Online Agent Supervision in the Situation Calculus

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. 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. 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 applications 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 *et al.*, 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 *et al.*, 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.

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) 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 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

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 $do(a_n, do(a_{n-1}, \ldots, do(a_2, do(a_1, s)) \ldots))$; for an action sequence $\vec{a}$, we often write $do(\vec{a}, s)$ for $do([\vec{a}], s)$.

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 *et al.*, 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 *et al.*, 2010]; differently from [De Giacomo *et al.*, 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 *et al.*, 2012] if for every sequence of transitions, the remaining program is determined by the resulting situation, i.e.,

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

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 Agents Executing Online

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 *et al.*, 2008], we model sensing as an ordinary action which queries a sensor, followed by the reporting of a sensor result, in the form of an exogenous action.

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, denoting 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 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 defined as:

$\langle \delta, \vec{a} \rangle \rightarrow_{A(\vec{n})} \langle \delta', \vec{a}A(\vec{n}) \rangle$
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 action $\vec{a}$, if $c^i \rightarrow^*_{\vec{a}} c'$ and $c^i \rightarrow^*_{\vec{a}} c''$ then $c' = c''$. In [Banihashemi *et al.*, 2016b], it is shown that for an agent to be online SD, it is sufficient that the agent always knows what the remaining program is after an exogenous action. 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: $\mathcal{GR}(\sigma) = \{\vec{a} \mid \exists c, c', \vec{a'}. c^i \rightarrow^*_{\vec{a}} c \wedge c \rightarrow^*_{\vec{a'}} c' \wedge c'^{\checkmark}\}$.

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 represent the agent's possible behaviors 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:

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

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

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

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 $Do(\delta^i \mathbin{\&} \delta^s, s, do(\vec{a}, s))$) that are controllable wrt 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:

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

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

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

$$\begin{array}{l} \delta_4^i = [P?; ((A; (C \mid U)) \mid (B; D))] \quad | \\ \qquad [\neg P?; ((A; D) \mid (B; (C \mid U)))] \end{array}$$

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

$$\delta_4^s = (\pi a. a \neq U?; a)^*$$

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:

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

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. $\square$

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 *et al.*, 2004].

Online Maximally Permissive Supervisor. In our account of supervision, we consider 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. To 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. Also suppose that we have a *supervision specification* δ^s of what behaviors we want to allow in the supervised system and 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:

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

i.e, 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 $\langle \mathcal{D}, \delta^s \rangle$ ($\vec{a} a_u \in \mathcal{GR}(\sigma)$ and $\vec{a} a_u \in \mathcal{GR}(\langle \mathcal{D}, \delta^s \rangle)$ together imply that $\vec{a} a_u \in \mathcal{GR}(\langle \mathcal{D}, \delta^i \mathbin{\&} \delta^s \rangle)$). This definition, differently from DLM's, applies to online 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. As 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:

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

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 wrt the agent σ. We can show that:

Theorem 1 *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:*

1. *$mps_{onl}(\delta^s, \sigma)$ always exists and is unique;*

[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.

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. $mps_{onl}(\delta^s_4, \langle \mathcal{D}, \delta^i_4\rangle) = \mathbf{set}(\{\epsilon\})$. □

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 Online Supervision Operator

We 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*. Wlog, 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). We define $\delta^i \&^{onl}_{A_u} \delta^s$ by extending the online transition relation as follows:

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

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 2

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 3

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

While $\delta^i \&^{onl}_{A_u} \delta^s$ and $mps_{onl}(\delta^s, \sigma)$ have the same complete runs, they differ in their set of partial runs. 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 contrasts $mps_{onl}(\delta^s, \sigma)$, which is guaranteed to be non-blocking (Theorem 1).

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 $\langle \delta^i_6 \&^{oln}_{A_u} \delta^s_6, \epsilon\rangle \rightarrow_B \langle \delta'_6 \&^{onl}_{A_u} \delta^s_6, B\rangle$ 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. □

6 Search Over a Controllable Process

When we have a specification/process δ^s that is (online) controllable wrt an agent $\langle \mathcal{D}, \delta^i \rangle$ (e.g. $\delta^i \,\&^{onl}_{A_u}\, \delta^s$), 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 define a search construct[2] that 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: [3]

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

The final configurations are extended as follows:

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

It is easy to show that:

Theorem 4 *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 5 *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 6

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

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 $(\langle \Sigma^w_{onl}(nil \,\&^{onl}_{A_u}\, \delta^s_6, nil), A \rangle)^{\checkmark}$.

7 Discussion

A popular approach to automated service composition [McIlraith and Son, 2002; Sohrabi *et al.*, 2006] customizes a generic ConGolog process based on the user's constraints. [Sardiña and De Giacomo, 2009] on the other hand, synthesizes a controller that orchestrates the concurrent execution of library of 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 *et al.*, 2013b] 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. [Alechina *et al.*, 2015] regulates multiagent systems using regimented norms. A transition system describes the behavior of a (multi-) agent system and a guard function can enable/disable options that (could) violate norms after a system history (possibly using bounded lookahead). Finally, the approach in [Aucher, 2014] reformulates the results of supervisory control theory in terms of model checking problems in an epistemic temporal logic. While these approaches model behaviors as (nondeterministic) finite state transition systems, our approach enables users to express the system model and the specifications in a high-level expressive language. Moreover, due to its first-order logic foundations, it can handle infinite object domains and infinite states.

In this paper we have developed an account of supervision for agents that execute online and can acquire new knowledge as they operate. The framework uses a truely first-order representation of states and allows for an infinite object domain and infinite states. Proofs and examples of using online agent supervision to customize a travel planner agent are presented in [Banihashemi *et al.*, 2016a]. 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 *et al.*, 2013a; 2014]; verification of temporal properties over such agents is known to be decidable.

Acknowledgments

We acknowledge the support of Sapienza project "Immersive Cognitive Environments" and the National Sicience and Engineering Research Council of Canada.

[2]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 [De Giacomo and Levesque, 1999]. However, this search does not deal with sensing and online executions.

[3]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.

References

[Alechina *et al.*, 2015] Natasha Alechina, Nils Bulling, Mehdi Dastani, and Brian Logan. Practical run-time norm enforcement with bounded lookahead. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 443–451. ACM, 2015.

[Aucher, 2014] Guillaume Aucher. Supervisory control theory in epistemic temporal logic. In *International conference on Autonomous Agents and Multi-Agent Systems*. IFAAMAS/ACM, 2014.

[Banihashemi *et al.*, 2016a] Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. Online agent supervision in the situation calculus - Extended version. Technical Report EECS-2016-02, York University, 2016.

[Banihashemi *et al.*, 2016b] Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. Online situation-determined agents and their supervision. In *Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning*. AAAI, 2016.

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

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

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

[De Giacomo *et al.*, 2010] Giuseppe De Giacomo, Yves Lespérance, and Adrian R. Pearce. Situation calculus based programs for representing and reasoning about game structures. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference*. AAAI Press, 2010.

[De Giacomo *et al.*, 2012] Giuseppe De Giacomo, Yves Lespérance, and Christian J. Muise. On supervising agents in situation-determined ConGolog. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 1031–1038. IFAAMAS, 2012.

[De Giacomo *et al.*, 2013a] Giuseppe De Giacomo, Yves Lespérance, and Fabio Patrizi. Bounded epistemic situation calculus theories. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence*. IJCAI/AAAI, 2013.

[De Giacomo *et al.*, 2013b] Giuseppe De Giacomo, Fabio Patrizi, and Sebastian Sardina. Automatic behavior composition synthesis. *Artificial Intelligence*, 196:106–142, 2013.

[De Giacomo *et al.*, 2014] Giuseppe De Giacomo, Yves Lespérance, Fabio Patrizi, and Stavros Vassos. LTL verification of online executions with sensing in bounded situation calculus. In *ECAI 2014 - 21st European Conference on Artificial Intelligence*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 369–374. IOS Press, 2014.

[Fritz and McIlraith, 2006] Christian Fritz and Sheila A. McIlraith. Decision-theoretic Golog with qualitative preferences. In *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning*, pages 153–163. AAAI Press, 2006.

[Ghallab *et al.*, 2004] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann/Elsevier, San Francisco, CA, USA, 2004.

[Lespérance *et al.*, 2008] Yves Lespérance, Giuseppe De Giacomo, and Atalay Nafi Ozgovde. A model of contingent planning for agent programming languages. In *7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 477–484. IFAAMAS, 2008.

[Liaskos *et al.*, 2012] Sotirios Liaskos, Shakil M. Khan, Marin Litoiu, Marina Daoud Jungblut, Vyacheslav Rogozhkin, and John Mylopoulos. Behavioral adaptation of information systems through goal models. *Information Systems*, 37(8):767–783, 2012.

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

[McIlraith and Son, 2002] Sheila A. McIlraith and Tran Cao Son. Adapting Golog for composition of semantic web services. In *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning*, pages 482–496. Morgan Kaufmann, 2002.

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

[Sardiña and De Giacomo, 2009] Sebastian Sardiña and Giuseppe De Giacomo. Composition of ConGolog programs. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 904–910, 2009.

[Sardiña *et al.*, 2004] Sebastian Sardiña, Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. On the semantics of deliberation in Indigolog - from theory to implementation. *Ann. Math. Artif. Intell.*, 41(2-4):259–299, 2004.

[Sohrabi *et al.*, 2006] Shirin Sohrabi, Nataliya Prokoshyna, and Sheila A. McIlraith. Web service composition via generic procedures and customizing user preferences. In *Proceedings of the 5th International Semantic Web Conference (ISWC-06)*, volume 4273, pages 597–611. Springer, 2006.

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

[Wonham, 2014] WM Wonham. *Supervisory Control of Discrete-Event Systems*. University of Toronto, 2014 edition, 2014.

[Yadav *et al.*, 2013] Nitin Yadav, Paolo Felli, Giuseppe De Giacomo, and Sebastian Sardina. Supremal realizability of behaviors with uncontrollable exogenous events. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, pages 1176–1182. IJCAI/AAAI, 2013.