we outline the basics of the Situation Calculus and ConGolog. Then in Section 3, we review the

agent abstraction framework of [Banihashemi *et al.*, 2017]. Following that in Section 4, we discuss how we model sensing, exogenous actions, and online execution. Then in Section 5, we identify sufficient conditions for sound abstractions to persist in online executions and examine their properties. Following that in Section 6, we formalize a suitable notion of agent ability. Then in Section 7, we present our main result, i.e., that under a reasonable set of assumptions, when an agent has a strategy by which she is able to achieve a goal at the high level, we can refine it into a low-level strategy to do so. The paper then concludes by discussing our contributions and topics for future work.

2 Preliminaries

Situation Calculus. The *situation calculus* is a well known predicate logic language for representing and reasoning about dynamically changing worlds. Within the language, one can formulate *basic action theories* (BATs) that describe how the world changes as a result of actions; see [Reiter, 2001] for details of how these are defined. Hereafter, we will use $\mathcal{D}$ to refer to the BAT under consideration. We assume that there is a *finite number of action types* $\mathcal{A}$. Moreover, we assume that the terms of object sort are in fact a countably infinite set $\mathcal{N}$ of standard names for which we have the unique name assumption and domain closure. For simplicity, and w.l.o.g., we assume that there are no functions other than constants and no non-fluent predicates. We write $do([a_1, a_2, \ldots, a_{n-1}, a_n], s)$ as an abbreviation for the situation term $do(a_n, do(a_{n-1}, \ldots, do(a_2, do(a_1, s)) \ldots))$; for an action sequence $\vec{a}$, we often write $do(\vec{a}, s)$ for $do([\vec{a}], s)$. A special predicate $\mathit{Poss}(a, s)$ is used to state that action a is executable in situation s. The abbreviation $Executable(s)$ means that every action performed in reaching situation s was executable in the situation in which it occurred.

ConGolog. To represent and reason about complex actions or processes obtained by suitably executing atomic actions, various so-called *high-level programming languages* have been defined [Levesque *et al.*, 1997]. Here we concentrate on (a variant of) ConGolog [De Giacomo *et al.*, 2000] that includes the following constructs:

$$\delta ::= nil \mid \alpha \mid \varphi? \mid \delta_1; \delta_2 \mid \delta_1 | \delta_2 \mid \pi x.\delta \mid \delta^* \mid \delta_1 \| \delta_2$$

In the above, nil denotes the empty program, which does nothing and is already terminated, α is an action term, possibly with parameters, and φ is situation-suppressed formula, i.e., a formula with all situation arguments in fluents suppressed. As usual, we denote by $\varphi[s]$ the formula obtained from φ by restoring the situation argument s into all fluents in φ. The sequence of program δ_1 followed by program δ_2 is denoted by $\delta_1; \delta_2$. Program $\delta_1 | \delta_2$ allows for the nondeterministic choice between programs δ_1 and δ_2, while $\pi x.\delta$ executes program δ for *some* nondeterministic choice of a legal binding for object variable x (observe that such a choice is, in general, unbounded). δ^* performs δ zero or more times. Program $\delta_1 \| \delta_2$ expresses the concurrent execution (interpreted as interleaving) of programs δ_1 and δ_2.

Formally, the semantics of ConGolog is specified in terms of single-step transitions, using the following two predicates: *(i)* $Trans(\delta, s, \delta', s')$, which holds if one step of program δ in situation s may lead to situation s' with δ' remaining to be executed; and *(ii)* $Final(\delta, s)$, which holds if program δ may legally terminate in situation s. The definitions of $Trans$ and $Final$ we use are as in [De Giacomo *et al.*, 2010]; differently from [De Giacomo *et al.*, 2000], the test construct $\varphi?$ does not yield any transition, but is $Final$ when satisfied. Predicate $Do(\delta, s, s')$ means that program δ, when executed starting in situation s, has s' as a legal terminating situation, and is defined as $Do(\delta, s, s') \doteq \exists \delta'. Trans^*(\delta, s, \delta', s') \wedge Final(\delta', s')$ where *Trans** denotes the reflexive transitive closure of *Trans*. In the rest, we use $\mathcal{C}$ to denote the axioms defining the ConGolog programming language.

Situation-Determined Programs. A ConGolog program δ is *situation-determined* (SD) in a situation s [De Giacomo *et al.*, 2012] if for every sequence of actions, the remaining program is determined by the resulting situation, i.e.,

$$\mathit{SituationDetermined}(\delta, s) \doteq \forall s', \delta', \delta''. \\ \mathit{Trans}^*(\delta, s, \delta', s') \wedge \mathit{Trans}^*(\delta, s, \delta'', s') \supset \delta' = \delta'',$$

For instance, the program $a; (b \mid c)$ (assuming the actions involved are always executable) is SD in situation S_0, since there is a unique remaining program $(b \mid c)$ in $do(a, S_0)$ (and similarly for the other reachable situations). However, the program $(a; b) \mid (a; c)$ is not SD in situation S_0, since after performing action a, and given only the situation $do(a, S_0)$, it is impossible to determine what the remaining program is (it could be either b or c).

3 Abstracting Agent Behavior

In the agent abstraction framework of [Banihashemi *et al.*, 2017], there is a high-level (HL) action theory $\mathcal{D}_h$ and a low-level (LL) action theory $\mathcal{D}_l$ representing the agent's possible behaviors at different levels of detail. $\mathcal{D}_h$ (resp. $\mathcal{D}_l$) involves a finite set of primitive action types $\mathcal{A}_h$ (resp. $\mathcal{A}_l$) and a finite set of primitive fluent predicates $\mathcal{F}_h$ (resp. $\mathcal{F}_l$). Also, $\mathcal{D}_h$ and $\mathcal{D}_l$ are assumed to share no domain specific symbols except for the set of standard names for objects $\mathcal{N}$.

Refinement Mapping. To relate the two theories, a *refinement mapping* m is defined as a function that associates each high-level primitive action type A in $\mathcal{A}_h$ to a situation-determined ConGolog program δ_A defined over the low-level theory that implements the action, i.e., $m(A(\vec{x})) = \delta_A(\vec{x})$; moreover, m maps each situation-suppressed high-level fluent $F(\vec{x})$ in $\mathcal{F}_h$ to a situation-suppressed formula $\phi_F(\vec{x})$ defined over the low-level theory that characterizes the concrete conditions under which $F(\vec{x})$ holds in a situation. We extend the notation so that $m(\phi)$ stands for the result of substituting every fluent $F(\vec{x})$ in situation-suppressed formula ϕ by $m(F(\vec{x}))$. Also, we apply m to action sequences with $m(\alpha_1, \ldots, \alpha_n) \doteq m(\alpha_1); \ldots; m(\alpha_n)$ for $n \geq 1$ and $m(\epsilon) \doteq nil$, where ϵ is the empty sequence of actions.

m-Bisimulation. To relate the high-level and low-level models/theories, a variant of bisimulation [Milner, 1971; 1989] is defined as follows. Given M_h a model of $\mathcal{D}_h \cup \mathcal{C}$, and M_l a model of $\mathcal{D}_l \cup \mathcal{C}$, a relation $B \subseteq \Delta_S^{M_h} \times \Delta_S^{M_l}$ (where Δ_S^M stands for the situation domain of M) is an *m-bisimulation relation between M_h and M_l* if $\langle s_h, s_l \rangle \in B$ implies that: (i) $s_h \sim_m^{M_h,M_l} s_l$, i.e., s_h and s_l evaluate each high-level primitive fluent the same;[1] (ii) for every high-level primitive action type A in $\mathcal{A}_h$, if there exists s_h' such that $M_h \models Poss(A(\vec{x}), s_h) \wedge s_h' = do(A(\vec{x}), s_h)$, then there exists s_l' such that $M_l \models Do(m(A(\vec{x})), s_l, s_l')$, and $\langle s_h', s_l' \rangle \in B$; and, (iii) for every high-level primitive action type A in $\mathcal{A}_h$, if there exists s_l' such that $M_l \models Do(m(A(\vec{x})), s_l, s_l')$, then there exists s_h' such that $M_h \models Poss(A(\vec{x}), s_h) \wedge s_h' = do(A(\vec{x}), s_h)$ and $\langle s_h', s_l' \rangle \in B$. M_h is *m-bisimilar* to M_l, written $M_h \sim_m M_l$, if and only if there exists an m-bisimulation relation B between M_h and M_l such that $(S_0^{M_h}, S_0^{M_l}) \in B$.

Sound Abstractions. In [Banihashemi *et al.*, 2017], $\mathcal{D}_h$ is a *sound abstraction of $\mathcal{D}_l$ relative to refinement mapping m* if and only if, for all models M_l of $\mathcal{D}_l \cup \mathcal{C}$, there exists a model M_h of $\mathcal{D}_h \cup \mathcal{C}$ such that $M_h \sim_m M_l$. With a sound abstraction, whenever the high-level theory *entails* that a sequence of actions is executable and achieves a certain condition, then the low level must also entail that there exists an executable refinement of the sequence such that the "translated" condition holds afterwards. Moreover, whenever the low level considers the executability of a refinement of a high-level action to be satisfiable, then the high level does also. Sound abstractions can be used to perform effectively several forms of reasoning about action, such as planning, monitoring, and generating high-level explanations of low-level behavior. Note that a dual notion is also defined: $\mathcal{D}_h$ is a *complete abstraction of $\mathcal{D}_l$ relative to refinement mapping m* if and only if, for all models M_h of $\mathcal{D}_h \cup \mathcal{C}$, there exists a model M_l of $\mathcal{D}_l \cup \mathcal{C}$ such that $M_l \sim_m M_h$.

Example 1 For our running example, we consider a travel planner agent that can book a seat (economy Ec or business Bz) on a flight f for a customer c. The high-level BAT $\mathcal{D}_h^{eg}$ abstracts over details of the booking procedure. The fluent $Upg(c, s)$ holds when an upgrade for customer c is possible. The action $book(c, cls, f)$ (where cls denotes either Bz or Ec) is always possible for economy seats but only possible for business seats when $Upg(c, s)$ holds. The fluent $Booked(c, cls, f, s)$ indicates a ticket has been booked for c. The low-level BAT $\mathcal{D}_l^{eg}$ models the process of booking a ticket in more detail. Two types of upgrades exist: $AirMiles(c, s)$ and $Promotion(c, s)$. Action $selFlt(c, cls, f)$ to select a flight is only possible for Bz seats if either of $AirMiles(c, s)$ or $Promotion(c, s)$ holds and action $pay(c, cls, f)$ makes payments. Fluents $Selected(c, cls, f, s)$ and $Paid(c, cls, f, s)$ indicate the ticket has been selected and paid for respectively. The refinement mapping m^{eg} is defined as:

$m^{eg}(Upg(c)) = AirMiles(c) \vee Promotion(c)$
$m^{eg}(Booked(c, cls, f)) = Selected(c, cls, f) \wedge Paid(c, cls, f)$
$m^{eg}(book(c, cls, f)) = selFlt(c, cls, f); pay(c, cls, f)$ □

[1] This defines a local condition for the bisimulation. We say that situation s_h in M_h is m-isomorphic to situation s_l in M_l, written $s_h \sim_m^{M_h,M_l} s_l$, if and only if $M_h, v[s/s_h] \models F(\vec{x}, s)$ iff $M_l, v[s/s_l] \models m(F(\vec{x}))[s]$ for every high-level primitive fluent $F(\vec{x})$ in $\mathcal{F}_h$ and every variable assignment v ($v[x/e]$ stands for the assignment that is like v except that x is mapped to e).

4 Online Execution and Sensing

In this paper, similarly to [Lespérance *et al.*, 2008], we model sensing as an ordinary action which queries a sensor, followed by the reporting of a sensor result, in the form of an exogenous action. With this way of doing sensing, we essentially store the sensing results in the situation (which includes all actions executed so far including the exogenous actions used for sensing). We update the theory/knowledge base (KB) to incorporate the sensed information by asserting $Poss(a, s)$ atoms and relying on precondition axioms to relate sensing to fluents. In particular the current KB after having performed the sequence of actions $\vec{a}$ is $\mathcal{D} \cup \mathcal{C} \cup \{Executable(do(\vec{a}, S_0)\}$, which we abbreviate as $\mathcal{D} \cup \mathcal{C} \cup$ *Sensed*$(\vec{a})$. Note that this also handles the agent's acquiring knowledge from an arbitrary exogenous action.

Example 2 To allow the high-level agent in Example 1 to learn if $Upg(c, s)$ holds, $\mathcal{D}_h^{eg}$ can include the following:

$$Poss(qryUpg(c), s) \equiv \neg\exists c'.(QrdUpg(c', s) \wedge \neg RecvRep(c', s))$$
$$Poss(repUpg(c, x), s) \equiv QrdUpg(c, s) \wedge \neg RecvRep(c, s) \wedge (Upg(c, s) \wedge x = 1 \vee \neg Upg(c, s) \wedge x = 0)$$

where $qryUpg(c)$ is an ordinary action that is used to query (i.e., sense) if $Upg(c, s)$ holds and $repUpg(c, x)$ is an exogenous action that informs the agent if $Upg(c, s)$ holds through its precondition axiom; $repUpg(c, 1)$ is executed if $Upg(c, s)$ holds, and otherwise, $repUpg(c, 0)$ is executed. The fluent $QrdUpg(c, s)$ (resp. $RecvRep(c, s)$) in the precondition indicates whether action $qryUpg(c)$ (resp. $repUpg(c, x)$) has already been performed in s. (The low-level BAT can be extended in a similar way to support sensing whether upgrades are possible through air miles and promotions.) □

We model an *online agent configuration* (c) as a pair $\langle \delta, \vec{a} \rangle$, where δ is the remaining program and $\vec{a}$ is the sequence of actions performed so far starting from S_0. The initial configuration c^i is $\langle \delta^i, \epsilon \rangle$. [Banihashemi *et al.*, 2016], define the *online transition relation* between configurations as follows:

$\langle \delta, \vec{a} \rangle \rightarrow_{A(\vec{n})} \langle \delta', \vec{a}A(\vec{n}) \rangle$ if and only if
either $A \in \mathcal{A}^o$, $\vec{n} \in \mathcal{N}^k$ and
$\mathcal{D} \cup \mathcal{C} \cup$ *Sensed*$(\vec{a}) \models$
$Trans(\delta, do(\vec{a}, S_0), \delta', do(A(\vec{n}), do(\vec{a}, S_0)))$
or $A \in \mathcal{A}^e$, $\vec{n} \in \mathcal{N}^k$ and $\mathcal{D} \cup \mathcal{C} \cup$ *Sensed*$(\vec{a}) \cup$
$\{Trans(\delta, do(\vec{a}, S_0), \delta', do(A(\vec{n}), do(\vec{a}, S_0)))\}$ is satisfiable,

where $\mathcal{A}^o$ (resp. $\mathcal{A}^e$) represents the ordinary (resp. exogenous) set of action types. Here, $\langle \delta, \vec{a} \rangle \rightarrow_{A(\vec{n})} \langle \delta', \vec{a}A(\vec{n}) \rangle$ means that configuration $\langle \delta, \vec{a} \rangle$ can make a single-step online transition to configuration $\langle \delta', \vec{a}A(\vec{n}) \rangle$ by performing action

$A(\vec{n})$. If $A(\vec{n})$ is an ordinary action, the agent must know that the action is executable and know what the remaining program is afterwards. If $A(\vec{n})$ is an exogenous action, the agent need only think that the action may be possible with δ' being the remaining program. The relation $c \rightarrow_{\vec{a}}^{*} c'$ is the reflexive-transitive closure of $c \rightarrow_{A(\vec{n})} c'$ denoting that online configuration c' can be reached from configuration c by performing a sequence of online transitions involving the sequence of actions $\vec{a}$. A (meta-theoretic) predicate $c\checkmark$, meaning that configuration c is known to be final, is defined as:

$\langle \delta, \vec{a} \rangle \checkmark$ if and only if $\mathcal{D} \cup \mathcal{C} \cup \mathit{Sensed}(\vec{a}) \models Final(\delta, do(\vec{a}, S_0))$.

5 Sound Abstraction in Online Executions

How can we use abstraction in agents that execute online and acquire new information during a run? The agent's theory/KB is updated when it gets new information. The first question is whether a sound abstraction remains so when that happens.

Example 3 Suppose that we have a high-level exogenous action a_h that is executable if and only if P_h holds. Let $m(P_h) = P_l$ and $m(a_h) = P_l?; a_l$, where a_l is a low-level exogenous action. Moreover, assume that at the low level, a_l is always executable. Initially, it is unknown whether P_h holds at the high level and similarly for P_l at the low level. It is easy to check that $\mathcal{D}_h$ is a sound abstraction of $\mathcal{D}_l$ with respect to the mapping m. However, if a_h and its refinement a_l occur and the theories are updated, then $\mathcal{D}_h \cup \{Poss(a_h, S_0)\}$ is no longer a sound abstraction of $\mathcal{D}_l \cup \{Poss(a_l, S_0)\}$ wrt m. In $do(a_h, S_0)$, the high-level agent has learned that P_h holds, as $Poss(a_h, S_0)$ has been added to the theory. However, at the low level, adding $Poss(a_l, S_0)$ has no effect and in $do(a_l, S_0)$, it is still unknown whether P_l holds. The updated low-level theory has a model where $\neg P_l$ holds, which has no m-bisimilar model in the updated high-level theory. □

Thus a sound abstraction does not always remain so when we update the theories after an action is executed online. For the sound abstraction to persist, we need to ensure that the low level acquires as much information as the high level, e.g., ensure that the low level learns that $m(P_h)$, i.e., P_l, holds. When a high-level action a occurs, the high level learns that a was executable. If the low level also learns that it has in fact just executed a refinement of a, then we can be certain that it has acquired as much information as the high level. We can generalize this to action sequences and prove that it is a sufficient condition for the sound abstraction to persist:[2]

Theorem 1 ***(Online Persistence of Sound Abstractions)*** *If $\mathcal{D}_h$ is a sound abstraction of $\mathcal{D}_l$ relative to mapping m and $\mathcal{D}_l \cup \mathcal{C} \cup \mathit{Sensed}(\vec{a}) \models Do(m(\vec{\alpha}), S_0, do(\vec{a}, S_0))$, then $\mathcal{D}_h \cup \mathit{Sensed}(\vec{\alpha})$ is a sound abstraction of $\mathcal{D}_l \cup \mathit{Sensed}(\vec{a})$ wrt m.*

Here, the condition $\mathcal{D}_l \cup \mathcal{C} \cup \mathit{Sensed}(\vec{a}) \models Do(m(\vec{\alpha}), S_0, do(\vec{a}, S_0))$ ensures that after executing a refinement $\vec{a}$ of high-level action sequence $\vec{\alpha}$, the low level knows it has just executed a refinement of $\vec{\alpha}$, and thus it has learned as much as the high level did, and we still have a sound abstraction.[3]

We can use the above to extend the results of [Banihashemi *et al.*, 2017] on sound abstractions so that the agent gets to use the knowledge acquired in an online execution. We can show that if after executing action sequence $\vec{\alpha}$ online the high level *knows* that it can execute action sequence $\vec{\beta}$ to achieve condition ϕ, then after executing a refinement $\vec{a}$ of $\vec{\alpha}$ online, being aware that $\vec{a}$ is a refinement of $\vec{\alpha}$, the low level also *knows* that there exists a refinement of $\vec{\beta}$ that achieves $m(\phi)$:

Proposition 2 *Suppose that $\mathcal{D}_h$ is a sound abstraction of $\mathcal{D}_l$ relative to mapping m and $\mathcal{D}_l \cup \mathcal{C} \cup \mathit{Sensed}(\vec{a}) \models Do(m(\vec{\alpha}), S_0, do(\vec{a}, S_0))$ for some ground high-level action sequence $\vec{\alpha}$ and ground low-level action sequence $\vec{a}$. Then for any high-level ground action sequence $\vec{\beta}$ and situation-suppressed formula ϕ,*

if $\mathcal{D}_h \cup \mathit{Sensed}(\vec{\alpha}) \models Executable(do(\vec{\alpha}\vec{\beta}, S_0)) \wedge \phi[do(\vec{\alpha}\vec{\beta}, S_0)]$,
then $\mathcal{D}_l \cup \mathcal{C} \cup \mathit{Sensed}(\vec{a}) \models \exists s. Do(m(\vec{\beta}), do(\vec{a}, S_0), s) \wedge m(\phi)[s]$.

Now, if we had complete information and a deterministic environment, then we could use this proposition to extract a plan at the low level to realize high-level action sequence $\vec{\beta}$ and achieve ϕ. However neither of these two conditions hold in the case we are studying. The agent has *incomplete information* and uses sensing to acquire the knowledge she needs. Moreover the environment is nondeterministic and produces *exogenous actions* that are not under the control of the agent. For these reasons, we cannot make use of this proposition. Instead, for the agent to be *able* to achieve a goal, she needs to have a *strategy* that ensures achieving the goal no matter how the environment behaves and how sensing turns out. To address this, we next develop an account of agent ability. Subsequently, we return to how abstraction can be exploited to find strategies and ensure ability.

6 Agent Ability

Intuitively, an agent is *able to perform a task/achieve a goal* if she can always choose an action that eventually leads to successful completion of the task/achievement of the goal no matter how the environment behaves and what information she finds out; i.e., she needs to have a *strategy* that she can follow to successfully complete the task/achieve the goal, where the strategy specifies how she should continue to act after the environment performs some action in response to what has occurred so far. Note that we assume that environment actions are fully observable. Ability is similar to the concept of *conditional* or *contingent planning* [Cimatti *et al.*, 2003; Alford *et al.*, 2009; Geffner and Bonet, 2013], where agents operating online in dynamic and incompletely known environments need to construct plans/strategies that prescribe different behaviors depending on new information acquired

[2] For proofs of our results, see [Banihashemi *et al.*, 2018a].

[3] This condition does not seem too difficult to guarantee in practice. Essentially, we must ensure that whenever a test succeeds in a refinement of a high-level action, the low level knows or learns the test was satisfied. E.g., we can redefine $m(a_h) = P_l?; confirm_{P_l}; a_l$, where $confirm_{P_l}$ is a new exogenous action that has P_l as precondition and no effects; this ensures that when $m(a_h)$ is executed, the agent learns that $m(P_h)$, i.e., P_l, holds.

(e.g., as a result of sensing) to ensure they achieve their goals/execute their tasks.

Turn Taking. To formalize ability, we need to specify when the agent/environment can act. We do this by enforcing turn taking between the agent and the environment. We assume that in any situation, either it is the agent's turn and all the executable actions are ordinary, or it is the environment's turn and all the executable actions are exogenous:[4]

Assumption 1 (Turn Taking) *For $\mathcal{D} \in \{\mathcal{D}_h, \mathcal{D}_l\}$, we have*

$$\mathcal{D} \models \forall s.\neg[(\bigvee_{A \in \mathcal{A}^e} \exists \vec{x}.Poss(A(\vec{x}), s)) \wedge (\bigvee_{A \in \mathcal{A}^o} \exists \vec{x}.Poss(A(\vec{x}), s))]$$

Furthermore, we assume that in any online configuration,[5] it is *known* whether it is the agent's or the environment's turn:

Assumption 2 (Always Known Whose Turn It Is) *For $\mathcal{D} \in \{\mathcal{D}_h, \mathcal{D}_l\}$ and for all ground sequences $\vec{a}$ such that $\langle (\pi a.a)^*, \epsilon\rangle \rightarrow_{\vec{a}}^* \langle \delta, \vec{a}\rangle$ and $\langle \delta, \vec{a}\rangle\checkmark$ for some δ, we have that*

$$\textit{either } \mathcal{D} \cup \mathcal{C} \cup \textit{Sensed}(\vec{a}) \models \bigvee_{A \in \mathcal{A}^o} \exists \vec{x}.Poss(A(\vec{x}), s)$$
$$\textit{or } \mathcal{D} \cup \mathcal{C} \cup \textit{Sensed}(\vec{a}) \models \bigvee_{A \in \mathcal{A}^e} \exists \vec{x}.Poss(A(\vec{x}), s).$$

Able By. Here, we will restrict attention to bounded-length strategies.[6] We represent *(bounded) strategies* γ as a restricted form of program using the following syntax:

$$\gamma ::= nil \mid [\alpha^o; \gamma] \mid \mathbf{set}(E)$$
where E is a non-empty set of programs of the form $[\beta_i^e; \gamma_i]$.

Here, α^o ranges over ordinary action terms, β_i^e over exogenous action terms, and γ_i over strategies, and $\mathbf{set}(\cdot)$ is an infinitary version of nondeterministic branch [De Giacomo *et al.*, 2012]. Thus nil is the strategy that does nothing and stops, $[\alpha^o; \gamma]$ represents the strategy where the agent does action α^o and then follows strategy γ, and $\mathbf{set}([\beta_1^e; \gamma_1], [\beta_2^e; \gamma_2], \ldots)$ with distinct actions $\beta_1^e, \beta_2^e, \ldots$ represents the strategy where exogenous action β_1^e may occur after which γ_1 is followed, exogenous action β_2^e may occur after which γ_2 is followed, etc.; there may be a finite or countably infinite set of such pairs $[\beta_i^e; \gamma_i]$.

We adapt [Lespérance *et al.*, 2008] and define $AbleBy(\delta, \vec{a}, \gamma)$, meaning that the agent *is able to successfully perform a task* represented by an *online situation-determined program*[7] δ in an environment that behaves as specified by δ in situation $do(\vec{a}, S_0)$ by executing the strategy γ, to be the smallest relation $\mathcal{R}(\delta, \vec{a}, \gamma)$ such that:

(A) for all pairs $(\delta, \vec{a})$, if $\langle \delta, \vec{a}\rangle\checkmark$, then $\mathcal{R}(\delta, \vec{a}, nil)$;

(B) for all $\delta, \vec{a}$, if there exists a, δ' such that $a \in \mathcal{A}^o$ and $\langle \delta, \vec{a}\rangle \rightarrow_a \langle \delta', \vec{a}a\rangle$ and $\mathcal{R}(\delta', \vec{a}a, \gamma)$, then $\mathcal{R}(\delta, \vec{a}, [a; \gamma])$;

(C) for all $\delta, \vec{a}$, if there exists a, δ' such that $a \in \mathcal{A}^e$ and $\langle \delta, \vec{a}\rangle \rightarrow_a \langle \delta', \vec{a}a\rangle$ and for all a, δ' such that $\langle \delta, \vec{a}\rangle \rightarrow_a \langle \delta', \vec{a}a\rangle$ there exists γ such that $\mathcal{R}(\delta', \vec{a}a, \gamma)$, then $\mathcal{R}(\delta, \vec{a}, \mathbf{set}(E))$ where $E = \{[a; \gamma] \mid \langle \delta, \vec{a}\rangle \rightarrow_a \langle \delta', \vec{a}a\rangle$ and $\mathcal{R}(\delta', \vec{a}a, \gamma)$ for some $\delta'\}$

That is, (A) if δ is final in situation $do(\vec{a}, S_0)$, then the agent is able to execute δ in situation $do(\vec{a}, S_0)$ by performing the empty strategy (nil); (B) if there is an ordinary action a that the agent can execute online to get from the current configuration $\langle \delta, \vec{a}\rangle$ to some configuration $\langle \delta', \vec{a}a\rangle$ after which the agent is able to execute δ' in situation $do(\vec{a}a, S_0)$ by following strategy γ, then the agent is able to execute δ in situation $do(\vec{a}, S_0)$ by following strategy $[a; \gamma]$; (C) if there is some exogenous action a that can be executed online from $\langle \delta, \vec{a}\rangle$ and for every exogenous action a that can be executed online from the current configuration $\langle \delta, \vec{a}\rangle$ to some configuration $\langle \delta', \vec{a}a\rangle$ there exists a strategy γ such that the agent is then able to execute δ' in situation $do(\vec{a}a, S_0)$ by following strategy γ, then the agent is able to execute δ in situation $do(\vec{a}, S_0)$ by following strategy $\mathbf{set}(E)$, where E is the set of all such pairs $[a; \gamma]$. Note that the task of simply achieving a goal ϕ (by performing some arbitrary action sequence) can be represented by the program $Achieve(\phi) \doteq (\pi a.a)^*; \phi?$.

Example 4 Returning to our running example, consider the task $\delta_{h_1} = Achieve(\phi_{h1})$, where $\phi_{h1} = (Upg(C1) \wedge Booked(C1, Bz, F1)) \vee (\neg Upg(C1) \wedge Booked(C1, Ec, F1))$, i.e., book a business seat for customer $C1$ on flight $F1$ if an upgrade is available and otherwise book an economy seat. Then at the high level, we have $AbleBy(\delta_{h_1}, \epsilon, \gamma_{h_1})$ with the strategy $\gamma_{h_1} = qryUpg(C1); \mathbf{set}((repUpg(C1, 1); book(C1, Bz, F1); nil), (repUpg(C1, 0); book(C1, Eco, F1); nil))$. □

7 Hierarchical Refinement of Abilities

The notions of ability and strategy we have defined can be used at both the high and low levels. If we have a strategy at the high level that the agent can follow to perform a task or achieve a goal, under what conditions can we obtain a strategy at the low level that refines this high-level strategy and that the agent can follow to perform a refinement of the high-level task or achieve a refinement of the high-level goal? What assumptions are sufficient to ensure we can always do this?

High-Level Monitorability. We assume the agent only executes low-level action sequences that refine some high-level action sequence, i.e., the agent is *high-level monitorable*.

Assumption 3 (All LL behaviors refine HL actions)

$$\mathcal{D}_l \cup \mathcal{C} \models \forall s.Executable(s) \supset \exists \delta.Trans^*(\text{ANYSEQHL}, S_0, \delta, s)$$
where $\text{ANYSEQHL} \doteq (|_{A_i \in \mathcal{A}_h} \pi \vec{x}.m(A_i(\vec{x})))^*$,
i.e., do any sequence of refinements of high-level actions.

Without high-level monitorability, at the concrete level, there is nothing preventing the environment from performing an action that is not part of any refinement of a high-level action

[4] If at some point, the "player" whose turn it is has the option of doing nothing, we can have it execute a "no-op" action.

[5] Note that $(\pi a.a)^*$ is a program that repeatedly executes a nondeterministically chosen primitive action zero or more times; thus it can perform any sequence of executable actions.

[6] For more general types of tasks that require unbounded strategies, one can use an approach like [Sardiña *et al.*, 2004a; 2006].

[7] A program is *online situation-determined* if for any sequence of actions that the agent can perform online, the resulting agent configuration is unique [Banihashemi *et al.*, 2016].

when it is its turn. Thus, if we have a strategy at the high level that achieves a goal/performs a task, it becomes irrelevant and it is impossible to realize this strategy at the low level; we can't use reasoning at the high level to guide the reasoning at the low level.

Low-Level Accountability. If it is the agent's turn at the high level and some ordinary action is executable online there, then as we have a sound abstraction, it follows that it is known that some refinement of it is executable at the low level. If instead it is the environment's turn and it is satisfiable that some exogenous action is executable at the high level, then sound abstraction is not sufficient to ensure that it is satisfiable that a refinement of a high-level exogenous action is executable at the low level. We need to avoid such cases, where an online transition exists at the high level, while the low level blocks, as no refinement of a high level action is executable.[8] Hence, we assume that after any sequence of refinements of high-level actions, i.e., in any situation s such that $Do(\text{ANYSEQHL}, S_0, s)$, if no refinement of any ordinary action is executable, then some refinement of some exogenous action is executable:

Assumption 4 (LL Exogenous Action Accountability)

$$\mathcal{D}_l \cup \mathcal{C} \models \forall s.Do(\text{ANYSEQHL}, S_0, s) \wedge \neg \bigvee_{A_i \in \mathcal{A}^o_{hl}} \exists \vec{x}\, \exists s'.Do(m(A_i(\vec{x})), s, s') \supset \bigvee_{A_i \in \mathcal{A}^e_{hl}} \exists \vec{x}\, \exists s'.Do(m(A_i(\vec{x})), s, s')$$

where $\mathcal{A}^o_{hl}$ (resp. $\mathcal{A}^e_{hl}$) represents the high-level ordinary (resp. exogenous) set of action types.

We also need another assumption to ensure that the high-level theory remains a sound abstraction of the low-level theory with respect to the mapping when a refinement of a high-level action is executed and the agent may obtain new knowledge. Therefore, we assume that along all online executions, whenever a refinement of a high-level action has been executed, the low-level agent *knows* that it has executed it:

Assumption 5 (Awareness of Executed HL Actions) *For all ground high-level action sequences $\vec{\alpha}$ and all ground low-level action sequences $\vec{a}$, if $\langle m(\vec{\alpha}), \epsilon\rangle \rightarrow^*_{\vec{a}} \langle \delta_l, \vec{a}\rangle$ and $\langle \delta_l, \vec{a}\rangle\checkmark$ for some δ_l, then $\mathcal{D}_l \cup \mathcal{C} \cup$ Sensed$(\vec{a}) \models Do(m(\vec{\alpha}), S_0, do(\vec{a}, S_0))$.*

Together, these assumptions amount to *low-level accountability*: the low level accounts for high-level ordinary actions because we have a sound abstraction, it accounts for high-level exogenous actions by Assumption 4, and it remains aware of the high-level actions executed by Assumption 5.

Local Refinement Enforceability. Assumptions 4 and 5 ensure that the low-level accounts for what happens at the high level, but does not guarantee its (local) enforceability, i.e., it does not guarantee the existence of strategies for the agent to enforce that refinements are successfully executed.

With respect to ordinary actions, (local) refinement enforceability requires that if it is known that there exists a refinement of an ordinary high-level action β that is executable at the low level then the agent has a strategy to successfully execute a refinement of β, no matter what the environment does. We assume this holds in every online configuration:

Assumption 6 (Ability to Execute Ordinary HL Actions) *For all ground high-level action sequences $\vec{\alpha}$ and all ground low-level action sequences $\vec{a}$ such that $\langle m(\vec{\alpha}), \epsilon\rangle \rightarrow^*_{\vec{a}} \langle \delta_l, \vec{a}\rangle$ and $\langle \delta_l, \vec{a}\rangle\checkmark$ for some δ_l, and for any ground high-level ordinary action $\beta \in \mathcal{A}^o$, if $\mathcal{D}_l \cup \mathcal{C} \cup$ Sensed$(\vec{a}) \models \exists s.Do(m(\beta), do(\vec{a}, S_0), s)$, then there exists a low-level strategy γ_l such that $AbleBy(m(\beta), \vec{a}, \gamma_l)$.*

If $\mathcal{D}_h \cup$ Sensed$(\vec{\alpha})$ is a sound abstraction of $\mathcal{D}_l \cup$ Sensed$(\vec{a})$ wrt mapping m and it is known at the high level that an ordinary action β is executable in $do(\vec{\alpha}, S_0)$, it follows that a refinement of β is known to be executable at the low level in $do(\vec{a}, S_0)$, and thus by Assumption 6, there is a strategy to successfully execute a refinement of β at the low level.

With respect to exogenous actions instead, we want to ensure that if a refinement of a high-level exogenous action is possibly executable, then a refinement of some exogenous action will eventually be successfully executed online no matter how the agent and environment choose to act.

To formalize this, similar to [Lespérance, 2002], we define a predicate *NecTerminates*$(\delta, \vec{a})$, meaning that all online executions of program δ in situation $do(\vec{a}, S_0)$ successfully terminate, as the least relation $\mathcal{R}(\delta, \vec{a})$ such that:

(A) for all pairs $(\delta, \vec{a})$, if $\langle \delta, \vec{a}\rangle\checkmark$ and there does not exist a, δ' such that $\langle \delta, \vec{a}\rangle \rightarrow_a \langle \delta', \vec{a}a\rangle$, then $\mathcal{R}(\delta, \vec{a})$;

(B) for all pairs $(\delta, \vec{a})$, if there exists a, δ' such that $\langle \delta, \vec{a}\rangle \rightarrow_a \langle \delta', \vec{a}a\rangle$ and for all a, δ' such that $\langle \delta, \vec{a}\rangle \rightarrow_a \langle \delta', \vec{a}a\rangle$, $\mathcal{R}(\delta', \vec{a}a)$, then $\mathcal{R}(\delta, \vec{a})$.

We also define a low-level program that represents all refinements of all high-level exogenous actions:

$$\text{ANYONEEXOHL} \doteq \mathbf{setp}(\{\pi \vec{x}.m(A_i(\vec{x})) \mid A_i \in \mathcal{A}^e_h\}),$$

i.e., do any refinement of an exogenous high-level primitive action,

where $\mathbf{setp}(P)$ a "delayed commitment" nondeterministic branch construct that executes a set of programs P nondeterministically without committing to which element of P is being executed until it has to.[9]

Given this, we can state our assumption:

Assumption 7 (Exogenous HL Actions Never Diverge) *For all ground high-level action sequences $\vec{\alpha}$ and all ground low-level action sequences $\vec{a}$ such that $\langle m(\vec{\alpha}), \epsilon\rangle \rightarrow^*_{\vec{a}} \langle \delta_l, \vec{a}\rangle$ and $\langle \delta_l, \vec{a}\rangle\checkmark$ for some δ_l, if there exists a ground high-level exogenous action $\beta \in \mathcal{A}^e$ such that $\mathcal{D}_l \cup \mathcal{C} \cup$ Sensed$(\vec{a}) \cup \{\exists s.Do(m(\beta), do(\vec{a}, S_0), s\}$ is satisfiable, then we have that NecTerminates$(\text{ANYONEEXOHL}, \vec{a})$.*

[8] Instead, we could assume the high-level theory is both a sound and *complete* abstraction of the low-level theory with respect to a mapping. However, this seems more restrictive; moreover, it would require showing that completeness is preserved as we execute the high-level strategy. We will investigate this in future work.

[9] [Banihashemi *et al.*, 2018b] axiomatize $\mathbf{setp}(P)$ as follows:
$Trans(\mathbf{setp}(P), s, \delta', s') \equiv$
$\exists \delta.\exists \delta''.\delta \in P \wedge Trans(\delta, s, \delta'', s') \wedge$
$\delta' = \mathbf{setp}(\{\delta'' \mid \exists \delta.\delta \in P \wedge Trans(\delta, s, \delta'', s')\})$
$Final(\mathbf{setp}(P), s) \equiv \exists \delta.\delta \in P \wedge Final(\delta, s)$.

These assumptions together with $\mathcal{D}_h$ being a sound abstraction of $\mathcal{D}_l$ wrt mapping m allow us to show our main result, i.e., if the high-level agent has a strategy γ_h to successfully execute a task represented by δ_h, then there exists a low-level strategy γ_l such that the low-level agent can ensure successful execution of a refinement of γ_h by using strategy γ_l, a refinement of γ_h; moreover, if it is known at the high level that after successful execution of δ_h, a situation-suppressed formula ϕ holds, then every such γ_l, ensures that *the refinement of ϕ holds* after γ_l terminates:

Theorem 3 ***(Strategy Refinement)*** *Suppose that $\mathcal{D}_h$ is a sound abstraction of $\mathcal{D}_l$ relative to mapping m and that Assumptions 1 to 7 hold. Then for all ground high-level action sequences $\vec{\alpha}$ and all ground low-level action sequences $\vec{a}$ such that $\langle \vec{\alpha}, \epsilon\rangle \rightarrow_{\vec{\alpha}}^{*} \langle \delta_h', \vec{\alpha}\rangle$ for some δ_h' and $\langle \delta_h', \vec{\alpha}\rangle\checkmark$ and $\langle m(\vec{\alpha}), \epsilon\rangle \rightarrow_{\vec{a}}^{*} \langle \delta_l, \vec{a}\rangle$ and $\langle \delta_l, \vec{a}\rangle\checkmark$ for some δ_l and for any online situation-determined high-level program δ_h and high-level strategy γ_h such that $AbleBy(\delta_h, \vec{\alpha}, \gamma_h)$, we have that:*[10]

1. *there exists a low-level strategy γ_l such that $AbleBy(m_p(\gamma_h), \vec{a}, \gamma_l)$;*
2. *moreover, for any situation-suppressed formula ϕ, if $\mathcal{D}_h \cup \mathcal{C} \models \forall s.Do(\delta_h, do(\vec{\alpha}, S_0), s) \supset \phi[s]$ then $AbleBy(m_p(\gamma_h); m(\phi)?, \vec{a}, \gamma_l)$.*

Part 1 is proven by induction on the length of the high-level strategy γ_h (i.e., number of actions in the longest branch), using Assumption 6 when γ_h starts with an ordinary action and Assumption 7 when it starts with (a set of) exogenous actions. Part 2 follows from Part 1, properties of $AbleBy$, and results on m_p from [Banihashemi *et al.*, 2018b]. Furthermore, it is clear from our proof that the low-level strategy γ_l can be obtained piecewise/incrementally: first obtain a refinement of the first step of γ_h, execute it online, and then repeat this for the remainder of γ_h until done.

Example 5 As discussed in Example 4, our agent has a high-level strategy γ_{h_1} by which she can execute the task δ_{h1} and achieve the goal ϕ_{h1}. Based on Theorem 3, there exists a low-level strategy γ_{l_1} by which the agent is able to accomplish a refinement of the goal $m(\phi_{h1})$; in fact we have

$$\begin{aligned}\gamma_{l_1} = {} & qryAM(C1); qryPr(C1); \mathbf{set}(\\ & (repAM(C1,1); repPr(C1,1); selFlt(C1,Bz,F1);\\ & \quad pay(C1,Bz,F1); nil),\\ & (repAM(C1,1); repPr(C1,0); selFlt(C1,Bz,F1);\\ & \quad pay(C1,Bz,F1); nil),\\ & (repAM(C1,0); repPr(C1,1); selFlt(C1,Bz,F1);\\ & \quad pay(C1,Bz,F1); nil),\\ & (repAM(C1,0); repPr(C1,0); selFlt(C1,Eco,F1);\\ & \quad pay(C1,Eco,F1); nil))\end{aligned}$$

where action $qryAM(c)$ (resp. $qryPr(c)$) queries if air miles (resp. promotion) exist for customer c and $repAM(c,x)$ (resp. $repPr(c,x)$) is an exogenous action that informs the agent whether $AirMiles(c)$ (resp. $Promotion(c)$) holds. □

[10] [Banihashemi *et al.*, 2018b] extend the mapping m to a mapping m_p that maps any SD high-level program δ^h to a SD low-level program that implements it: $m_p(\delta^h) \doteq \mathbf{setp}(\{\delta^h[A(\vec{t})/\mathbf{atomic}(m(A(\vec{t})))$ for all $A \in \mathcal{A}$, and $F(\vec{t})/m(F(\vec{t}))$ for all $F \in \mathcal{F}]\})$, where $\mathbf{atomic}(\delta)$ performs δ as an atomic unit, without allowing interleaving.

8 Conclusion

In this paper, we showed that abstraction can be used to make reasoning about ability much easier, even for agents that operate online, i.e., acquire new information as they execute. Specifically, we showed that under some reasonable assumptions, if the agent has a strategy by which she is able to achieve a goal at the high level, then we can refine it piecewise into a low-level strategy to achieve the refinement of the goal. Also, we identified general sufficient conditions for soundness of an abstraction to persist when the agent acquires new information.

The framework developed in this paper is very general and handles arbitrary first-order representations of the states of the dynamic systems. In such a general setting not much can be said about the computational aspects. However note that it is indeed possible to get an effective setting from the computational point of view if we restrict, for example, the high level to be propositional. In this way we get a finite state abstract system on which doing conditional planning becomes effective, (under the assumption that we are able to compute refinement of atomic actions). Similar results can be obtained for first-order bounded action theories [De Giacomo *et al.*, 2016]. We leave these questions for future work.

Developing methodologies and (partially) automated techniques for obtaining sound abstractions are important topics for future work. [Banihashemi *et al.*, 2017] shows that verifying that a number of properties are entailed by the low-level theory is sufficient to guarantee that one has a sound abstraction. Some of our assumptions are also straightforward entailment checks. For the others, we will investigate techniques for verifying them. Of course, one also needs to impose restrictions on the theory to get decidability as discussed above.

Acknowledgments

We acknowledge the support of Sapienza Ateneo Project Immersive Cognitive Environments and the National Science and Engineering Research Council of Canada.

References

[Alford *et al.*, 2009] Ronald Alford, Ugur Kuter, Dana S. Nau, Elnatan Reisner, and Robert P. Goldman. Maintaining focus: Overcoming attention deficit disorder in contingent planning. In *FLAIR*, 2009.

[Baier and McIlraith, 2006] Jorge A. Baier and Sheila A. McIlraith. On planning with programs that sense. In *KR*, pages 492–502. AAAI Press, 2006.

[Banihashemi *et al.*, 2016] Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. Online situation-determined agents and their supervision. In *KR*, pages 517–520. AAAI Press, 2016.

[Banihashemi *et al.*, 2017] Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. Abstraction in situation calculus action theories. In *AAAI*, pages 1048–1055. AAAI Press, 2017.

[Banihashemi *et al.*, 2018a] Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. Abstraction of agents executing online and their abilities in the situation calculus - Extended version. Technical Report EECS-2018-02, York University, 2018.

[Banihashemi *et al.*, 2018b] Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. Hierarchical agent supervision. To appear in AAMAS, 2018.

[Cimatti *et al.*, 2003] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1-2):35–84, 2003.

[De Giacomo and Levesque, 1999] Giuseppe De Giacomo and Hector J. Levesque. An incremental interpreter for high-level programs with sensing. In *Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter*, pages 86–102. Springer, Berlin, 1999.

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

[De Giacomo *et al.*, 2010] Giuseppe De Giacomo, Yves Lespérance, and Adrian R. Pearce. Situation calculus based programs for representing and reasoning about game structures. In *KR*. AAAI Press, 2010.

[De Giacomo *et al.*, 2012] Giuseppe De Giacomo, Yves Lespérance, and Christian J. Muise. On supervising agents in situation-determined ConGolog. In *AAMAS*, pages 1031–1038. IFAAMAS, 2012.

[De Giacomo *et al.*, 2016] Giuseppe De Giacomo, Yves Lespérance, and Fabio Patrizi. Bounded situation calculus action theories. *Artificial Intelligence*, 237:172–203, 2016.

[Dignum *et al.*, 2015] Frank Dignum, Virginia Dignum, Rui Prada, and Catholijn M. Jonker. A conceptual architecture for social deliberation in multi-agent organizations. *Multiagent and Grid Systems*, 11(3):147–166, 2015.

[Gabaldon, 2002] Alfredo Gabaldon. Programming hierarchical task networks in the situation calculus. In *AIPS02 Workshop on On-line Planning and Scheduling*, 2002.

[Geffner and Bonet, 2013] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers, 2013.

[Giunchiglia and Walsh, 1992] Fausto Giunchiglia and Toby Walsh. A theory of abstraction. *Artificial Intelligence*, 5(2):323–389, 1992.

[Kuter *et al.*, 2007] Ugur Kuter, Dana Nau, Elnatan Reisner, and Robert Goldman. Conditionalization: Adapting forward-chaining planners to partially observable environments. In *ICAPS 2007 - Workshop on planning and execution for real-world systems*, 2007.

[Lespérance *et al.*, 2000] 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.

[Lespérance *et al.*, 2008] Yves Lespérance, Giuseppe De Giacomo, and Atalay Nafi Ozgovde. A model of contingent planning for agent programming languages. In *AAMAS*, pages 477–484. IFAAMAS, 2008.

[Lespérance, 2002] Yves Lespérance. On the epistemic feasibility of plans in multiagent systems specification. In *Intelligent Agents VIII, 8th International Workshop, ATAL 2001*, volume 2333, pages 69–85. Springer, 2002.

[Levesque *et al.*, 1997] Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programing*, 31(1-3):59–83, 1997.

[McCarthy and Hayes, 1969] John McCarthy and Patrick J. Hayes. Some Philosophical Problems From the Standpoint of Artificial Intelligence. *Machine Intelligence*, 4:463–502, 1969.

[Milner, 1971] Robin Milner. An algebraic definition of simulation between programs. In *IJCAI*, pages 481–489. William Kaufmann, 1971.

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

[Mo *et al.*, 2016] Peiming Mo, Naiqi Li, and Yongmei Liu. Automatic verification of Golog programs via predicate abstraction. In *ECAI*, pages 760–768, 2016.

[Moore, 1985] Robert C. Moore. A formal theory of knowledge and action. In *Formal Theories of the Commonsense World*, pages 319–358. Ablex, NJ, 1985.

[Nayak and Levy, 1995] P. Pandurang Nayak and Alon Y. Levy. A semantic theory of abstractions. In *IJCAI*, pages 196–203. Morgan Kaufmann, 1995.

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

[Sardiña *et al.*, 2004a] Sebastian Sardiña, Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. On ability to autonomously execute agent programs with sensing. In *AAMAS*, pages 1522–1523. IEEE, 2004.

[Sardiña *et al.*, 2004b] Sebastian Sardiña, Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. On the semantics of deliberation in IndiGolog - from theory to implementation. *Annals of Mathematics and Artificial Intelligence*, 41(2-4):259–299, 2004.

[Sardiña *et al.*, 2006] Sebastian Sardiña, Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. On the limits of planning over belief states under strict uncertainty. In *KR*, pages 463–471. AAAI Press, 2006.