Online agent supervision in the situation calculus - Extended version

Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance

Technical Report EECS-2016-02

July 12 2016

Department of Electrical Engineering and Computer Science
4700 Keele Street, Toronto, Ontario M3J 1P3 Canada

Online Agent Supervision in the Situation Calculus - Extended Version

Bita Banihashemi
York University
Toronto, 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. In this work, we investigate supervision of an agent that may acquire new knowledge about her environment during execution, for example, by sensing. Thus we 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. This is done in a setting based on the situation calculus and a variant of the ConGolog programming language. To reason about such agents, we first define a notion of online situation-determined agent that ensures that for any sequence of actions that the agent can perform online, the resulting agent configuration is unique. The main results of this paper are (i) a formalization of the online maximally permissive supervisor, (ii) a sound and complete technique for execution of the agent as constrained by such a supervisor, and (iii) a new type of lookahead search construct that ensures nonblockingness over such online executions.

1 Introduction

In many settings, we want to restrict an agent's behavior 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. Similarly, a mobile robot has to conform to safety specifications and avoid causing injuries to others. 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 (Fritz and McIlraith 2006) and configuration (Liaskos et al. 2012) and finds application in number of areas.

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; Wonham 2014; Cassandras and Lafortune 2008), 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. 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.[1]

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 study how we can apply the DLM framework in the case where the agent may acquire new knowledge while executing, for example through sensing. This means that the knowledge base that the agent uses in her reasoning needs to be updated during the execution. For instance, 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), which, differently from offline executions, can only be defined meta-theoretically (unless one adds a knowledge operator/fluent (Scherl and Levesque 2003)) since at every time point the knowledge base used by the agent to deliberate about the next action is different.

Based on online executions, we formalize the notion of *online maximally permissive supervisor* and show its existence and uniqueness, as in the simpler case of DLM. Moreover, we meta-theoretically define a program construct (i.e., supervision operator) for online supervised execution that given the agent and specification, executes them to obtain

[1] In some cases, a declarative specification language would be preferable, e.g., linear temporal logic (LTL). (Fritz and McIlraith 2006) show how an extended version of LTL interpreted over a finite horizon can be compiled into ConGolog.

only runs allowed by the maximally permissive supervisor, showing its soundness and completeness. We also define a new lookahead search construct that ensures the agent can successfully complete the execution (i.e., ensures nonblockingness).

2 Preliminaries

2.1 The Situation Calculus

The *situation calculus* (SC) is a logical language specifically designed for representing and reasoning about dynamically changing worlds (McCarthy and Hayes 1969; Reiter 2001). All changes to the world are the result of *actions*, which are terms in the logic. A possible world history is represented by a term called a *situation*. The constant S_0 is used to denote the initial situation where no actions have yet been performed. Sequences of actions are built using the function symbol do, such that $do(a, s)$ denotes the successor situation resulting from performing action a in situation s. Predicates and functions whose value varies from situation to situation are called *fluents*, and are denoted by symbols taking a situation term as their last argument (e.g., $Holding(x, s)$). 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 take the place of 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 $do(a_n, do(a_{n-1}, \ldots, do(a_2, do(a_1, s)) \ldots))$; for an action sequence $\vec{a}$, we often write $do(\vec{a}, s)$ for $do([\vec{a}], s)$.

2.2 High-Level Programs and 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. Here, we concentrate on (a fragment of) ConGolog (De Giacomo, Lespérance, and Levesque 2000) that includes the following constructs:

α	atomic action
$\varphi?$	test for a condition
$\delta_1; \delta_2$	sequence
if φ **then** δ_1 **else** δ_2	conditional
while φ **do** δ	while loop
$\delta_1 \vert \delta_2$	nondeterministic branch
$\pi x.\delta$	nondeterministic choice of argument
δ^*	nondeterministic iteration
$\delta_1 \Vert \delta_2$	interleaved concurrency
$\delta_1 \& \delta_2$	synchronous concurrency/intersection

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. The test action $\varphi?$ determines if condition φ holds. 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 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.

2.3 Situation-Determined ConGolog (SDConGolog)

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

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

where *Trans** denotes the reflexive transitive closure of *Trans*. 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.

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

3 A Travel Planning Example

As a motivating example, we consider the task of planning a trip, which may involve booking a hotel and/or flight. Imagine we have a *very generic* travel agent whose behavior is defined by the ConGolog process $\delta_{travelPlanner}$ below, which we will later customize based on a client's requirements:

$\delta_{travelPlanner}(cID, oC, dC, bD, eD) =$
$(\delta_{qryHtl}(cID, dC, bD, eD) \mid \delta_{qryAir}(cID, oC, dC, bD, eD) \mid$
$\delta_{pickHtl}(cID, dC, bD, eD) \mid \delta_{pickAir}(cID, oC, dC, bD, eD))^*;$
if $\exists htlID\ SelHtl(cID, htlID) \vee \exists airID\ SelAir(cID, airID)$
then [
$displaySelectedProposals(cID);$
$\pi htlID, airID.[clntResponse(cID, htlID, airID);$
if $htlID \neq NULL$ then $bookHtl(cID, htlID);$
if $airID \neq NULL$ then $bookAir(cID, airID)]$
] else
$reportFailure(cID)$
endIf

The argument cID stands for the client session ID, oC for the origin city, dC for destination city, and finally bD and eD represent the begin and end dates of the trip respectively. These are assumed to be known at the outset of the process. The process begins by querying 0 or more hotel and/or airline web services in any order, to check if they can fulfill the given request. The querying is done by two subprograms δ_{qryHtl} and δ_{qryAir}, with the former defined as follows:

$\delta_{qryHtl}(cID, dC, bD, eD) =$
$(\pi htlID.[\neg QrdHtlWS(cID, htlID, dC, bD, eD)?;$
$qryHtlWS(cID, htlID, dC, bD, eD);$
$\pi hC, stat.replyHtlWS(cID, htlID, hC, stat)]$

This program can query any hotel web service ($htlID$), as long as it has not been queried before. The web service's reply to the query is represented by the exogenous action $replyHtlWS$, where the parameters hC and $stat$ return the cost and status of the web service (OK if it is able to fulfill the request and $NotAvail$ otherwise) respectively. The fluent $RpldHtl(cID, htlID, dC, bD, eD, hC, stat, s)$ stores the reply from the hotel web service after it is received. The procedure δ_{qryAir} is defined similarly and the fluent $RpldAir$ stores the reply from the airline web service.

As it is querying hotel and/or airline web services, $\delta_{travelPlanner}$ also executes the subprograms $\delta_{pickHtl}$ and $\delta_{pickAir}$ to select 0 or more hotel and/or airline web service IDs, among those that replied positively, in any order. The procedure $\delta_{pickHtl}$ is defined as follows:

$\delta_{pickHtl}(cID, oC, dC, bD, eD) =$
$((\pi htlD, hC.[RpldHtl(cID, htlID, dC, bD, eD, hC, OK)?;$
$selectHtl(cID, htlID)])$

$\delta_{pickAir}$ is defined similarly. The fluents $SelHtl(cID, htlID, s)$ and $SelAir(cID, airID, s)$ become true for the selected hotels and airlines after actions $selectHtl(cID, htlID)$ and $selectAir(cID, airID)$ respectively.

If some hotels and/or airlines are selected, the process then presents them to user by executing the action $displaySelectedProposals(cID)$ after which the fluents $ProposedHtl(cID, htlID, s)$ and $ProposedAir(cID, airID, s)$ become true for those hotels and airlines. Subsequently, the client's response, represented by the exogenous action $clntResponse(cID, htlID, airID)$ is obtained, where $htlID$ and $airID$ may contain the chosen hotel and airline or $NULL$. Finally, the process will book the hotel and/or airline chosen by the client, if any. The fluents $BookedHtl(cID, htlID, s)$ and/or $BookedAir(cID, htlID, s)$ become true for the hotel and/or airline booked for the client. The process may also simply report failure without selecting any hotels and airlines.

In addition to a client's basic requirements such as begin and end date of the trip, the client may have further constraints and preferences. For instance, he may not want to fly with a certain airline, or he may want the travel planner to propose at least two hotels, if possible. He may also have budget constraints. Such constraints and preferences can be represented by another ConGolog program. Then, *supervision* can be utilized to personalize the travel planner process based on the given specification. For example, suppose that the client wants a hotel but not $HtlX$. We can represent this by the following supervision specification:

$\delta_{client1}(cID, oC, dC, bD, eD) =$
$[(\pi a.a)^*;$
$\exists htlID.\ ProposedHtl(cID, htlID) \vee$
$\forall htlID.IsHtl(htlID, dc) \wedge htlID \neq HtlX \supset$
$RpldHtl(cID, htlID, dC, bD, eD, hC, NotAvail)?;$
$(\pi a.a)^*]\ \&$
$\pi a.(a; \neg BookedHtl(cID, HtlX)?)^*$

This specification is satisfied if eventually (i.e., after some sequence of actions) a hotel has been proposed or all hotels other than $HtlX$ replied that no room is available, and $HtlX$ is never booked. In the above, $IsHtl$ is a relation that holds for all hotels in the destination city.

What executions can we get if we perform supervision on the generic travel planner agent with this specification? Suppose for simplicity that there are only three hotel web services ($HtlX$, $HtlY$, and $HtlZ$) and no airline web services. The supervision does not enforce any specific order in which these hotel web services are queried. Assuming the process queries $HtlY$ first and it is not able to fulfill the request, then $HtlZ$ must be queried. The process can only report failure if neither of $HtlY$ and $HtlZ$ have available rooms. In case $HtlY$ is able to fulfill the request, then $HtlZ$ may be queried. In either case, $HtlX$ may be queried as well. Once $HtlY$ and/or $HtlZ$ indicated availability, the travel planner must then select and present at least one of them to the client. After obtaining the client's response, the process must book the client's chosen hotel, if any. Supervision must ensure that $HtlX$ is not selected and proposed to the client. If it is, it might be chosen by the client, meaning that the agent must book it, which clearly violates the supervision specification. At the same time, supervision must leave as much freedom for the process as possible, so for example, the process can still query $HtlX$ and need not query $HtlZ$ in case $HtlY$ has a room available. What supervision produces will become clearer after we have given the formal definitions in Section 5. We will return to our example there.

4 Agents Executing Online

In our account of agent supervision, we want to accommodate agents that can acquire new knowledge about their en-

vironment 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:

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

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

$$\mathcal{D} \cup \mathcal{C} \cup Poss(repValP(1), S_1) \models Trans(nil; [P?; A] \mid [\neg P?; B], do(repValP(1), S_1), nil, do([repValP(1), A], S_1)).$$

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:

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

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
$\mathcal{D} \cup \mathcal{C} \cup \{Executable(do(\vec{a}, S_0))\} \models Trans(\delta, do(\vec{a}, S_0), \delta', do(A(\vec{n}), do(\vec{a}, S_0)))$
or $A \in \mathcal{A}^e,\ \vec{n} \in \mathcal{N}^k$ and
$\mathcal{D} \cup \mathcal{C} \cup \{Executable(do(\vec{a}, S_0)), Trans(\delta, do(\vec{a}, S_0), \delta', do(A(\vec{n}), do(\vec{a}, S_0)))\}$ 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 $Executable(do([\vec{a}, A(\vec{n})], S_0))$ will be added to the theory when it is queried in later transitions, thus incorporating the fact that $Poss(A(\vec{n}), do(\vec{a}, S_0))$ 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
$\mathcal{D} \cup \mathcal{C} \cup \{Executable(do(\vec{a}, S_0))\} \models Final(\delta, do(\vec{a}, S_0))$.

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 actions $\vec{a}$, 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 re-*

maining 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 $\mathcal{D} \cup \mathcal{C} \cup \{Executable(do(\vec{a}, S_0)),$
$Trans(\delta, do(\vec{a}, S_0), \delta', do([\vec{a}, A(\vec{n})], S_0))$ is satisfiable,
then there exists a program δ' such that
$\mathcal{D} \cup \mathcal{C} \cup \{Executable(do([\vec{a}, A(\vec{n})], S_0))\} \models$
$Trans(\delta, do(\vec{a}, S_0), \delta', do([\vec{a}, A(\vec{n})], S_0))$.

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.

We can show that:[2]

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., $\mathcal{D} \cup \mathcal{C} \models SituationDetermined(\delta^i, S_0)$, and if σ always knows the remaining program after an exogenous action, then σ is online SD.*

Being online SD is an important property. It means that for any sequence of actions that the agent can perform in an online execution, there is a unique resulting agent configuration, i.e., agent belief state and remaining program. From now on, we assume that the agent 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: $\mathcal{GR}(\sigma) = \{\vec{a} \mid \exists c, c', \vec{a'}.c^i \rightarrow^*_{\vec{a}} c \wedge c \rightarrow^*_{\vec{a'}} c' \wedge c'^{\checkmark}\}$.

5 Online Agent Supervision

5.1 Motivation

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:

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

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) $mps_{\mathit{off}}(\delta^i, \delta^s, s)$ of the agent behavior δ^i which fulfills the supervision specification δ^s as:

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

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.[3]

DLM show that their notion of offline MPS, $mps_{\mathit{off}}(\delta^i, \delta^s, s)$, 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:

[2] Proofs of this and all other results in the paper appear in appendix.

[3] 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.

$$\mathcal{D} \cup \mathcal{C} \models (P(S_0) \supset mps_{\mathit{offl}}(\delta_4^i, \delta_4^s, S_0) = \mathbf{set}(\{[B, D]\})) \wedge (\neg P(S_0) \supset mps_{\mathit{offl}}(\delta_4^i, \delta_4^s, S_0) = \mathbf{set}(\{[A, D]\}))$$

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_4^s in S_0 is $\{[B, D], [A, C]\}$ and $\mathbf{set}(\{[A, C]\})$ is not controllable wrt δ_4^i 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_4^s in S_0 is $\{[A, D], [B, C]\}$ and $\mathbf{set}(\{[B, C]\})$ is not controllable wrt δ_4^i in S_0. Since it is not known if P holds, it seems that a correct supervisor should neither allow A nor B. □

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. The difference between offline MPS and online MPS is analogous to the difference between classical plans and conditional plans that include sensing in the planning literature (Ghallab, Nau, and Traverso 2004).

5.2 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:

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

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. We can show that:

Theorem 2 *For 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, where $\langle \mathcal{D}, \delta^s \rangle$ is also online SD, the following properties hold:*

1. *$mps_{onl}(\delta^s, \sigma)$ always exists and is unique;*
2. *$\langle \mathcal{D}, mps_{onl}(\delta^s, \sigma) \rangle$ is online SD;*
3. *$mps_{onl}(\delta^s, \sigma)$ is online controllable wrt σ;*
4. *for every possible online controllable supervision specification $\hat{\delta^s}$ for σ such that $\mathcal{CR}(\langle \mathcal{D}, \delta^i \& \hat{\delta^s} \rangle) \subseteq \mathcal{CR}(\langle \mathcal{D}, \delta^i \& \delta^s \rangle)$, we have that $\mathcal{CR}(\langle \mathcal{D}, \delta^i \& \hat{\delta^s} \rangle) \subseteq \mathcal{CR}(\langle \mathcal{D}, mps_{onl}(\delta^s, \sigma) \rangle)$, i.e., mps_{onl} is maximally permissive;*
5. *$\mathcal{RR}(\langle \mathcal{D}, mps_{onl}(\delta^s, \sigma) \rangle) = \mathcal{GR}(\langle \mathcal{D}, mps_{onl}(\delta^s, \sigma) \rangle)$, i.e., $mps_{onl}(\delta^s, \sigma)$ is non-blocking.*

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. $mps_{onl}(\delta_4^s, \langle \mathcal{D}, \delta_4^i \rangle) = \mathbf{set}(\{\epsilon\})$. □

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 $\delta_5^i = Sense_P; \delta_4^i$. We can show that:

$$\mathcal{D} \cup \mathcal{C} \models (P(S_0) \supset mps_{\mathit{offl}}(\delta_5^i, \delta_4^s, S_0) = \mathbf{set}(\{[QryIfP, repValP(1), B, D]\})) \wedge (\neg P(S_0) \supset mps_{\mathit{offl}}(\delta_5^i, \delta_4^s, S_0) = \mathbf{set}(\{[QryIfP, repValP(0), A, D]\}))$$

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:

$$mps_{onl}(\delta_4^s, \langle \mathcal{D}, \delta_5^i \rangle) = \mathbf{set}(\{[QryIfP, repValP(1), B, D], [QryIfP, repValP(0), A, D]\})$$

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.

5.3 Online Supervision Operator

We can also introduce a meta-theoretic version of a synchronous concurrency operator $\delta^i \&^{onl}_{A_u} \delta^s$ that captures the *maximally permissive execution of an agent $\langle \mathcal{D}, \delta^i \rangle$ under online supervision for specification δ^s*. Without loss of generality, we assume that both δ^i and δ^s start with a common controllable action (if not, it is trivial to add a dummy action in front of both so as to fulfill the requirement). We define $\delta^i \&^{onl}_{A_u} \delta^s$ by extending the online transition relation as follows:

$$\langle \delta^i \&^{onl}_{A_u} \delta^s, \vec{a} \rangle \rightarrow_a \langle \delta^{i'} \&^{onl}_{A_u} \delta^{s'}, \vec{a}a \rangle$$
if and only if
$$\langle \delta^i, \vec{a} \rangle \rightarrow_a \langle \delta^{i'}, \vec{a}a \rangle \text{ and } \langle \delta^s, \vec{a} \rangle \rightarrow_a \langle \delta^{s'}, \vec{a}a \rangle \text{ and}$$
$$\text{if } \mathcal{D} \cup \{Executable(do(\vec{a}, S_0))\} \models \neg A_u(a, do(\vec{a}, S_0))$$
$$\text{then for all } \vec{a_u} \text{ s.t. } \mathcal{D} \cup \{Executable(do(\vec{a}\vec{a_u}, S_0)),$$
$$A_u(\vec{a_u}, do(\vec{a}a, S_0))\} \text{ is satisfiable,}$$
$$\text{if } \vec{a}a\vec{a_u} \in \mathcal{GR}(\langle \mathcal{D}, [\vec{a}; \delta^i] \rangle), \text{then } \vec{a}a\vec{a_u} \in \mathcal{GR}(\langle \mathcal{D}, [\vec{a}; \delta^s] \rangle).$$

where $A_u(\vec{a_u}, s)$, means that action sequence $\vec{a_u}$ is uncontrollable in situation s, and is inductively defined on the length of $\vec{a_u}$ as the smallest predicate such that: *(i)* $A_u(\epsilon, s) \equiv \texttt{true}$; *(ii)* $A_u(a_u\vec{a_u}, s) \equiv A_u(a_u, s) \wedge A_u(\vec{a_u}, do(a_u, s))$. Thus, the online maximally permissive supervised execution of δ^i for the specification δ^s is allowed to perform action a in situation $do(\vec{a}, S_0)$ if a is allowed by both δ^i and δ^s and moreover, if a is known to be controllable, then for every sequence of actions $\vec{a_u}$ not known to be controllable, if $\vec{a_u}$ may be performed by δ^i right after a on one of its complete runs, then it must also be allowed by δ^s (on one of its complete runs). Essentially, a controllable action a by the agent must be forbidden if it can be followed by some sequence of actions not known to be controllable that violates the specification.

The final configurations are extended as follows:

$$(\langle \delta^i \&^{onl}_{A_u} \delta^s, \vec{a} \rangle)^{\checkmark} \text{ if and only if } (\langle \delta^i, \vec{a} \rangle)^{\checkmark} \text{ and } (\langle \delta^s, \vec{a} \rangle)^{\checkmark}$$

We can show that firstly, if both the agent and supervision specification processes are online SD, then so is the program obtained using the online supervision operator, and moreover, this program is controllable wrt to the agent process:

Theorem 3

1. *If $\langle \mathcal{D}, \delta^s \rangle$ and $\langle \mathcal{D}, \delta^i \rangle$ are online SD, then so is $\langle \mathcal{D}, \delta^i \&^{onl}_{A_u} \delta^s \rangle$.*
2. *$\delta^i \&^{onl}_{A_u} \delta^s$ is online controllable wrt $\langle \mathcal{D}, \delta^i \rangle$.*

Moreover, the complete runs of the program obtained using the online supervision operator are exactly the same the complete runs generated under synchronous concurrency of the agent and $mps_{onl}(\delta^s, \sigma)$:

Theorem 4

$$\mathcal{CR}(\langle \mathcal{D}, \delta^i \&^{onl}_{A_u} \delta^s \rangle) = \mathcal{CR}(\langle \mathcal{D}, \delta^i \,\&\, mps_{onl}(\delta^s, \sigma) \rangle).$$

While $\delta^i \&^{onl}_{A_u} \delta^s$ and $mps_{onl}(\delta^s, \sigma)$ have the same complete runs, it is not the case that they have the same set of partial runs. In fact in general, $\mathcal{RR}(\langle \mathcal{D}, \delta^i \&^{onl}_{A_u} \delta^s \rangle) \neq \mathcal{GR}(\langle \mathcal{D}, \delta^i \&^{onl}_{A_u} \delta^s \rangle)$, i.e., the program obtained using the online supervision operator is not necessarily non-blocking. This is in contrast with $mps_{onl}(\delta^s, \sigma)$, which is guaranteed to be non-blocking (Theorem 2).

Example 4 Suppose we have the agent program:

$$\delta^i_6 = (A \mid [B; C; (U1 \mid U2; D)])$$

where all actions except $U1$ and $U2$ are ordinary and controllable. Moreover, assume the supervision specification is:

$$\delta^s_6 = (\pi a.a \neq D?; a)^*$$

i.e. any action except D can be performed. The online MPS for this agent is simply $\mathbf{set}(\{A\})$, since $\mathcal{CR}(\langle \mathcal{D}, \delta^s_6 \rangle) = \{A, [B, C, U1]\}$ and $\mathbf{set}(\{[B, C, U1]\})$ is not controllable wrt δ^i_6. However, under online supervised execution, the agent may execute the action B. We have $\langle \delta^i_6 \&^{onl}_{A_u} \delta^s_6, \epsilon \rangle \rightarrow_B \langle \delta'_6 \&^{onl}_{A_u} \delta^s_6, B \rangle$ where δ'_6 is what remains from δ^i_6 after executing B. The resulting program is not final in $do(B, S_0)$, yet there is no transition from this state, as the action C could be followed by the uncontrollable action $U2$ and it is not possible to ensure successful completion of the process, as the action D is not allowed. Thus, one must do lookahead search over online executions of $\delta^i_6 \&^{onl}_{A_u} \delta^s_6$ to obtain good/complete runs. We propose such a search/lookahead construct next. □

5.4 Search Over a Controllable Process

When we have a specification/process δ^s that is controllable with respect to an agent $\langle \mathcal{D}, \delta^i \rangle$ (like for instance, $\delta^i \&^{onl}_{A_u} \delta^s$), for any choice of uncontrollable action that is on a good run of δ^i, it is always possible to find a way to continue executing δ^s until the process successfully completes. We can define a search construct[4] that is applicable to these cases. It makes an arbitrary choice of action that is on a good run of δ^i when the action is not known to be controllable, while still only performing actions that are on a good run of δ^s otherwise. We call this construct *weak online search* $\Sigma^w_{onl}(\delta^s, \delta^i)$ and define it (metatheoretically) as follows:[5]

$$\langle \Sigma^w_{onl}(\delta^s, \delta^i), \vec{a} \rangle \rightarrow_a \langle \Sigma^w_{onl}(\delta^{s'}, \delta^{i'}), \vec{a}a \rangle$$
if and only if
$$\langle \delta^s, \vec{a} \rangle \rightarrow_a \langle \delta^{s'}, \vec{a}a \rangle \text{ and } \langle \delta^i, \vec{a} \rangle \rightarrow_a \langle \delta^{i'}, \vec{a}a \rangle \text{ and}$$
$$\text{if } \mathcal{D} \cup \{Executable(\vec{a}, S_0)\} \models \neg A_u(a, do(\vec{a}, S_0))$$
$$\text{then } \vec{a}a \in \mathcal{GR}(\langle \mathcal{D}, [\vec{a}a; \delta^{s'}] \rangle)$$
$$\text{else } \vec{a}a \in \mathcal{GR}(\langle \mathcal{D}, [\vec{a}a; \delta^{i'}] \rangle)$$

The final configurations are extended as follows:

$$(\langle \Sigma^w_{onl}(\delta^s, \delta^i), \vec{a} \rangle)^{\checkmark} \text{ iff } (\langle \delta^s, \vec{a} \rangle)^{\checkmark} \text{ and } (\langle \delta^i, \vec{a} \rangle)^{\checkmark}$$

[4] In IndiGolog a simple type of search is provided that only allows a transition if the remaining program can be executed to reach a final state in all the offline executions (De Giacomo and Levesque 1999). However, this search does not deal with sensing and online executions.

[5] Since δ^i can include exogenous actions, in general, executions of the process could actually perform exogenous actions that are not on a good run of δ^i. However, in this paper we are interested in the case where the exogenous actions are mainly sensor reports and external requests (rather than the actions of an adversary) and assume that this won't occur. Handling adversarial nondeterminism in δ^i is left for future work.

It is easy to show that:

Theorem 5 *If $\langle \mathcal{D}, \delta^s \rangle$ and $\langle \mathcal{D}, \delta^i \rangle$ are online SD, than so is $\langle \mathcal{D}, \Sigma^w_{onl}(\delta^s, \delta^i) \rangle$.*

Now, we can show that the weak online search construct has many nice properties when the process is controllable:

Theorem 6 *Suppose that we have an agent $\langle \mathcal{D}, \delta^i \rangle$, and a supervision specification δ^s which are online SD. Suppose also that δ^s is online controllable with respect to $\langle \mathcal{D}, \delta^i \rangle$, and that $\mathcal{CR}(\langle \mathcal{D}, \delta^s \rangle) \subseteq \mathcal{CR}(\langle \mathcal{D}, \delta^i \rangle)$. Then we have that:*

1. *$\mathcal{CR}(\langle \mathcal{D}, \Sigma^w_{onl}(\delta^s, \delta^i) \rangle) = \mathcal{CR}(\langle \mathcal{D}, \delta^s \rangle)$, i.e. the complete runs of $\Sigma^w_{onl}(\delta^s, \delta^i)$ are the complete runs of δ^s.*
2. *If $\mathcal{CR}(\langle \mathcal{D}, \delta^s \rangle) \neq \emptyset$, then $\mathcal{RR}(\langle \mathcal{D}, \Sigma^w_{onl}(\delta^s, \delta^i) \rangle) = \mathcal{GR}(\langle \mathcal{D}, \delta^s \rangle)$, i.e., the partial runs of $\Sigma^w_{onl}(\delta^s, \delta^i)$ are the good runs of δ^s.*
3. *If $\mathcal{CR}(\langle \mathcal{D}, \delta^s \rangle) \neq \emptyset$, then $\mathcal{RR}(\langle \mathcal{D}, \Sigma^w_{onl}(\delta^s, \delta^i) \rangle) = \mathcal{GR}(\langle \mathcal{D}, \Sigma^w_{onl}(\delta^s, \delta^i) \rangle)$, i.e., partial runs must be good runs, and the resulting program is "non blocking".*

It is also easy to show that none of these properties hold for arbitrary non-controllable processes.

Now we can show that if we apply this weak lookahead search to $\delta^i \ \&^{onl}_{A_u} \ \delta^s$, we obtain a program that has the same partial runs as $mps_{onl}(\delta^s, \sigma)$ and is thus non-blocking:

Theorem 7

$$\mathcal{RR}(\langle \mathcal{D}, \Sigma^w_{onl}(\delta^i \ \&^{onl}_{A_u} \ \delta^s, \delta^i) \rangle) = \mathcal{RR}(\langle \mathcal{D}, \delta^i \ \& \ mps_{onl}(\delta^s, \sigma) \rangle).$$

If we apply the weak online search construct over $\delta^i_6 \ \&^{onl}_{A_u} \ \delta^s_6$ in Example 4, we no longer have an online transition involving action B; the only possible online transition is $\langle \Sigma^w_{onl}(\delta^i_6 \&^{onl}_{A_u} \delta^s_6, \delta^i_6), \epsilon \rangle \rightarrow_A \langle \Sigma^w_{onl}(nil \ \&^{onl}_{A_u} \delta^s_6, nil), A \rangle$ where action A is performed, after which we have $(\langle \Sigma^w_{onl}(nil \ \&^{onl}_{A_u} \ \delta^s_6, nil), A \rangle)^{\checkmark}$.

5.5 Travel Planning Example Revisited

Let's return to the travel planning example of Section 3. There we presented a generic travel planning agent/process $\delta_{travelPlanner}$ and a simple customization supervision specification $\delta_{client1}$ where the client requires at least one hotel to be proposed provided one has a room available and that hotel $HtlX$ is never booked. What is the online MPS for this? Essentially, the supervised process cannot report failure unless it has queried all hotels at the destination other than $HtlX$ and none has a room available; if some such hotel has a room available, then one such hotel must be selected; moreover, $HtlX$ cannot be selected because if it is selected, then it must be proposed, and then the client may choose it and then it must be booked, thus violating the supervision specification.

The resulting online MPS can be obtained by inserting the following program in the $\delta_{travelPlanner}$ process right after the lines for querying/selecting airlines and/or hotels:

$$\begin{array}{l} \neg SelHtl(cID, HtlX) \wedge \\ (\exists htlID.\ ProposedHtl(cID, htlID) \vee \\ \ \forall htlID.IsHtl(htlID, dc) \wedge htlID \neq HtlX \supset \\ \ RpldHtl(cID, htlID, dC, bD, eD, hC, NotAvail)?; \end{array}$$

Observe that the intersection of the process and supervision specification $\delta_{travelPlanner} \ \& \ \delta_{client1}$ does not give the right result in this case. In particular, it allows the agent to select $HtlX$ and so it is not controllable for $\delta_{travelPlanner}$, since if $HtlX$ is selected, then it must be proposed, and then the client may choose it, in which case it it must be booked and the supervision specification process will not terminate successfully.

Now let's look at a second example of customization. Suppose the client has a given budget (B) and does not want the total cost of hotels (hC) and flights (aC) proposed to him to be over this budget. One could represent this as follows:

$$\begin{array}{l} \delta_{client2}(cID, oC, dC, bD, eD) = \\ \ \pi a.(a; \neg\exists\ htlID.ProposedHtl(cID, htlID) \wedge \\ \ \ RpldHtl(cID, htlID, dC, bD, eD, hC, OK) \wedge \\ \ \ \forall airID.ProposedAir(cID, airID) \wedge \\ \ \ RpldAir(cID, airID, oC, dC, bD, eD, aC, OK) \supset \\ \ \ hC + aC > B?)^* \end{array}$$

This says the process should never have proposed a hotel such that with every proposed flight the total cost of hotel and flight exceeds the budget (normally, this would be combined with other constraints). Intuitively, the online MPS for this specification is such that the agent can do anything it wants, as long as it never selects any flight, or if it does select a flight, then it does not select a hotel unless it knows of an available flight such that the combined hotel and flight cost is within budget and then such a flight must be selected before the selected proposals are displayed. Note that the agent may choose to simply report failure or avoid selecting any hotel. Again, the intersection of the agent process and supervision specification is not controllable with respect to the former, and is not the MPS. If a flight has been selected and a hotel is selected without the agent knowing of an available flight such that the combined hotel and flight cost is within budget, then the responses of airline websites about flight availability and cost, which are uncontrollable, may not yield such a flight, and the process may have no way of satisfying the constraint and completing successfully.

In a similar way, it is possible to perform configuration of the generic travel planner process for a travel service provider. For example, suppose we have a service provider for business executives that only offers 4 or 5 star hotels which are located in a downtown area. We can define a supervision specification for this configuration task as follows:

$$\begin{array}{l} \delta_{bzProfileConfig}(cID, oC, dC, bD, eD) = \\ \ \pi a.(a; [\neg\exists htlID.QrdHtlWS(cID, htlID, dC, bD, eD) \wedge \\ \ \neg BzHtlProfile(CID, htlID))]?)^* \\ \text{where } BzHtlProfile(cID, htlID) = \\ \ [HtlRating(htlID, 4) \vee HtlRating(htlID, 5)] \wedge \\ \ HtlArea(htlID, Downtown) \end{array}$$

Personalization of the travel planner process for a client can be done simultaneously with configuration for the service provider. In this case, the supervision specification will be the intersection of the service provider's configuration specification with the the client's personalization specification.

6 Discussion

One popular approach to automated service composition (McIlraith and Son 2002; Sohrabi, Prokoshyna, and

McIlraith 2006) involves customizing a generic ConGolog process based on the user's constraints and preferences. (Sardiña and De Giacomo 2009) on the other hand, synthesizes a controller that orchestrates the concurrent execution of library of available (nondeterministic) ConGolog programs to realize a target program not in the library. However, they assume complete information on the initial situation, and their controller is not maximally permissive. In related work, (De Giacomo, Patrizi, and Sardina 2013) synthesize a *controller generator* that represents all possible compositions of the target behavior and may adapt reactively based on runtime feedback. In (Yadav et al. 2013), optimal realization of the target behavior (in the presence of uncontrollable exogenous events) is considered when its full realization is not possible. In contrast to agent supervision, the latter does not assume the controllability of events to be dynamic. Moreover, both approaches model behaviors/services as (nondeterministic) finite state transition systems.

Also related, is the approach in (Alechina et al. 2015) that regulates multiagent systems using regimented norms. A transition system describes the behavior of a (multi-) agent system and a guard function (characterized by LTL formulae with past operators) can enable/disable options that (could) violate norms after a system history (possibly using bounded lookahead). This work does not consider uncontrollable events. Finally, we would like to mention the work in (Aucher 2014), which reformulates the results of supervisory control theory in terms of model checking problems in an epistemic temporal logic. Our work differs from these approaches in that due to its first-order logic foundations, it can handle infinite object domains and infinite states. It also enables users to express the system model and the specifications in a high-level expressive language.

In the literature on supervisory control theory, nondeterminism in a discrete event system is often considered as the result of lack of information (e.g. partial observation or unmodeled internal dynamics of a system). Nondeterministic plants e.g. (Kumar and Shayman 1997), nondeterministic supervisors e.g. (Inan 1994) and nondeterministic specifications e.g. (Zhou, Kumar, and Jiang 2006) have been studied.

In this paper, building upon DLM's proposal (De Giacomo, Lespérance, and Muise 2012) and earlier work on supervisory control of discrete event systems (Wonham and Ramadge 1987; Wonham 2014; Cassandras and Lafortune 2008), we have developed an account of supervision for agents that execute online and can acquire new information through sensing and exogenous actions as they operate. In future work, we will examine how we can relax the assumption that the supervisor and supervised agent share the same belief state. Furthermore, we would like to generalize our approach to support supervision of complex multiagent systems, where local supervisors control individual agents or teams of agents that play particular roles so as to ensure that the whole system satisfies its specification. We conclude by mentioning that if the object domain is finite, then ConGolog programs assume only a finite number of possible configurations, and in this case, 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 develop effective techniques for synthesizing supervisors for agents that use bounded action theories, where the agent only has beliefs about a finite number of objects in any situation, even though it may deal with an infinite number of objects over an infinite run (De Giacomo, Lespérance, and Patrizi 2013; De Giacomo et al. 2014); verification of temporal properties over such agents is known to be decidable.

Acknowledgments

We acknowledge the support of the Sapienza 2015 project "Immersive Cognitive Environments" and the National Science and Engineering Research Council of Canada under grant RGPIN-2015-03756 Specification, Verification, and Synthesis of Autonomous Adaptive Agents.

References

Alechina, N.; Bulling, N.; Dastani, M.; and Logan, B. 2015. Practical Run-Time Norm Enforcement with Bounded Lookahead. In Weiss, G.; Yolum, P.; Bordini, R. H.; and Elkind, E., eds., *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*, 443–451. ACM.

Aucher, G. 2014. Supervisory control theory in epistemic temporal logic. In Bazzan, A. L. C.; Huhns, M. N.; Lomuscio, A.; and Scerri, P., eds., *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014*. IFAAMAS/ACM.

Cassandras, C. G., and Lafortune, S. 2008. *Introduction to Discrete Event Systems, Second Edition*. Springer.

De Giacomo, G., and Levesque, H. J. 1999. An incremental interpreter for high-level programs with sensing. In Levesque, H. J., and Pirri, F., eds., *Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter*. Springer Berlin Heidelberg. 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 Schaub, T.; Friedrich, G.; and O'Sullivan, B., eds., *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, 369–374. IOS Press.

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

De Giacomo, G.; Lespérance, Y.; and Muise, C. J. 2012. On supervising agents in situation-determined ConGolog. In van der Hoek, W.; Padgham, L.; Conitzer, V.; and Winikoff, M., eds., *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012*, 1031–1038. IFAAMAS.

De Giacomo, G.; Lespérance, Y.; and Patrizi, F. 2013. Bounded epistemic situation calculus theories. In Rossi, F., ed., *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*. IJCAI/AAAI.

De Giacomo, G.; Lespérance, Y.; and Pearce, A. R. 2010. Situation calculus based programs for representing and reasoning about game structures. In Lin, F.; Sattler, U.; and Truszczynski, M., eds.,

Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010. AAAI Press.

De Giacomo, G.; Patrizi, F.; and Sardina, S. 2013. Automatic behavior composition synthesis. *Artificial Intelligence* 196:106–142.

Fritz, C., and McIlraith, S. A. 2006. Decision-theoretic Golog with qualitative preferences. In Doherty, P.; Mylopoulos, J.; and Welty, C. A., eds., *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, 153–163. AAAI Press.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. San Francisco, CA, USA: Morgan Kaufmann/Elsevier.

Inan, K. 1994. Nondeterministic supervision under partial observations. In Cohen, G., and Quadrat, J.-P., eds., *11th International Conference on Analysis and Optimization of Systems Discrete Event Systems*, volume 199 of *Lecture Notes in Control and Information Sciences*. Springer Berlin Heidelberg. 39–48.

Kumar, R., and Shayman, M. A. 1997. Centralized and Decentralized Supervisory Control of Nondeterministic Systems Under Partial Observation. *SIAM Journal on Control and Optimization* 35(2):363–383.

Lespérance, Y.; De Giacomo, G.; and Ozgovde, A. N. 2008. A model of contingent planning for agent programming languages. In Padgham, L.; Parkes, D. C.; Müller, J. P.; and Parsons, S., eds., *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, May 12-16, 2008, Volume 1*, 477–484. IFAAMAS.

Liaskos, S.; Khan, S. M.; Litoiu, M.; Jungblut, M. D.; Rogozhkin, V.; and Mylopoulos, J. 2012. Behavioral adaptation of information systems through goal models. *Information Systems* 37(8):767–783.

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

McIlraith, S. A., and Son, T. C. 2002. Adapting Golog for composition of semantic web services. In *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France, April 22-25, 2002*, 482–496. Morgan Kaufmann.

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

Sardiña, S., and De Giacomo, G. 2009. Composition of ConGolog programs. In Boutilier, C., ed., *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, 904–910.

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. *Artificial Intelligence* 144(1-2):1–39.

Sohrabi, S.; Prokoshyna, N.; and McIlraith, S. A. 2006. Web service composition via generic procedures and customizing user preferences. In *Proceedings of the 5th International Semantic Web Conference (ISWC-06)*, volume 4273, 597–611. Springer.

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.

Wonham, W. 2014. *Supervisory Control of Discrete-Event Systems*. University of Toronto, 2014 edition.

Yadav, N.; Felli, P.; De Giacomo, G.; and Sardina, S. 2013. Supremal Realizability of Behaviors with Uncontrollable Exogenous Events. In Rossi, F., ed., *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, 1176–1182. IJCAI/AAAI.

Zhou, C.; Kumar, R.; and Jiang, S. 2006. Control of nondeterministic discrete-event systems for bisimulation equivalence. *IEEE Transactions on Automatic Control* 51(5):754–765.

A Proofs

A.1 Some Lemmas about the set Construct

It is straightforward to show that:

Proposition 8

If $\langle \mathbf{set}(E), \vec{a}\rangle \rightarrow_a c$ *and* $\langle \mathbf{set}(E), \vec{a}\rangle \rightarrow_a c'$*, then* $c = c'$.

Proof. If a is not an exogenous action, then it must be known in $do(\vec{a}, S_0)$ that a is executable and there must be some action sequence in E that starts with a. The unique new configuration c is then $\langle \mathbf{set}(E'), \vec{a}a\rangle$ with $E' = \{\vec{b} \mid a\vec{b} \in E\}$. If a is an exogenous action, then it must be consistent in $do(\vec{a}, S_0)$ that a is executable and some action sequence in E must start with a; the the unique new configuration c is just as in the previous case. □

It then trivially follows that:

Corollary 9

Any agent $\langle \mathcal{D}, \delta^i\rangle$ *with the initial program* $\delta^i = \mathbf{set}(E)$ *is online situation determined.*

The following lemmas about the $\mathbf{set}(E)$ construct will also prove useful later:

Lemma 10 *If* $\vec{a} \in \mathcal{RR}(\langle \mathcal{D}, \mathbf{set}(E)\rangle)$*, then there exists* $\vec{b}$ *such that* $\vec{a}\vec{b} \in E$.

Proof (sketch). By induction on the length of $\vec{a}$. □

Lemma 11 *If* $\langle \delta^i, \epsilon\rangle \rightarrow_{\vec{a}}^* c$*, then* $\langle \mathbf{set}(E \cup \{\vec{a}\vec{b}\}), \epsilon\rangle \rightarrow_{\vec{a}}^* \langle \mathbf{set}(E' \cup \{\vec{b}\}), \vec{a}\rangle$.

Proof (sketch). By induction on the length of $\vec{a}$, noticing that the antecedent implies that the agent actions are known to be executable and the exogenous actions are thought to be possibly executable, and so the transitions exist. □

Lemma 12

If $\vec{a} \in \mathcal{RR}(\langle \mathcal{D}, \delta^i\rangle)$ *and* $\vec{a} \in E$*, then* $\vec{a} \in \mathcal{CR}(\langle \mathcal{D}, \mathbf{set}(E)\rangle)$.

Proof. Assume that the antecedent. Since $\vec{a} \in \mathcal{RR}(\langle \mathcal{D}, \delta^i\rangle)$, there exists c such that $\langle \delta^i, \epsilon\rangle) \rightarrow_{\vec{a}}^* c$. Since $\vec{a} \in E$, it then follows by Lemma 11 that $\langle \mathbf{set}(E), \epsilon\rangle \rightarrow_{\vec{a}}^* \langle \mathbf{set}(E'), \vec{a}\rangle$ with $\epsilon \in E'$. Thus $Final(\langle \mathbf{set}(E'), \vec{a}\rangle)$, and therefore $\vec{a} \in \mathcal{CR}(\langle \mathcal{D}, \mathbf{set}(E)\rangle)$. □

A.2 Online Situation-Determined Agents Proofs

Proof of Theorem 1

Proof. By induction on the length of the action (and online transition) sequence. If the sequence is empty, the result trivially follows. Assume that the result holds for all action sequences of length k (IH). Suppose that $c^i \rightarrow^*_{\vec{a}a} \langle \delta_1, \vec{a}a\rangle$ and $c^i \rightarrow^*_{\vec{a}a} \langle \delta_2, \vec{a}a\rangle$ and $\delta_1 \neq \delta_2$ with $\vec{a}a$ of length $k+1$. It follows by the definition of online execution and the IH that $c^i \rightarrow^*_{\vec{a}} \langle \delta, \vec{a}\rangle$ and $\langle \delta, \vec{a}\rangle \rightarrow_a \langle \delta_1, \vec{a}a\rangle$ and $\langle \delta, \vec{a}\rangle \rightarrow_a \langle \delta_2, \vec{a}a\rangle$ and $\delta_1 \neq \delta_2$. If action a is not exogenous, then by the definition of online transition $\mathcal{D} \cup \mathcal{C} \cup \{Executable(do(\vec{a}, S_0))\} \models$ *Trans*$(\delta, a, \delta', do(\vec{a}, S_0))$ (where δ' is unique), which contradicts $\delta_1 \neq \delta_2$. If action a is exogenous, then both δ_1 and δ_2 are satisfiable as remaining programs. In each case, $Poss(a, do(\vec{a}, S_0))$ is also satisfiable.

Since σ always *knows* the remaining program after an exogenous action, we have that $\mathcal{D} \cup \mathcal{C} \cup \{Executable(do(\vec{a}a, S_0))\} \models$ *Trans*$(\delta, a, \delta', do(\vec{a}, S_0))$ (where δ' is unique), which contradicts $\delta_1 \neq \delta_2$. □

A.3 Online MPS Proofs

Proof of Theorem 2

Proof. [For claims 1, 3, and 4, we make essentially the same argument as in the offline case.]

Claim 1. The online MPS exists as $\mathbf{set}(\emptyset)$ satisfies the conditions to be included in $mps_{onl}(\delta^s, \sigma)$. Uniqueness follows from the existence of a supremal element.

Claim 2. Trivially follows from Corollary 9.

Claim 3. It suffices to show that for all $\vec{a}$ and a_u such that $\vec{a} \in \mathcal{GR}(\langle \mathcal{D}, mps_{onl}(\delta^s, \sigma)\rangle)$ and $\mathcal{D} \cup \{Executable(do(\vec{a}, S_0))\} \not\models \neg A^u(a^u, do(\vec{a}, S_0))$, we have that if $\vec{a}a_u \in \mathcal{GR}(\sigma)$ then $\vec{a}a_u \in \mathcal{GR}(\langle \mathcal{D}, mps_{onl}(\delta^s, \sigma)\rangle)$. Indeed, if $\vec{a} \in \mathcal{GR}(\langle \mathcal{D}, mps_{onl}(\delta^s, \sigma)\rangle)$ then there is an online controllable supervision specification $\mathbf{set}(E)$ such that $\vec{a} \in \mathcal{GR}(\langle \mathcal{D}, \mathbf{set}(E)\rangle)$. $\mathbf{set}(E)$ being online controllable wrt σ, if $\vec{a}a_u \in \mathcal{GR}(\sigma)$ then $\vec{a}a_u \in \mathcal{GR}(\langle \mathcal{D}, \mathbf{set}(E)\rangle)$, but then $\vec{a}a_u \in \mathcal{GR}(\langle \mathcal{D}, mps_{onl}(\delta^s, \sigma)\rangle)$.

Claim 4. This follows immediately from the definition of $mps_{onl}(\delta^s, \sigma)$, by noticing that $\mathcal{CR}(\langle \mathcal{D}, \delta^i \& \hat{\delta^s}\rangle) = \mathcal{CR}(\langle \mathcal{D}, \delta^i \& \mathbf{set}(E_{\hat{\delta^s}})\rangle)$, and observing that $mps_{onl}(\delta^s, \sigma)$ is essentially the union of such controllable $\mathbf{set}(E_{\hat{\delta^s}})$.

Claim 5. Suppose that $\vec{a} \in \mathcal{RR}(\langle \mathcal{D}, mps_{onl}(\delta^s, \sigma)\rangle)$. By the definition of mps_{onl}, $mps_{onl}(\delta^s, \sigma) = \mathbf{set}(E)$ where $E \subseteq \mathcal{CR}(\langle \mathcal{D}, \delta^i \ \& \ \delta^s\rangle)$. Since $\vec{a} \in \mathcal{RR}(\langle \mathcal{D}, \mathbf{set}(E)\rangle)$, by Lemma 10 there exists $\vec{b}$ such that $\vec{a}\vec{b} \in E$. Since $\vec{a}\vec{b} \in \mathcal{CR}(\langle \mathcal{D}, \delta^i \ \& \ \delta^s\rangle)$, by Lemma 12, we have that $\vec{a}\vec{b} \in \mathcal{CR}(\langle \mathcal{D}, \mathbf{set}(E)\rangle)$. It then follows by the definition of $\mathcal{GR}$ that $\vec{a} \in \mathcal{GR}(\langle \mathcal{D}, mps_{onl}(\delta^s, \sigma)\rangle)$. □

A.4 Supervision Operator Proofs

Proof of Theorem 3

Proof. Claim 1. By induction on the length of the action (and online transition) sequence. If the sequence is empty, the result trivially follows. Assume that the result holds for all action sequences $\vec{a}$ of length k (IH). We need to show that the result holds for all action sequences $\vec{a}a$ of length $k+1$.

Assume $\langle \delta^i \&^{onl}_{Au} \delta^s, \vec{a}\rangle \rightarrow_a c'$ where $c' = \langle \delta^{i'} \&^{onl}_{Au} \delta^{s'}, \vec{a}a\rangle$. Due to the way online transition is defined for $\&^{onl}_{Au}$, configuration c' can only be reached if we have both $\langle \delta^i, \vec{a}\rangle \rightarrow_a \langle \delta^{i'}, \vec{a}a\rangle$ and $\langle \delta^s, \vec{a}\rangle \rightarrow_a \langle \delta^{s'}, \vec{a}a\rangle$.

Since both $\langle \mathcal{D}, \delta^s\rangle$ and $\langle \mathcal{D}, \delta^i\rangle$ are online SD, they both evolve to a unique configuration. The new configuration of $\langle \mathcal{D}, \delta^i \&^{onl}_{Au} \delta^s\rangle$ (i.e. c') is obtained from these two, so it must be unique if it exists.

Claim 2. We have to show that for all $\vec{a}$ and a^u, $\vec{a} \in \mathcal{GR}(\langle \mathcal{D}, \delta^i \ \&^{onl}_{A_u} \ \delta^s\rangle)$ 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 $\vec{a}a^u \in \mathcal{GR}(\langle \mathcal{D}, \delta^i \ \&^{onl}_{A_u} \ \delta^s\rangle)$.

Since, wlog we assume that $\langle \mathcal{D}, \delta^i\rangle$ and $\langle \mathcal{D}, \delta^s\rangle$ started with a common controllable action, we can write $\vec{a} = \vec{a'}a^c\vec{a^u}$, where $\mathcal{D} \cup \{Executable(do(\vec{a'}, S_0))\} \models \neg A_u(a^c, do(\vec{a'}, S_0))$ and $D \cup \{Executable(do(\vec{a'}a^c, S_0)), A_u(\vec{a^u}, do(\vec{a'}a^c, S_0))\}$ is satisfiable. Let $\langle \delta^{i'}, \vec{a'}\rangle$ and $\langle \delta^{s'}, \vec{a'}\rangle$ denote the configurations reached by $\langle \delta^i, \epsilon\rangle$ and $\langle \delta^s, \epsilon\rangle$ after performing $\vec{a'}$ respectively; in other words: $\langle \delta^i, \epsilon\rangle \rightarrow^*_{\vec{a'}} \langle \delta^{i'}, \vec{a'}\rangle$ and $\langle \delta^s, \epsilon\rangle \rightarrow^*_{\vec{a'}} \langle \delta^{s'}, \vec{a'}\rangle$ By the fact that $\vec{a'}a^c\vec{a^u} \in \mathcal{GR}(\langle \mathcal{D}, \delta^i \ \&^{onl}_{A_u} \ \delta^s\rangle)$, we know that there is a configuration such that $\langle \delta^{i'} \ \&^{onl}_{A_u} \ \delta^{s'}, \vec{a'}\rangle \rightarrow_{a^c} \langle \delta^{i''} \ \&^{onl}_{A_u} \ \delta^{s''}, \vec{a'}a^c\rangle$ But then by the definition of the online transition relation ($\rightarrow$) we have that for all $\vec{b^u}$ such that $D \cup \{Executable(do(\vec{a'}a^c, S_0)), A_u(\vec{b^u}, do(\vec{a'}a^c, S_0))\}$ is satisfiable, if $\vec{a'}a^c\vec{b^u} \in \mathcal{GR}(\langle \mathcal{D}, \delta^i\rangle)$ then $\vec{a'}a^c\vec{b^u} \in \mathcal{GR}(\langle \mathcal{D}, \delta^s\rangle)$. In particular this holds for $\vec{b^u} = \vec{a^u}a^u$. Hence we have that if $\vec{a}a^u \in \mathcal{GR}(\sigma)$ then $\vec{a}a^u \in \mathcal{GR}(\langle \mathcal{D}, \delta^i \ \&^{onl}_{A_u} \ \delta^s\rangle)$. □

Proof of Theorem 4

Proof. We start by showing: $\mathcal{CR}(\langle \mathcal{D}, \delta^i \ \&^{onl}_{A_u} \ \delta^s\rangle) \subseteq \mathcal{CR}(\langle \mathcal{D}, \delta^i \ \& \ mps_{onl}(\delta^s, \sigma)\rangle)$. By Theorem 3 claim 2 we have that $\langle \mathcal{D}, \delta^i \ \&^{onl}_{A_u} \ \delta^s\rangle$ is online controllable for $\langle \mathcal{D}, \delta^i\rangle$. Considering that $\langle \mathcal{D}, \delta^i \ \& \ mps_{onl}(\delta^s, \sigma)\rangle$ is the largest online controllable supervisor for $\langle \mathcal{D}, \delta^i\rangle$, and that $\mathcal{RR}(\langle \mathcal{D}, \delta^i \ \& \ (\delta^i \ \&^{onl}_{A_u} \ \delta^s)\rangle) = \mathcal{RR}(\langle \mathcal{D}, \delta^i \ \&^{onl}_{A_u} \ \delta^s\rangle)$, we get the thesis.

Next we prove: $\mathcal{CR}(\langle \mathcal{D}, \delta^i \ \& \ mps_{onl}(\delta^s, \sigma)\rangle) \subseteq \mathcal{CR}(\langle \mathcal{D}, \delta^i \ \&^{onl}_{A_u} \ \delta^s\rangle)$. Suppose not. Then there exist a complete run $\vec{a}$ such that $\vec{a} \in \mathcal{CR}(\langle \mathcal{D}, \delta^i \ \& \ mps_{onl}(\delta^s, \sigma)\rangle)$ but $\vec{a} \notin \mathcal{CR}(\langle \mathcal{D}, \delta^i \ \&^{onl}_{A_u} \ \delta^s\rangle)$. As an aside, notice that if $\vec{a} \in \mathcal{CR}(\langle \mathcal{D}, \delta\rangle)$ then $\vec{a} \in \mathcal{GR}(\langle \mathcal{D}, \delta\rangle)$, and for all prefixes $\vec{a'}$ such that $\vec{a'}\vec{b} = \vec{a}$, we have $\vec{a'} \in \mathcal{GR}(\langle \mathcal{D}, \delta\rangle)$. Hence, let $\vec{a'} = \vec{a''}a$ such that $\vec{a''} \in \mathcal{GR}(\langle \mathcal{D}, \delta^i \ \&^{onl}_{A_u} \ \delta^s\rangle)$, $\vec{a''}a \in \mathcal{GR}(\langle \mathcal{D}, \delta^i \ \& \ mps_{onl}(\delta^s, \sigma)\rangle)$, but $\vec{a''}a \notin \mathcal{GR}(\langle \mathcal{D}, \delta^i \ \&^{onl}_{A_u} \ \delta^s\rangle)$, and let $\langle \delta^i, \epsilon\rangle \rightarrow^*_{\vec{a''}} \langle \delta^{i''}, \vec{a''}\rangle$ and $\langle \delta^s, \epsilon\rangle \rightarrow^*_{\vec{a''}} \langle \delta^{s''}, \vec{a''}\rangle$. Since $\vec{a''}a \notin \mathcal{GR}(\langle \mathcal{D}, \delta^i \ \&^{onl}_{A_u} \ \delta^s\rangle)$, it must be the case that there is no configuration c such that $\langle \delta^{i''} \ \&^{onl}_{A_u} \ \delta^{s''}, \vec{a''}\rangle \rightarrow_a c$. Since,

$\vec{a''}a \in \mathcal{GR}(\langle \mathcal{D}, \delta^i \ \& \ mps_{onl}(\delta^s, \sigma)\rangle)$, it follows that both $\langle \delta^{i''}, \vec{a''}\rangle \rightarrow_a \langle \delta^{i'''}, \vec{a''}a\rangle$ and $\langle \delta^{s''}, \vec{a''}\rangle \rightarrow_a \langle \delta^{s'''}, \vec{a''}a\rangle$. But then it must be the case that $\mathcal{D} \cup \{Executable(do(\vec{a''}, S_0))\} \models \neg A_u(a, do(\vec{a''}, S_0))$, and there exists $\vec{b^u}$ such that $D \cup \{Executable(do(\vec{a''}a, S_0)), A_u(\vec{b_u}, do(\vec{a''}a, S_0))\}$ is satisfiable and $\vec{a''}a\vec{b^u} \in \mathcal{GR}(\langle \mathcal{D}, \delta^i\rangle)$ but $\vec{a''}a\vec{b^u} \notin \mathcal{GR}(\langle \mathcal{D}, \delta^s\rangle)$.

Notice that $\vec{b^u} \neq \epsilon$, since we have that $\vec{a''}a \in \mathcal{GR}(\langle \mathcal{D}, \delta^s\rangle)$. So $\vec{b^u} = \vec{c^u}b^u\vec{d^u}$ with $\vec{a''}a\vec{c^u} \in \mathcal{GR}(\langle \mathcal{D}, \delta^s\rangle)$ but $\vec{a''}a\vec{c^u}b^u \notin \mathcal{GR}(\langle \mathcal{D}, \delta^s\rangle)$. Now $\vec{a'} \in \mathcal{GR}(\langle \mathcal{D}, \delta^i \ \& \ mps_{onl}(\delta^s, \sigma)\rangle)$ and $D \cup \{Executable(do(\vec{a''}a, S_0)), A_u(\vec{c^u}b^u, do(\vec{a''}a, S_0))\}$ is satisfiable, we have that $\vec{a'}\vec{c^u}b^u \in \mathcal{GR}(\langle \mathcal{D}, \delta^i \ \& \ mps_{onl}(\delta^s, \sigma)\rangle)$; this holds since, $mps_{onl}(\delta^s, \sigma)$ is controllable for σ, and we have that, if $\vec{a'}\vec{c^u}b^u \in \mathcal{GR}(\langle \mathcal{D}, \delta^i\rangle)$ then $\vec{a'}\vec{c^u}b^u \in \mathcal{GR}(\langle \mathcal{D}, mps_{onl}(\delta^s, \sigma)\rangle)$. This, by the definition of $mps_{onl}(\delta^s, \sigma)$, implies $\vec{a'}\vec{c^u}b^u \in \mathcal{GR}(\langle \mathcal{D}, \delta^i \ \& \ \delta^s\rangle)$. Hence, we can conclude that $\vec{a'}\vec{c^u}b^u \in \mathcal{GR}(\langle \mathcal{D}, \delta^s\rangle)$, getting a contradiction. □

Proof of Theorem 5

If $\langle \mathcal{D}, \delta^s\rangle$ and $\langle \mathcal{D}, \delta^i\rangle$ are online SD, than so is $\langle \mathcal{D}, \Sigma^w_{onl}(\delta^s, \delta^i)\rangle$.

Proof. By induction on the length of the action (and online transition) sequence. It can be shown in a similar way to Theorem 3 claim 1. □

Proof of Theorem 6

Proof. Claim 1. ($\subseteq$) Suppose that $\vec{a} \in \mathcal{CR}(\langle \mathcal{D}, \Sigma^w_{ol}(\delta^s, \delta^i)\rangle)$. By the definition of online transition for Σ^w_{ol}, it is easy to show that $\langle \delta^s, \epsilon\rangle \rightarrow^*_{\vec{a}} \langle \delta^{s'}, \vec{a}\rangle$ for some $\delta^{s'}$. By the definition of $Final$ for Σ^w_{ol}, we must have that $Final(\langle \delta^{s'}, \vec{a}\rangle)$. Thus $\vec{a} \in \mathcal{CR}(\langle \mathcal{D}, \delta^s\rangle)$.

($\supseteq$) Suppose that $\vec{a} \in \mathcal{CR}(\langle \mathcal{D}, \delta^s\rangle)$. Since $\mathcal{CR}(\langle \mathcal{D}, \delta^s\rangle) \subseteq \mathcal{CR}(\langle \mathcal{D}, \delta^i\rangle)$, we also have that $\vec{a} \in \mathcal{CR}(\langle \mathcal{D}, \delta^i\rangle)$. Clearly, every prefix of $\vec{a}$ is in $\mathcal{GR}(\langle \mathcal{D}, \delta^s\rangle)$ and $\mathcal{GR}(\langle \mathcal{D}, \delta^i\rangle)$. By the definition of online transition for Σ^w_{onl}, it is easy to show that $\langle \Sigma^w_{onl}(\delta^s, \delta^i), \epsilon\rangle \rightarrow^*_{\vec{a}} \langle \Sigma^w_{onl}(\delta^{s'}, \delta^{i'}), \vec{a}\rangle$ for some $\delta^{s'}$ and $\delta^{i'}$. By the definition of $Final$ for Σ^w_{onl}, we must have that $Final(\langle \Sigma^w_{onl}(\delta^{s'}, \delta^{i'}), \vec{a}\rangle)$. Thus $\vec{a} \in \mathcal{CR}(\langle \mathcal{D}, \Sigma^w_{onl}(\delta^s, \delta^i)\rangle)$.

Claim 2. ($\subseteq$) By contradiction. Suppose that $\vec{a} \in \mathcal{RR}(\langle \mathcal{D}, \Sigma^w_{onl}(\delta^s, \delta^i)\rangle)$ but $\vec{a} \notin \mathcal{GR}(\langle \mathcal{D}, \delta^s\rangle)$. Then there exists $\vec{b}$, a, and $\vec{c}$ such that $\vec{a} = \vec{b}a\vec{c}$ and $\vec{b} \in \mathcal{GR}(\langle \mathcal{D}, \delta^s\rangle)$ and $\vec{b}a \notin \mathcal{GR}(\langle \mathcal{D}, \delta^s\rangle)$ (note that since $\mathcal{CR}(\langle \mathcal{D}, \delta^s\rangle) \neq \emptyset$, we have that $\epsilon \in \mathcal{GR}(\langle \mathcal{D}, \delta^s\rangle)$). If a is not an exogenous action, then by the definition of online transition for Σ^w_{onl}, $\vec{b}a \notin \mathcal{RR}(\langle \mathcal{D}, \Sigma^w_{onl}(\delta^s, \delta^i)\rangle)$, and thus $\vec{a} \notin \mathcal{RR}(\langle \mathcal{D}, \Sigma^w_{onl}(\delta^s, \delta^i)\rangle)$, contradiction. Suppose that a is an exogenous action. Since δ^s is controllable wrt $\langle \mathcal{D}, \delta^i\rangle$, if $\vec{b}a \in \mathcal{GR}(\langle \mathcal{D}, \delta^i\rangle)$, then $\vec{b}a \in \mathcal{GR}(\langle \mathcal{D}, \delta^s\rangle)$, contradiction.

($\supseteq$) Suppose that $\vec{a} \in \mathcal{GR}(\langle \mathcal{D}, \delta^s\rangle)$. Then there exists $\vec{b}$ such that $\vec{a}\vec{b} \in \mathcal{CR}(\langle \mathcal{D}, \delta^s\rangle)$. Since $\mathcal{CR}(\langle \mathcal{D}, \delta^s\rangle) \subseteq \mathcal{CR}(\langle \mathcal{D}, \delta^i\rangle)$, we also have that $\vec{a}\vec{b} \in \mathcal{CR}(\langle \mathcal{D}, \delta^i\rangle)$. Clearly, every prefix of $\vec{a}\vec{b}$ is in $\mathcal{GR}(\langle \mathcal{D}, \delta^s\rangle)$ and $\mathcal{GR}(\langle \mathcal{D}, \delta^i\rangle)$. Thus by the definition of online transition for Σ^w_{onl}, it is easy to show that $\vec{a} \in \mathcal{RR}(\langle \mathcal{D}, \Sigma^w_{onl}(\delta^s, \delta^i)\rangle)$.

Claim 3. ($\subseteq$) Suppose that $\vec{a} \in \mathcal{RR}(\langle \mathcal{D}, \Sigma^w_{onl}(\delta^s, \delta^i)\rangle)$. By Claim 2, $\vec{a} \in \mathcal{GR}(\langle \mathcal{D}, \delta^s\rangle)$. Then by the definition of $\mathcal{GR}$, there exists $\vec{b}$ such that $\vec{a}\vec{b} \in \mathcal{CR}(\langle \mathcal{D}, \delta^s\rangle)$. By Claim 1, it follows that $\vec{a}\vec{b} \in \mathcal{CR}(\langle \mathcal{D}, \Sigma^w_{onl}(\delta^s, \delta^i)\rangle)$. Thus $\vec{a} \in \mathcal{GR}(\langle \mathcal{D}, \Sigma^w_{onl}(\delta^s, \delta^i)\rangle)$.

($\supseteq$) Follows trivially from the definitions of $\mathcal{RR}$ and $\mathcal{GR}$. □

Proof of Theorem 7

Proof. By Theorem 3 Claim 2 we have that $\langle \mathcal{D}, \delta^i \ \&^{onl}_{A_u} \ \delta^s\rangle$ is online controllable for $\langle \mathcal{D}, \delta^i\rangle$. By Theorem 4, $\mathcal{CR}(\langle \mathcal{D}, \delta^i \ \&^{onl}_{A_u} \ \delta^s\rangle) \subseteq \mathcal{CR}(\langle \mathcal{D}, \delta^i \ \& \ mps_{onl}(\delta^s, \sigma)\rangle)$. Thus by the definition of &, it is easy to show that $\mathcal{CR}(\langle \mathcal{D}, \delta^i \ \&^{onl}_{A_u} \ \delta^s\rangle) \subseteq \mathcal{CR}(\langle \mathcal{D}, \delta^i\rangle)$. Therefore by Theorem 6 Claim 2, we have that $\mathcal{RR}(\langle \mathcal{D}, \Sigma^w_{onl}(\delta^i \ \&^{onl}_{A_u} \ \delta^s, \delta^i)\rangle) = \mathcal{GR}(\langle \mathcal{D}, \delta^i \ \&^{onl}_{A_u} \ \delta^s\rangle)$.

Theorem 4 says that $\mathcal{CR}(\langle \mathcal{D}, \delta^i \ \&^{onl}_{A_u} \ \delta^s\rangle) = \mathcal{CR}(\langle \mathcal{D}, \delta^i \ \& \ mps_{onl}(\delta^s, \sigma)\rangle)$, and since a set of complete runs has a unique set of prefixes, it follows that $\mathcal{GR}(\langle \mathcal{D}, \delta^i \ \&^{onl}_{A_u} \ \delta^s\rangle) = \mathcal{GR}(\langle \mathcal{D}, \delta^i \ \& \ mps_{onl}(\delta^s, \sigma)\rangle)$. Thus $\mathcal{RR}(\langle \mathcal{D}, \Sigma^w_{onl}(\delta^i \ \&^{onl}_{A_u} \ \delta^s)\rangle) = \mathcal{GR}(\langle \mathcal{D}, \delta^i \ \& \ mps_{onl}(\delta^s, \sigma)\rangle)$.

It remains to show that $\mathcal{GR}(\langle \mathcal{D}, \delta^i \ \& \ mps_{onl}(\delta^s, \sigma)\rangle) = \mathcal{RR}(\langle \mathcal{D}, \delta^i \ \& \ mps_{onl}(\delta^s, \sigma)\rangle)$. By the definition of mps_{onl}, $\mathcal{CR}(\langle \mathcal{D}, mps_{onl}(\delta^s, \sigma)\rangle) \subseteq \mathcal{CR}(\langle \mathcal{D}, \delta^i \ \& \ \delta^s\rangle)$. Thus by the definition of &, it is easy to show that $\mathcal{CR}(\langle \mathcal{D}, mps_{onl}(\delta^s, \sigma)\rangle) \subseteq \mathcal{CR}(\langle \mathcal{D}, \delta^i\rangle)$. Then by the definition of $\mathcal{GR}$ and $\mathcal{CR}$, it follows that $\mathcal{GR}(\langle \mathcal{D}, mps_{onl}(\delta^s, \sigma)\rangle) \subseteq \mathcal{GR}(\langle \mathcal{D}, \delta^i\rangle)$. Thus by the definition of &, it is easy to show that $\mathcal{GR}(\langle \mathcal{D}, \delta^i \ \& \ mps_{onl}(\delta^s, \sigma)\rangle) = \mathcal{GR}(\langle \mathcal{D}, mps_{onl}(\delta^s, \sigma)\rangle)$. By Theorem 2, we have that $\mathcal{GR}(\langle \mathcal{D}, mps_{onl}(\delta^s, \sigma)\rangle) = \mathcal{RR}(\langle \mathcal{D}, mps_{onl}(\delta^s, \sigma)\rangle)$. Since $\mathcal{GR}(\langle \mathcal{D}, mps_{onl}(\delta^s, \sigma)\rangle) \subseteq \mathcal{GR}(\langle \mathcal{D}, \delta^i\rangle)$, it follows that $\mathcal{RR}(\langle \mathcal{D}, mps_{onl}(\delta^s, \sigma)\rangle) \subseteq \mathcal{GR}(\langle \mathcal{D}, \delta^i\rangle)$. Therefore by the definition of &, it is easy to show that $\mathcal{RR}(\langle \mathcal{D}, \delta^i \ \& \ mps_{onl}(\delta^s, \sigma)\rangle) = \mathcal{RR}(\langle \mathcal{D}, mps_{onl}(\delta^s, \sigma)\rangle)$. Thus, $\mathcal{RR}(\langle \mathcal{D}, \delta^i \ \& \ mps_{onl}(\delta^s, \sigma)\rangle) = \mathcal{GR}(\langle \mathcal{D}, \delta^i \ \& \ mps_{onl}(\delta^s, \sigma)\rangle)$. □