Online Situation-Determined Agents and their Supervision

Bita Banihashemi
York University
Toronto, ON, Canada
bita@cse.yorku.ca

Giuseppe De Giacomo
Sapienza Università di Roma
Roma, Italy
degiacomo@dis.uniroma1.it

Yves Lespérance
York University
Toronto, Canada
lesperan@cse.yorku.ca

Abstract

Agent supervision is a form of control/customization where a supervisor restricts the behavior of an agent to enforce certain requirements, while leaving the agent as much autonomy as possible. This is done in a setting based on the situation calculus and a variant of the ConGolog programming language that is situation-determined, i.e., the remaining program is a function of the action performed. When an agent may acquire new knowledge about her environment during execution, for example, by sensing, being situation-determined is not sufficient to ensure a unique remaining program. In such cases we need to consider an agent's *online executions*, where as she executes the program, at each time point she must make decisions on what to do next based on what her current knowledge is. In this work, we define a notion of *online situation-determined* agent program that ensures unique remaining configuration in online executions. Moreover, we formalize the definition of the *online maximally permissive supervisor*, and discuss its properties.

1 Introduction

In many settings, an agent's behavior needs to be restricted to conform to a set of specifications. For instance, the activities of agents in an organization have to adhere to some business rules and privacy/security protocols. One form of this is *customization*, where a generic process for performing a task or achieving a goal is refined to satisfy a client's constraints or preferences. Process customization includes personalization (e.g. of travel planning agents) and configuration (e.g. of e-commerce sites for each service provider).

A key challenge in such settings is ensuring conformance to specifications while preserving the agent's autonomy. Motivated by this and inspired by supervisory control of discrete event systems (Wonham and Ramadge 1987) De Giacomo, Lespérance and Muise (De Giacomo, Lespérance, and Muise 2012) (DLM) proposed *agent supervision* as a form of control/customization of an agent's behavior. The DLM framework is based on the situation calculus (McCarthy and Hayes 1969; Reiter 2001) and a variant of the ConGolog (De Giacomo, Lespérance, and Levesque 2000) programming language that is *situation-determined*, i.e., while allowing nondeterminism, requires the remaining program to be a function of the action performed. DLM represent the agent's possible behaviors as a nondeterministic ConGolog process. Another ConGolog process represents the supervision specification, i.e., which behaviors are acceptable/desirable.

If it is possible to control all of the agent's actions, then it is easy to obtain the behaviors of the supervised agent through a kind of synchronous concurrent execution of the agent process and the supervision specification process. However, some of the agent's actions may be *uncontrollable*. DLM formalize a notion of *maximally permissive supervisor* that minimally constrains the behavior of the agent in the presence of uncontrollable actions so as to enforce the desired behavioral specifications. The original DLM account of agent supervision assumes that the agent does not acquire new knowledge about her environment while executing. This means that all reasoning is done using the same knowledge base. The resulting executions are said to be *offline executions*.

In this paper, we investigate how we can apply DLM's framework in cases where an agent may acquire new knowledge about her environment during execution, for example, by sensing. Thus the knowledge base used by the agent in her reasoning needs to be updated during the execution. In such cases, being situation-determined is not sufficient to ensure a unique remaining program, as at each time point, the knowledge base used by the agent to deliberate about the next action is different. For example, consider a travel planner agent that needs to book a seat on a certain flight. Only after querying the airline web service offering that flight will the agent know if there are seats available on the flight.

Technically this requires switching from offline executions to *online executions* (De Giacomo and Levesque 1999; Sardiña et al. 2004). [??Knowledge change in online executions may be modeled by adding an epistemic accessibility relation?] (Scherl and Levesque 2003). [Here, we use a different approach and define it meta-theoretically?]. We define a notion of *online situation-determined* agent program that ensures unique remaining configuration in online executions. Moreover, we formalize the definition of the *online maximally permissive supervisor*, and discuss its properties.

2 Preliminaries

The *situation calculus* (SC) is a well known predicate logic language for representing and reasoning about dynamically changing worlds. Within the language, one can formulate action theories that describe how the world changes as the result of actions (Reiter 2001). 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. As a result a basic action theory (BAT) $\mathcal{D}$ is the union of the following disjoint sets: the foundational, domain independent, (second-order, or SO) axioms of the situation calculus (Σ), (first-order, or FO) precondition axioms stating when actions can be legally performed ($\mathcal{D}_{poss}$), (FO) successor state axioms describing how fluents change between situations ($\mathcal{D}_{ssa}$), (FO) unique name axioms for actions and domain closure on action types ($\mathcal{D}_{ca}$); (SO) unique name axioms and domain closure for object constants ($\mathcal{D}_{coa}$); and (FO) axioms describing the initial configuration of the world ($\mathcal{D}_{S_0}$). A special predicate $Poss(a, s)$ is used to state that action a is executable in situation s; precondition axioms in $\mathcal{D}_{poss}$ characterize this predicate. The abbreviation $Executable(s)$ means that every action performed in reaching situation s was possible in the situation in which it occurred. In turn, successor state axioms encode the causal laws of the world being modeled; they replace the so-called effect axioms and provide a solution to the frame problem. We write $do([a_1, a_2, \ldots, a_{n-1}, a_n], s)$ as an abbreviation for the situation term $do(a_n, do(a_{n-1}, \ldots, do(a_2, do(a_1, s)) \ldots))$.

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. Here, we concentrate on (a fragment of) ConGolog that includes the following constructs:

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

In the above, α is an action term, possibly with parameters, and φ is situation-suppressed formula, i.e., a SC 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. 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 variable x (observe that such a choice is, in general, unbounded). δ^* performs δ zero or more times. Program $\delta_1\|\delta_2$ represents the interleaved concurrent execution of programs δ_1 and δ_2. The intersection/synchronous concurrent execution of programs δ_1 and δ_2 (introduced by DLM) is denoted by $\delta_1\&\ \delta_2$.

Formally, the semantics of ConGolog is specified in terms of single-step transitions, using two predicates (De Giacomo, Lespérance, and Levesque 2000): *(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, Lespérance, and Pearce 2010); differently from (De Giacomo, Lespérance, and Levesque 2000), the test construct $\varphi?$ does not yield any transition, but is final when satisfied. Thus, it is a *synchronous* version of the original test construct (it does not allow interleaving). As a result, in our version of ConGolog, every transition involves the execution of an action. Predicate $Do(\delta, s, s')$ means that program δ, when executed starting in situation s, has as a legal terminating situation s', 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 $Trans$.

A ConGolog program δ is *situation-determined* (SD) in a situation s (De Giacomo, Lespérance, and Muise 2012) if for every sequence of transitions, the remaining program is determined by the resulting situation, i.e.,

$$\begin{aligned}&SituationDetermined(\delta, s) \doteq \forall s', \delta', \delta''.\\ &\quad Trans^*(\delta, s, \delta', s') \wedge Trans^*(\delta, s, \delta'', s') \supset \delta' = \delta'',\end{aligned}$$

For example, program $(a; b) \mid (a; c)$ is not SD, while $a; (b \mid c)$ is (assuming the actions involved are always executable). Thus, a (partial) execution of a SD program is uniquely determined by the sequence of actions it has produced. Hence a program in a starting situation generates a set/language of action sequences, its executions, and operations like intersection and union become natural.

In the rest, we use $\mathcal{C}$ to denote the axioms defining the ConGolog programming language.

3 Online Situation-Determined Agents

In our account of agent supervision, we want to accommodate agents that can acquire new knowledge about their environment during execution, for example by sensing, and where their knowledge base is updated with this new knowledge. Thus we consider an agent's *online executions*, where, as she executes the program, at each time point, she makes decisions on what to do next based on what her current knowledge is.

Sensing. A crucial aspect of online executions is that the agent can take advantage of sensing. Similarly to (Lespérance, De Giacomo, and Ozgovde 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.

Specifically, to sense whether fluent P holds within a program, we use a macro:

$$SenseP \doteq QryIfP; (repValP(1) \mid repValP(0)),$$

where $QryIfP$ is an ordinary action that is always executable and is used to query (i.e., sense) if P holds and $repValP(x)$ is an exogenous action with no effect that informs the agent if P holds through its precondition axiom, which is of the form:

$$Poss(repValP(x), s) \equiv P(s) \wedge x = 1 \vee \neg P(s) \wedge x = 0.$$

Thus, we can understand that $SenseP$ reports value 1 through the execution of $repValP(1)$ if P holds, and 0 through the execution of $repValP(0)$ otherwise.

For example, consider the following agent program:

$$\delta^i = \mathit{SenseP}; [P?; A] \mid [\neg P?; B]$$

and assume the agent does not know if P holds initially. Initially we have $\mathcal{D} \cup \mathcal{C} \models \mathit{Trans}(\delta^i, S_0, \delta', S_1)$ where $S_1 = do(\mathit{QryIfP}, S_0)$ and $\delta' = nil; (\mathit{repValP}(1) \mid \mathit{repValP}(0))); [P?; A] \mid [\neg P?; B]$. At S_1, the agent knows either of the exogenous actions $\mathit{repValP}(0)$ or $\mathit{repValP}(1)$ could occur, but does not know which. After the occurrence of one of these actions, the agent learns whether P holds. For example, if $\mathit{repValP}(1)$ occurs, the agent's knowledge base is now updated to $\mathcal{D} \cup \mathcal{C} \cup \{\mathit{Poss}(\mathit{repValP}(1), S_1)\}$. With this updated knowledge, she knows which action to do next: $\mathcal{D} \cup \mathcal{C} \cup \mathit{Poss}(\mathit{repValP}(1), S_1) \models \mathit{Trans}(nil; [P?; A] \mid [\neg P?; B], do(\mathit{repValP}(1), S_1), nil, do([\mathit{repValP}(1), A], S_1))$.

Notice that 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). In particular the current KB after having performed the sequence of actions $\vec{a}$ is:

$$\mathcal{D} \cup \mathcal{C} \cup \{\mathit{Executable}(do(\vec{a}, S_0)\}.$$

Note that this approach also handles the agent's acquiring knowledge from an arbitrary exogenous action.

Agent online configurations and transitions. We denote an *agent* by σ, which stands for a pair $\langle \mathcal{D}, \delta^i \rangle$, where δ^i is the initial program of the agent expressed in ConGolog and $\mathcal{D}$ is a BAT that represents the agent's initial knowledge (which may be incomplete). We assume that we have a finite set of primitive action types $\mathcal{A}$, which is the disjoint union of a set of ordinary primitive action types $\mathcal{A}^o$ and exogenous primitive action types $\mathcal{A}^e$.

An *agent configuration* is modeled as a pair $\langle \delta, \vec{a} \rangle$, where δ is the remaining program and $\vec{a}$ is the current history, i.e, the sequence of actions performed so far starting from S_0. The initial configuration c^i is $\langle \delta^i, \epsilon \rangle$, where ϵ is the empty sequence of actions.

The *online transition relation* between agent configurations is (a meta-theoretic) binary relation between configurations defined 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
$\mathcal{D} \cup \mathcal{C} \cup \{\mathit{Executable}(do(\vec{a}, S_0))\} \models$
$\mathit{Trans}(\delta, do(\vec{a}, S_0), \delta', do(A(\vec{n}), do(\vec{a}, S_0)))$
or $A \in \mathcal{A}^e$, $\vec{n} \in \mathcal{N}^k$ and
$\mathcal{D} \cup \mathcal{C} \cup \{\mathit{Executable}(do(\vec{a}, S_0)),$
$\mathit{Trans}(\delta, do(\vec{a}, S_0), \delta', do(A(\vec{n}), do(\vec{a}, S_0)))\}$ is satisfiable.

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, i.e., it must be consistent with what she knows that the action is executable and δ' is the remaining program. As part of the transition, the theory is (implicitly) updated in that the new exogenous action $A(\vec{n})$ is added to the action sequence, and $\mathit{Executable}(do([\vec{a}, A(\vec{n})], S_0))$ will be added to the theory when it is queried in later transitions, thus incorporating the fact that $\mathit{Poss}(A(\vec{n}), do(\vec{a}, S_0))$ is now known to hold.

The (meta-theoretic) relation $c \rightarrow^*_{\vec{a}} c'$ is the reflexive-transitive closure of $c \rightarrow_{A(\vec{n})} c'$ and denotes that online configuration c' can be reached from the online configuration c by performing a sequence of online transitions involving the sequence of actions $\vec{a}$.

We also define a (meta-theoretic) predicate $c^{\checkmark}$ meaning that the online configuration c is known to be final:

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

Online situation determined agents. In this paper, we are interested in programs that are SD, i.e., given a program, a situation and an action, we want the remaining program to be determined. However this is not sufficient when considering online executions. We want to ensure that the agent always knows what the remaining program is after any sequence of actions. We say that an agent is *online situation-determined* (online SD) if for any sequence of actions that the agent can perform online, the resulting agent configuration is unique. Formally, an agent $\sigma = \langle \mathcal{D}, \delta^i \rangle$ with initial configuration $c^i = \langle \delta^i, \epsilon \rangle$ is *online SD* if and only if for all sequences of action $\vec{a}$, if $c^i \rightarrow^*_{\vec{a}} c'$ and $c^i \rightarrow^*_{\vec{a}} c''$ then $c' = c''$.

We say that an agent $\sigma = \langle \mathcal{D}, \delta^i \rangle$ *always knows the remaining program after an exogenous action* if and only if

for all action sequences $\vec{a}$, $A \in \mathcal{A}^e$, $\vec{n} \in \mathcal{N}^k$
if $\mathcal{D} \cup \mathcal{C} \cup \{\mathit{Executable}(do(\vec{a}, S_0)),$
$\mathit{Trans}(\delta, do(\vec{a}, S_0), \delta', do([\vec{a}, A(\vec{n})], S_0)$ is satisfiable,
then there exists a program δ' such that
$\mathcal{D} \cup \mathcal{C} \cup \{\mathit{Executable}(do([\vec{a}, A(\vec{n})], S_0))\} \models$
$\mathit{Trans}(\delta, do(\vec{a}, S_0), \delta', do([\vec{a}, A(\vec{n})], S_0))$.

Essentially, this states that whenever the agent considers it possible that an exogenous action may occur, then she knows what the remaining program is afterwards if it does occur.

Finally, we say that an agent $\sigma = \langle \mathcal{D}, \delta^i \rangle$ with initial configuration $c^i = \langle \delta^i, \epsilon \rangle$ is *online SD* if and only if for all sequences of action $\vec{a}$, if $c^i \rightarrow^*_{\vec{a}} c'$ and $c^i \rightarrow^*_{\vec{a}} c''$ then $c' = c''$. We can show that:

Theorem 1 *For any agent $\sigma = \langle \mathcal{D}, \delta^i \rangle$, if δ^i is known to be SD in $\mathcal{D}$, i.e., $\mathcal{D} \cup \mathcal{C} \models \mathit{SituationDetermined}(\delta^i, S_0)$ and if σ always knows the remaining program after an exogenous action, then σ is online SD.*

Online Runs. For an agent σ that is online SD, online executions can be succinctly represented by *runs* formed by the corresponding sequence of actions. The set $\mathcal{RR}(\sigma)$ of (partial) *runs* of an online SD agent σ with starting configuration c^i is the sequences of actions that can be produced by executing c^i from S_0: $\mathcal{RR}(\sigma) = \{\vec{a} \mid \exists c. c^i \rightarrow^*_{\vec{a}} c\}$. A run is *complete* if it reaches a final configuration. Formally we define the set $\mathcal{CR}(\sigma)$ of complete runs as: $\mathcal{CR}(\sigma) = \{\vec{a} \mid$

$\exists c. c^i \rightarrow^*_{\vec{a}} c \wedge c^{\checkmark}\}$. Finally we say that a run is *good* if it can be extended to a complete run. Formally we define the set $\mathcal{GR}(\sigma)$ of good runs as: $\mathcal{GR}(\sigma) = \{\vec{a} \mid \exists c, c', \vec{a'}. c^i \rightarrow^*_{\vec{a}} c \wedge c \rightarrow^*_{\vec{a'}} c' \wedge c'^{\checkmark}\}$.

4 Online Agent Supervision

Agent supervision aims at restricting an agent's behavior to ensure that it conforms to a supervision specification while leaving it as much autonomy as possible. DLM's account of agent supervision is based on offline executions and does not accommodate agents that acquire new knowledge during a run. DLM assume that the agent's possible behaviors are represented by a (nondeterministic) SD ConGolog program δ^i relative to a BAT $\mathcal{D}$. The supervision specification is represented by another SD ConGolog program δ^s. First note that if it is possible to control all the actions of the agent, then it is straightforward to specify the result of supervision as the intersection of the agent and the specification processes ($\delta^i \& \delta^s$). However in general, some of agent's actions may be *uncontrollable*. These are often the result of interaction of an agent with external resources, or may represent aspects of agent's behavior that must remain autonomous and cannot be controlled directly. This is modeled by the special fluent $A_u(a, s)$ that means action a is uncontrollable in situation s.

DLM say that a supervision specification δ^s is *controllable* wrt the agent program δ^i in situation s iff:

$$\forall \vec{a} a_u. \exists \vec{b}. Do(\delta^s, s, do([\vec{a}, \vec{b}], s)) \wedge A_u(a_u, do(\vec{a}, s)) \supset \\ (\exists \vec{d}. Do(\delta^i, s, do([\vec{a}, a_u, \vec{d}], s)) \supset \exists \vec{b'}. Do(\delta^s, s, do([\vec{a}, a_u, \vec{b'}], s))),$$

i.e., if we postfix an action sequence $\vec{a}$ that is good offline run for δ^s (i.e., such that $\exists \vec{b}. Do(\delta^s, s, do([\vec{a}, \vec{b}], s))$ holds) with an uncontrollable action a_u which is good for δ^i, then a_u must also be good for δ^s.

Then, DLM define the *offline maximally permissive supervisor* (offline MPS) $mps_{\mathit{offl}}(\delta^i, \delta^s, s)$ of the agent behavior δ^i which fulfills the supervision specification δ^s as:

$$mps_{\mathit{offl}}(\delta^i, \delta^s, s) = \mathbf{set}(\textstyle\bigcup_{E \in \mathcal{E}} E) \text{ where}$$
$$\mathcal{E} = \{E \mid \forall \vec{a} \in E \supset Do(\delta^i \ \& \ \delta^s, s, do(\vec{a}, s)) \\ \text{and } \mathbf{set}(E) \text{ is controllable wrt } \delta^i \text{ in } s\}$$

This says that the offline MPS is the union of all sets of action sequences that are complete offline runs of both δ^i and δ^s (i.e., such that $Do(\delta^i \ \& \ \delta^s, s, do(\vec{a}, s)))$ that are controllable for δ^i in situation s.

The above definition uses the $\mathbf{set}(E)$ construct introduced by DLM, which is a sort of infinitary nondeterministic branch; it takes an arbitrary set of sequences of actions E and turns it into a program. We define its semantics as follows:

$$Trans(\mathbf{set}(E), s, \delta', s') \equiv \exists a, \vec{a}. a\vec{a} \in E \wedge Poss(a, s) \wedge \\ s' = do(a, s) \wedge \delta' = \mathbf{set}(\{\vec{a} \mid a\vec{a} \in E \wedge Poss(a, s)\}) \\ Final(\mathbf{set}(E), s) \equiv \epsilon \in E$$

Therefore $\mathbf{set}(E)$ can be executed to produce any of the sequences of actions in E.[1]

[1]Obviously there are certain sets that can be expressed directly in ConGolog, e.g., when E is finite. However in the general case, the object domain may be infinite, and $\mathbf{set}(E)$ may not be representable as a finitary ConGolog program.

DLM show that their notion of offline MPS, $mps_{\mathit{offl}}(\delta^i, \delta^s, s)$, has many nice properties: it always exists and is unique, it is controllable wrt the agent behavior δ^i in s, and it is the largest set of offline complete runs of δ^i that is controllable wrt δ^i in s and satisfy the supervision specification δ^s in s, i.e., is maximally permissive. However, the notion of offline MPS is inadequate in the context of online execution, as the following example shows.

Example 1 Suppose that we have an agent that does not know whether P holds initially, i.e., $\mathcal{D} \not\models P(S_0)$ and $\mathcal{D} \not\models \neg P(S_0)$. Suppose that the agent's initial program is:

$$\delta^i_4 = [P?; ((A; (C \mid U)) \mid (B; D))] \ \mid \\ [\neg P?; ((A; D) \mid (B; (C \mid U)))]$$

where all actions are ordinary, always executable, and controllable except for U, which is always uncontrollable. Suppose that the supervision specification is:

$$\delta^s_4 = (\pi a. a \neq U?; a)^*$$

i.e., any action except U can be performed. It is easy to show that the offline MPS obtained using DLM's definition is different depending on whether P holds or not:

$$\mathcal{D} \cup \mathcal{C} \models (P(S_0) \supset mps_{\mathit{offl}}(\delta^i_4, \delta^s_4, S_0) = \mathbf{set}(\{[B; D]\})) \wedge \\ (\neg P(S_0) \supset mps_{\mathit{offl}}(\delta^i_4, \delta^s_4, S_0) = \mathbf{set}(\{[A; D]\}))$$

For models of the theory where P holds, the offline MPS is $\mathbf{set}(\{B; D\})$, as the set of complete offline runs of δ^s_4 in S_0 is $\{[B; D], [A; C]\}$ and $\mathbf{set}(\{[A; C]\})$ is not controllable wrt δ^i_4 in S_0. For models where P does not hold, the offline MPS is $\mathbf{set}(\{A; D\})$, since the set of complete offline runs of δ^s_4 in S_0 is $\{[A; D], [B; C]\}$ and $\mathbf{set}(\{[B; C]\})$ is not controllable wrt δ^i_4 in S_0. Since it is not known if P holds, it seems that a correct supervisor should neither allow A nor B. $\Box$

As the above example illustrates, we have an offline MPS for each model of the theory. Instead, we want a single online MPS that works for all models and includes sensing information when acquired.

Online Maximally Permissive Supervisor. In our account of supervision, we want to deal with agents that may acquire knowledge through sensing and exogenous actions as they operate and make decisions based on what they know, and we model these as online SD agents. Let's see how we can formalize supervision for such agents. Assume that we have an online SD agent $\sigma = \langle \mathcal{D}, \delta^i \rangle$ whose behavior we want to supervise. Let's also suppose that we have a *supervision specification* δ^s of what behaviors we want to allow in the supervised system, where δ^s is a SD ConGolog program relative to the BAT $\mathcal{D}$ of the agent. In fact, we assume that the system $\langle \mathcal{D}, \delta^s \rangle$ is also online SD. We say that a specification δ^s is *online controllable* wrt online SD agent $\sigma = \langle \mathcal{D}, \delta^i \rangle$ iff:

$\forall \vec{a}a_u . \vec{a} \in \mathcal{GR}(\langle \mathcal{D}, \delta^s \rangle)$ and
$\mathcal{D} \cup \{Executable(do(\vec{a}, S_0))\} \not\models \neg A_u(a_u, do(\vec{a}, S_0))$ implies
if $\vec{a}a_u \in \mathcal{GR}(\sigma)$ then $\vec{a}a_u \in \mathcal{GR}(\langle \mathcal{D}, \delta^s \rangle)$.

This says that if we postfix a good online run $\vec{a}$ for $\langle \mathcal{D}, \delta^s \rangle$ with an action a_u that is not known to be controllable which is good for σ (and so $\vec{a}$ must be good for σ as well), then a_u must also be good for $\langle \mathcal{D}, \delta^s \rangle$. (Note that $\vec{a}a_u \in \mathcal{GR}(\sigma)$ and $\vec{a}a_u \in \mathcal{GR}(\langle \mathcal{D}, \delta^s \rangle)$ together imply that $\vec{a}a_u \in \mathcal{GR}(\langle \mathcal{D}, \delta^i \& \ \delta^s \rangle)$.) This definition is quite similar to DLM's. But it differs in that it applies to online runs as opposed to offline runs. Moreover it treats actions that are not known to be controllable as uncontrollable, thus ensuring that δ^s is controllable in all possible models/worlds compatible with what the agent knows. Note that like DLM, we focus on good runs of the process, assuming that the agent will not perform actions that don't lead to a final configuration of δ^i. The supervisor only ensures that given this, the process always conforms to the specification.

Given this, we can then define the *online maximally permissive supervisor* $mps_{onl}(\delta^s, \sigma)$ of the online SD agent $\sigma = \langle \mathcal{D}, \delta^i \rangle$ which fulfills the supervision specification δ^s:

$$mps_{onl}(\delta^s, \sigma) = \mathbf{set}(\textstyle\bigcup_{E \in \mathcal{E}} E) \text{ where}$$
$$\mathcal{E} = \{E \mid E \subseteq \mathcal{CR}(\langle \mathcal{D}, \delta^i \ \& \ \delta^s \rangle) \text{ and } \mathbf{set}(E) \text{ is online controllable wrt } \sigma\}$$

i.e., the online MPS is the union of all sets of action sequences that are complete online runs of both δ^i and δ^s that are online controllable for the agent σ. Again, our definition is similar to DLM's, but applies to online runs, and relies on online (as opposed to offline) controllability. For the maximally permissive supervisor $mps_{onl}(\delta^s, \sigma)$ of the online SD agent $\sigma = \langle \mathcal{D}, \delta^i \rangle$ which fulfills the supervision specification δ^s, where $\langle \mathcal{D}, \delta^s \rangle$ is also online SD, the following properties hold: it always exists and is unique, it is online SD, it is online controllable wrt σ, it is non-blocking, and it is maximally permissive.

Example 2 If we return to the agent of Example 1, who does not know whether P holds initially, it is easy to show that our definition of online MPS yields the correct result, i.e. $mps_{onl}(\delta^s_4, \langle \mathcal{D}, \delta^i_4 \rangle) = \mathbf{set}(\{\epsilon\})$. □

Example 3 Supervision can also depend on the information that the agent acquires as it executes. Again, suppose that we have an agent that does not know whether P holds initially. Suppose also that the agent's initial program is $\delta^i_5 = Sense_P; \delta^i_4$. We can show that:

$$\begin{aligned}\mathcal{D} \cup \mathcal{C} \models (P(S_0) \supset mps_{\mathit{offl}}(\delta^i_5, \delta^s_4, S_0) = &\\ \mathbf{set}(\{[QryIfP, repValP(1), B, D]\})) \wedge &\\ (\neg P(S_0) \supset mps_{\mathit{offl}}(\delta^i_5, \delta^s_4, S_0) = &\\ \mathbf{set}(\{[QryIfP, repValP(0), A, D]\})) &\end{aligned}$$

Again, we have different offline MPSs depending on whether P holds. But since the exogenous report makes the truth value of P known after the first action, we get one online MPS for this agent as follows:

$$\begin{aligned}mps_{onl}(\delta^s_4, \langle \mathcal{D}, \delta^i_5 \rangle) = \mathbf{set}(\{&[QryIfP, repValP(1), B, D],\\ &[QryIfP, repValP(0), A, D]\})\end{aligned}$$

Because the agent queries if P holds, the supervisor has enough information to decide the maximal set of runs from then on in each case. So if the reported value of P is true, then the online supervisor should eliminate the complete run $[A, C]$ as it is not controllable, and if P does not hold, the run $[B, C]$ should be eliminated for the same reason. □

As well, an action's controllability or whether it satisfies the specification may depend on a condition whose truth only becomes known during the execution. Such cases cannot be handled by DLM's original offline account but our online supervision account does handle them correctly.

5 Discussion

In this paper we have formalized the notion of online SD programs and defined online maximally permissive supervisor for agents that execute online and can acquire new knowledge as they operate. Our definition requires the use of program construct *set* which is mostly of theoretical interest. In future work, we focus on meta-theoretically defining a program construct (i.e. supervision operator) for online supervised execution that given the agent and specification, executes them to obtain only runs allowed by the maximally permissive supervisor. We also plan to define a new lookahead search construct that ensures the agent can successfully complete the execution (i.e., ensures nonblockingness).

If the object domain is finite, then finite-state techniques developed for discrete events systems (Wonham and Ramadge 1987) can be adapted to synthesize a program that characterizes the online MPS. It should also be possible to effectively synthesize supervisors for agents that use bounded action theories (De Giacomo, Lespérance, and Patrizi 2013; De Giacomo et al. 2014); verification of temporal properties over such agents is known to be decidable.

References

De Giacomo, G., and Levesque, H. J. 1999. An incremental interpreter for high-level programs with sensing. In *Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter*. 86–102.

De Giacomo, G.; Lespérance, Y.; Patrizi, F.; and Vassos, S. 2014. LTL verification of online executions with sensing in bounded situation calculus. In *ECAI*, volume 263, 369–374.

De Giacomo, G.; Lespérance, Y.; and Levesque, H. J. 2000. ConGolog, a concurrent programming language based on the situation calculus. *Artif. Intell.* 121(1–2):109–169.

De Giacomo, G.; Lespérance, Y.; and Muise, C. J. 2012. On supervising agents in situation-determined ConGolog. In *AAMAS*, 1031–1038.

De Giacomo, G.; Lespérance, Y.; and Patrizi, F. 2013. Bounded epistemic situation calculus theories. In *IJCAI*.

De Giacomo, G.; Lespérance, Y.; and Pearce, A. R. 2010. Situation calculus based programs for representing and reasoning about game structures. In *KR*.

Lespérance, Y.; De Giacomo, G.; and Ozgovde, A. N. 2008. A model of contingent planning for agent programming languages. In *AAMAS*, 477–484.

McCarthy, J., and Hayes, P. J. 1969. Some Philosophical Problems From the StandPoint of Artificial Intelligence. *Machine Intelligence* 4:463–502.

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

Sardiña, S.; De Giacomo, G.; Lespérance, Y.; and Levesque, H. J. 2004. On the semantics of deliberation in Indigolog - from theory to implementation. *Ann. Math. Artif. Intell.* 41(2-4):259–299.

Scherl, R. B., and Levesque, H. J. 2003. Knowledge, action, and the frame problem. *Artif. Intell.* 144(1-2):1–39.

Wonham, W., and Ramadge, P. 1987. On the supremal controllable sub-language of a given language. *SIAM Journal on Control and Optimization* 25(3):637–659.