Supervisory Control for Behavior Composition

Paolo Felli
The University of Melbourne, Australia
paolo.felli@unimelb.edu.au

Nitin Yadav and Sebastian Sardina
RMIT University, Australia
{name.surname}@rmit.edu.au

Abstract

We relate behavior composition, a synthesis task studied in AI, to supervisory control theory from the discrete event systems field. In particular, we show that realizing (i.e., implementing) a target behavior module (e.g., a house surveillance system) by suitably coordinating a collection of available behaviors (e.g., automatic blinds, doors, lights, cameras, etc.) amounts to imposing a supervisor onto a special discrete event system. Such a link allows us to leverage on the solid foundations and extensive work on discrete event systems, including borrowing tools and ideas from that field. As evidence of that we show how simple it is to introduce preferences in the mapped framework.

1 Introduction

In this paper, we formally relate two automatic synthesis tasks, namely, *behavior composition*, as studied within the AI community (e.g., [5, 10, 26, 11]) and *supervisory control* in discrete event systems [28, 19, 20, 8]. By doing that, we aim at facilitating the awareness and cross-fertilization between the two different communities and techniques available.

The composition problem involves automatically "realizing" (i.e., implementing) a desired, though virtual, *target* behavior module by suitably coordinating the execution of a set of concrete *available* behavior modules. From an AI perspective, a behavior refers to the abstract operational model of a device or program, generally represented as a nondeterministic transition system. For instance, one may be interested in implementing a house entertainment system by making use of various devices installed, such as game/music consoles, TVs, lights, etc.

Supervisory Control, on the other hand, is the task of automatically synthesizing "supervisors" that *restrict* the behavior of a "plant", i.e. a discrete event system (DES) which is assumed to spontaneously generate events, such that a given specification is fulfilled. DES models a wide spectrum of physical systems, including manufacturing, traffic, logistics, and database systems. In Supervisory Control Theory (SCT), an automaton $\mathcal{G}$—known as *"the plant"*—is used to model both controllable and uncontrollable behaviors of a given DES. The assumption is that the overall behavior of $\mathcal{G}$ is *not* satisfactory and must be controlled. To that end, a so-called *supervisor* V is imposed on $\mathcal{G}$ so as to meet a given specification on event orderings and legality of states. Supervisors observe (some of) the events executed by $\mathcal{G}$ and can *disable* those that are controllable in order to guarantee a given specification.

Both behavior composition and supervisory control can be seen as *generalized* forms of automated planning tasks [25]. Rather than building (linear) plans to bring about an (achievement) goal, the aim is to keep the system in certain "good" states. Since we are to build controllers meant to run continuously, solutions generally include "loops." Moreover, in contrast with classical planning, the domains are nondeterministic in nature, which relates to FOND planning and strong-cyclic notions of plans, as shown in [21].

To build a bridge between the two problems and communities, this article provides the following technical contributions:

- A formal, provably correct (Theorems 4 and 5) reduction of the AI behavior composition problem to the problem of controlling a regular language on a particular DES plant (which we call "composition plant").
- A technique to extract the controller generator—the universal solution for a composition problem—from a supervisor of the DES composition plant. We show the technique is correct (Theorem 6), optimal w.r.t. computational complexity (in the worst case; Theorem 7), and realizable using existing off-the-shelf SCT tools.
- An approach to DES-based behavior composition *approximation* for the special case of deterministic system (as it is the case, for example, in web-service composition). This is appealing when no composition solution exists and one hence looks for "the best" possible controller.

The motivations behind linking behavior composition to supervisory control theory are threefold. First, supervisory control theory has rigorous *foundations rooted in formal languages*. It was first developed by [20] and others in the

80's and then further strengthened by many other researchers w.r.t. both theory and application (see, e.g., [8] for a broad overview of the field). Recasting the composition task as the supervision of a DES provides us with a solid foundation for studying composition. Second, computational properties for supervisor synthesis have been substantially studied and tools for supervisor synthesis are available, including TCT/STCT [31], GRAIL [22], DESUMA [23], and SUPREMICA [18]. Thus, we can apply very different techniques for solving the composition problem than those already available within the AI literature (e.g., PDL satisfiability [10], direct search [26], LTL/ATL synthesis [15, 12], and computation of special kind of simulation relations [11, 6]). Finally, once linked, we expect cross-fertilization between AI-style composition and DES supervisory theory. To that end, for instance, we demonstrate here how DES-based composition can directly and naturally handle constraints over a composition task. Indeed, one may look at importing powerful notions, and corresponding techniques, common in SCT, such as hierarchical and decentralized supervision, maximal controllability, and tolerance supervision.

2 Preliminaries

In this section, we very briefly review the required background to understand the rest of the paper.

2.1 The Behavior Composition Problem

The behavior composition problem has been recently much studied in the web-services and AI literature [4, 6, 11, 26, 15]. The problem amounts to synthesising a *controller* that is able to "realize" (i.e., implement) a desired, but nonexistent, *target behavior* module, by suitably coordinating a set of *available behavior* modules. In what follows, we mostly follow the model detailed in [11].[1]

Generally speaking, behaviors represent the operational logic of a device or program, and they are modeled using, possibly nondeterministic, finite transition systems. Nondeterminism is used to express the fact that one may have have incomplete information about a behavior's logic.

Example 1. Consider the example depicted in Figure 1. Target $\mathcal{T}$ encapsulates the desired functionality of a mining system, which allows to unboundedly extract minerals from the ground (DIG), move (some transportation vehicle) to the extraction area (GOMINE) and stock the loose materials to a certain deposit (action sequence LOAD–GODEPOT–UNLOAD). Finally, routine repairs are performed (REPAIR). At every step, the user requests an action compatible with this specification, and a (good) controller should guarantee that it can fulfill such request by delegating the action to one of the three machines actually available in the mine: a dumper truck (initially at the depot), a loader, and an old excavator (which are initially at the mine). Note that such machines may be nondeterministic: the truck can break down while trying to reach the mining area due to terrain conditions, whereas the loader might need to perform repairs after unloading. The excavator (which can not move from the extraction area) is instead deterministic; however, it is mainly intended for digging, not for loading. As a result, whenever it performs a LOAD action, it needs to be repaired before being able to load again. □

Technically, a *behavior* is a tuple $\mathcal{B} = \langle B, A, b_0, \delta \rangle$ where:

- B is the finite set of states;
- A is the set of actions;
- $b_0 \in B$ is the initial state; and
- $\delta \subseteq B \times A \times B$ is $\mathcal{B}$'s (nondeterministic) transition relation: $\langle b, a, b' \rangle \in \delta$ denotes that action a executed in behavior state b may lead the behavior to successor state b'.

We also use alternative notations for the transition relation, by freely exchanging the notations $\langle b, a, b' \rangle \in \delta$, $b \xrightarrow{a} b'$ in $\mathcal{B}$, and $b' \in \delta(b, a)$. If, for any state b and action a, there exists a unique successor state $b' \in \delta(b, a)$, then we say that $\mathcal{B}$ is *deterministic*, in the sense that its successor state is uniquely determined by the current state and the chosen action, thus we write $b' = \delta(b, a)$. A *trace* of $\mathcal{B}$ is the possibly infinite sequence $\tau = b_0 \xrightarrow{a_1} b_1 \xrightarrow{a_2} \cdots$ such that $b_{i+1} \in \delta(b_\ell, a_{\ell+1})$, for $\ell \geq 0$. A *history* is a finite trace.

An (available) *system* is a tuple $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n \rangle$, where each $\mathcal{B}_i = \langle B_i, A, b_{0i}, \delta_i \rangle$ is referred to as an *available behavior* in $\mathcal{S}$ (over shared actions A). The joint asynchronous execution of $\mathcal{S}$ is captured by the so-called *enacted system* $\mathcal{B}_\mathcal{S} = \langle B_\mathcal{S}, A, \vec{b}_0, \delta_\mathcal{S} \rangle$,[2] where:

- $B_\mathcal{S} = B_1 \times \ldots \times B_n$ is the set of system states of $\mathcal{B}_\mathcal{S}$ (given $\vec{b} = \langle b_1, \ldots, b_n \rangle \in B_\mathcal{S}$, we denote $st_i(\vec{b}) = b_i$ for all $i \in \{1, \ldots, n\}$);
- $\vec{b}_0 = \langle b_{01}, \ldots, b_{0n} \rangle \in B_\mathcal{S}$ is the initial state of $\mathcal{B}_\mathcal{S}$; and
- $\langle \vec{b}, a, j, \vec{b}' \rangle \in \delta_\mathcal{S}$ *iff* $st_j(\vec{b}) \xrightarrow{a} st_j(\vec{b}')$ in $\mathcal{B}_j$ and $st_i(b) = st_i(\vec{b}')$, for all $i \in \{1, \ldots, n\} \setminus \{j\}$.

A *system history* is a straightforward generalization of behavior histories to an available system $\mathcal{B}_\mathcal{S}$, that is, a sequence of the form $h = \vec{b}_0 \xrightarrow{a_1, j_1} \vec{b}_1 \xrightarrow{a_2, j_2} \cdots \xrightarrow{a_\ell, j_\ell} \vec{b}_\ell$. We denote with $last(h)$ the last state $\vec{b}_\ell$ of h and with $\mathcal{H}$ the set of all system histories. Finally, the *target behaviour* module is just a deterministic behavior $\mathcal{T} = \langle T, A_t, t_0, \delta_t \rangle$. For clarity, we denote

[1] For legibility, and without loss of generality, we leave out the so-called shared environment, used to model action's preconditions.

[2] The term "enacted" is due to the fact that, in the full composition setting, all behaviors are meant to be run within a shared environment (which, without loss of generality, we left out in this work for simplicity).

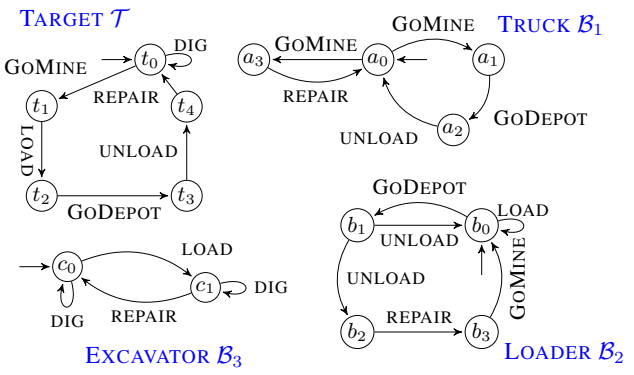

Figure 1: A mining system with three available machines.

its states by t instead of b. Hence, a *target trace* is a, possibly infinite, sequence $\tau = t_0 \xrightarrow{a_1} t_1 \xrightarrow{a_2} \cdots$, such that $t_i \xrightarrow{a_{i+1}} t_{i+1}$ in $\mathcal{T}$, for all $i \geq 0$.

A so-called *controller* is a function of the form

$$P : \mathcal{H} \times A_t \to \{1, \ldots, n\}$$

that takes a history (i.e., a run) of the system and the next action request, and outputs the index of the available behavior where the action is to be delegated. The composition task then amounts to whether there exists (and if so, how to compute it) a controller P such that the target behavior is "realized," that is, it looks as if the target module is being executed.

Roughly speaking, a controller realizes a target module if it is always able to further extend all the system traces (by prescribing adequate action delegations), no matter how the available behaviors happen to evolve (after each step). To capture this, one first define the set $\mathcal{H}_{P,\tau}$ of all (P,τ)-induced system histories, that is, those system histories (i.e., histories of enacted system $\mathcal{B}_\mathcal{S}$) with action requests as per target trace τ and action delegations as per controller P.[3]

Definition 1 ([10, 11]). Controller P *realizes a target trace* τ, as above, in a system $\mathcal{S}$ if for all (P,τ)-induced system histories $h \in \mathcal{H}_{P,\tau}$ with $|h| < |\tau|$, there exists an enacted system (successor) state $\vec{b}_{|h|+1}$ such that $last(h) \xrightarrow{a_{|h|+1}, j_h} \vec{b}_{|h|+1}$ is in $\mathcal{B}_\mathcal{S}$ with $j_h = P(h, a_{|h|+1})$. A controller P *realizes a target behavior* $\mathcal{T}$ (in a system $\mathcal{S}$) iff it realizes all the traces of $\mathcal{T}$. ■

The existence requirement of a system successor state $\vec{b}_{|h|+1}$ implies that delegation $P(h, a_{|h|+1})$ (of action request $a_{|h|+1}$ in system history h) is legal (i.e., is able to extend the current history h). Whenever a controller realizes a target behavior $\mathcal{T}$ in a system $\mathcal{S}$, we say that such controller is an *exact compositions* of $\mathcal{T}$ in $\mathcal{S}$. It is not difficult to see that there is indeed an exact composition for the example in Figure 1: all actions requested as per the target logic will *always* be fulfilled (i.e., delegated to an available behavior) by the controller, forever.

As one may expect, checking the existence of an exact composition is EXPTIME-complete [10], as it resembles conditional planning under full observability [24]. Interestingly, by revisiting a certain stream of work in service composition area, the technique devised in De Giacomo et al. [11] allows to synthesize a sort of meta-controller, called *controller generator* (CG) representing *all* possible compositions. Concretely, a CG is a function

$$cg : B_\mathcal{S} \times A_t \to 2^{\{1,\ldots,n\}}$$

that, given a system state and a target action a, returns a *set* of behavior indexes to which the requested action a may be legally delegated. A controller generator cg *generates* a concrete controller P iff $P(h,a) \in cg(last(h), a)$ for any system history h and action a compatible with P and the target logic, respectively. The CG is unique and finite, and represents a flexible and robust solution concept to the composition problem [11].[4]

We close by noting that De Giacomo et al. [11]'s technique is directly based on the idea that a composition amounts to a module that coordinates the concurrent execution of the available behaviours so as to "mimic" the desired target behaviour. This "mimicking" is captured through the formal notion of simulation [16], suitably adapted to deal with nondeterministic behaviors. Intuitively, a behavior $\mathcal{B}_1$ "simulates" another behavior $\mathcal{B}_2$, denoted $\mathcal{B}_2 \preceq \mathcal{B}_1$, if $\mathcal{B}_1$ is able to always *match* all of $\mathcal{B}_2$'s moves.

Formally, given two behaviors $\mathcal{B}_1 = \langle B_1, A, b_{01}, \delta_1 \rangle$ and $\mathcal{B}_2 = \langle B_2, A, b_{02}, \delta_2 \rangle$, a simulation relation of $\mathcal{B}_1$ by $\mathcal{B}_2$ is a relation $R \subseteq B_1 \times B_2$ such that $\langle b_1, b_2 \rangle \in R$ implies that for any action $a \in A$ and transition $b_1 \xrightarrow{a} b_1'$ in $\mathcal{B}_1$, there exists a transition $b_2 \xrightarrow{a} b_2'$ in $\mathcal{B}_2$.

Importantly, De Giacomo et al. [11] defined a so-called (greatest) ND-*simulation* relation (ND stands for nondeterministic) between (the states of) the target behavior $\mathcal{T}$ and (the states of) the enacted system $\mathcal{B}_\mathcal{S}$, denoted $\preceq_{\text{ND}}$.

Definition 2. Consider a target $\mathcal{T} = \langle T, A_t, t_0, \delta_t \rangle$ and the enacted system $\mathcal{B}_\mathcal{S} = \langle B_\mathcal{S}, A, \vec{b}_0, \delta_\mathcal{S} \rangle$. An ND-simulation relation of $\mathcal{T}$ by $\mathcal{B}_\mathcal{S}$ is a relation $R \subseteq T \times B_\mathcal{S}$ such that $\langle t, \vec{b} \rangle \in R$ implies that for all actions $a \in A$ there exists an index $j \in \{1, \ldots, n\}$ such that for all transitions $t \xrightarrow{a} t'$ in $\mathcal{T}$ *(i)* there exists a transition $\vec{b} \xrightarrow{a,j} \vec{b}'$ in $\mathcal{B}_\mathcal{S}$, and *(ii)* for all $\vec{b} \xrightarrow{a,j} \vec{b}'$ in $\mathcal{B}_\mathcal{S}$ we have $\langle t', \vec{b}' \rangle \in R$. ■

[3] The set of (P,τ)-induced system histories $\mathcal{H}_{P,\tau}$ is defined as $\mathcal{H}_{P,\tau} = \bigcup_k \mathcal{H}^k_{P,\tau}$, where $\mathcal{H}^0_{P,\tau} = \{\vec{b}_0\}$ (all behaviors are in their initial state) and $\mathcal{H}^{k+1}_{P,\tau}$ is the set of all possible histories of length k+1 obtained by applying P to a history in $\mathcal{H}^k_{P,\tau}$. See [11] for details.

[4] Note that there is a potentially uncountable set of composition controllers. To see this, consider any subset $E \subseteq \mathbb{N}$ of natural numbers and define controller C_E to delegate to behavior $\mathcal{B}_1$ if the length n of the current history is in E and to $\mathcal{B}_2$ otherwise. There is clearly one controller for each subset of $\mathbb{N}$, and thus there is an uncountable number of controllers.

The following result holds.

Theorem 1 ([6, 11]). *There exists a composition controller of a target behavior $\mathcal{T}$ in an available system $\mathcal{S}$ if and only if $t_0 \preceq_{ND} \vec{b}_0$ (where t_0 and $\vec{b}_0$ are $\mathcal{T}$'s and $\mathcal{B}_\mathcal{S}$'s initial states).*

In this paper we are indeed interested in synthesising controller generators, and not just single composition controllers. However, instead of building an ND-simulation relation, we aim at extracting the controller generator by leveraging on existing techniques in Supervisory Control Theory.

2.2 Supervisory Control in Discrete Event Systems

Discrete event systems range across a wide variety of physical systems that arise in technology (e.g., manufacturing and logistic systems, DBMSs, communication protocols and networks, etc.), whose processes are discrete (in time and state space), event-driven, and nondeterministic [8]. Generally speaking, Supervisory Control Theory is concerned with the *controllability* of the sequences (or strings/words) of events that such processes/systems—commonly referred as the *plant*—may generate [20].

As standard in formal languages, a *language* L over a set Σ is any set $L \subseteq \Sigma^*$, and $\epsilon \in \Sigma^*$ denotes the empty string. The *prefix-closure* of a language L, denoted by $\overline{L}$, is the language of all prefixes of words in L, that is, $w \in \overline{L}$ if and only if $w \cdot w' \in L$, for some $w' \in \Sigma^*$ ($w \cdot w'$ denotes the concatenation of words w and w'). A language L is *closed* if $L = \overline{L}$.

In SCT, the plant is viewed as a generator of the language of string of events characterizing its processes. Formally, a *generator* is a deterministic finite-state machine $\mathcal{G} = \langle \Sigma, G, g_0, \gamma, G_m \rangle$, where Σ is the finite alphabet of events; G is a finite set of states; $g_0 \in G$ is the initial state; $\gamma : G \times \Sigma \to G$ is the transition function; and $G_m \subseteq G$ is the set of marked states. We generalize transition function γ to words as follows: $\gamma : G \times \Sigma^* \to G$ is such that $\gamma(g, \epsilon) = g$ and $\gamma(g, w \cdot \sigma) = \gamma(\gamma(g, w), \sigma)$, with $w \in \Sigma^*$ and $\sigma \in \Sigma$. We say that a state $g \in G$ is *reachable* if $g = \gamma(g_0, w)$ for some word $w \in \Sigma^*$. Finally, given two words $w_1, w_2 \in \Sigma^*$, $w_1 > w_2$ iff $w_1 = w_2 \cdot w$, for some $w \neq \epsilon$.

The *language generated* by generator $\mathcal{G}$ is $L(\mathcal{G}) = \{w \in \Sigma^* \mid \gamma(g_0, w) \text{ is defined}\}$, whereas the *marked* language of $\mathcal{G}$ is $L_m(\mathcal{G}) = \{w \in L(\mathcal{G}) \mid \gamma(g_0, w) \in G_m\}$. Words in the former language stand for, possibly partial, operations or tasks, while words in the marked language represent the completion of some operations or tasks. Note that $L(\mathcal{G})$ is always closed, but $L_m(\mathcal{G})$ may not be.

Central to generators is the distinction between those events that are controllable and those that they are not. Technically, the generator's alphabet is partitioned into *controllable* (Σ_c) and *uncontrollable* (Σ_u) events, that is, $\Sigma = \Sigma_c \cup \Sigma_u$, where $\Sigma_c \cap \Sigma_u = \varnothing$. All events may occur only when *enabled*. Whereas controllable events may be enabled or disabled, uncontrollable events are assumed to be always enabled.

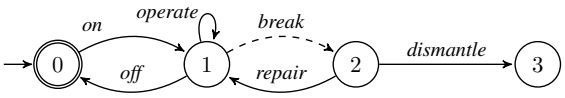

Figure 2: A generator modeling a simple machine.

Example 2. Figure 2 shows a generator $\mathcal{G}$ modeling a generic industrial machine. The machine can be started (event *on*), then repeatedly operated (event *operate*), finally stopped (event *off*). All these events are *controllable*, in that their occurrence is in the hand of the machine's user. While machine is in state 1, the machine may unexpectedly break down, signaled by the occurrence of event *break*. The occurrence of such event is however outside the control of the machine's user—the event is *uncontrollable*. When the machine breaks down, it ought to be either repaired or dismantled (events *repair* and *dismantle*, resp.), both within the control of the user.

As a generator, $\mathcal{G}$ produces words—those in $L(\mathcal{G})$—representing the possible runs (i.e., executions) of the machine being modelled. In particular, the machine's marked language $L_m(\mathcal{G})$ is equivalent to the regular expression $(on \cdot (operate \mid (break \cdot repair))^* \cdot off)^*$ that corresponds to those sequences of events that leave the machine in state 0. Words in $L_m(\mathcal{G})$ are said to be "marked," in that they are judged "complete," and therefore "good" (in the eye of the machine's designer). □

As expected, the overarching idea in SCT is to check whether one is able to guarantee certain specified (good) behavior of the device being modeled by a generator, and if so, how. A *specification* for a generator plant $\mathcal{G}$ is a language $K \subseteq L(\mathcal{G})$. We are now prepared to formally introduce the key notion of controllability in SCT.

Definition 3. A specification K is *controllable* in generator-plant $\mathcal{G}$ if and only if $\overline{K} \cdot \Sigma_u \cap L(\mathcal{G}) \subseteq \overline{K}$. ■

That is, every prefix of K immediately followed by a legal uncontrollable event (i.e., one compatible with $\mathcal{G}$) can be extended to a word in the specification itself. Intuitively, K is controllable if it is not possible to be "pushed" outside of it, regardless of potential uncontrollable events.

Example 3. Consider the specification $K_1 = L((on \cdot operate^* \cdot off)^*)$ requiring that the machine from Example 2 will always function without break downs. Clearly, such specification is *not* controllable: there exists an uncontrollable event (*break*) that can violate it. In fact, any word $w' = w \cdot on$ in $L(\mathcal{G})$ (therefore any such $w' \in \overline{K_1}$) can be extended with the uncontrollable event $break \in \Sigma_u$, resulting in word $w' \cdot break$ not meeting the specification, that is, $w' \cdot break \notin \overline{K_1}$.

Consider alternative specification $K_2 = L((on \mid off \mid operate \mid break \mid repair)^*)$, which prohibits that the machine be dismantled. Such specification is indeed controllable: every time the machine breaks down, one needs to repair it. Note how a very concrete process specification is embedded in K_1, whereas K_2 is more abstract, in that it does not describe a process but rather compactly captures a set of (good) processes. □

The next step is to define what it means for a generator to be "supervised" in order to achieve certain behavior. The idea is that one—the supervisor—can disable certain controllable events to achieve a desired behavior. Technically, a *supervisor* for a plant $\mathcal{G}$ is a function of the form

$$V : L(\mathcal{G}) \to \{\Sigma_e \mid \Sigma_e \in 2^{\Sigma},\ \Sigma_u \subseteq \Sigma_e\}$$

that outputs, for each word in $L(\mathcal{G})$, the set of events that are enabled (i.e., allowed) next. Notice that uncontrollable events are always enabled. A plant $\mathcal{G}$ under supervisor V yields the *controlled system* $V/\mathcal{G}$ whose generated and marked languages are defined as follows:

$$L(V/\mathcal{G}) = \{w \cdot \sigma \in L(\mathcal{G}) \mid\ w \in L(V/\mathcal{G}),\ \sigma \in V(w)\} \cup \{\epsilon\};$$
$$L_m(V/\mathcal{G}) = L(V/\mathcal{G}) \cap L_m(\mathcal{G}).$$

Informally, $L(V/\mathcal{G})$ represents all processes that plant $\mathcal{G}$ may yield while supervised by V, whereas $L_m(V/\mathcal{G})$ stands for the subset that are, in some sense, "complete".

A key result in SCT states that being able to control a (closed) specification in a plant amounts to finding a supervisor for such specification.

Theorem 2 ([27]). *Let $\mathcal{G}$ be a generator and $K \subseteq L(\mathcal{G})$ be a closed and non-empty specification. There exists a supervisor V such that $L(V/\mathcal{G}) = K$ iff K is controllable in $\mathcal{G}$.*

In many settings, one would further aim to control the language representing *complete* processes, that is, the marked fragment of the plant. In such cases, one shall focus on supervisors that can always drive the plant's execution towards the generation of words in the marked (supervised) language. Technically, supervisor V is *nonblocking* in plant $\mathcal{G}$ if $L(V/\mathcal{G}) = \overline{L_m(V/\mathcal{G})}$. This means that the strings in the supervised language $L(V/\mathcal{G})$ are prefixes of marked supervised language $\overline{L_m(V/\mathcal{G})}$, and therefore they can always be potentially extended into a complete marked string.

Example 4. Consider a specification K_3 for the generator in Figure 2 stating that one should *dismantle* the machine when it breaks after exactly n number of repairs. Concretely, $w \in K_3$ iff either w does not mention *dismantle* and mentions *repair* less than n times, or $w = w' \cdot break \cdot dismantle$ and w' mentions *repair* exactly n times but does not mention *dismantle*.

Specification $K_3 \cap L(\mathcal{G})$ is controllable, as there exists a supervisor V that disables event *dismantle* in any run/word containing less than n break-down events, and then enables it while disabling event *repair*. In particular, $L(V/\mathcal{G}) = K_3 \cap L(\mathcal{G})$ (see some words in K_3 may never arise in the plant $\mathcal{G}$).

Notice, however, that such supervisor V is *not* nonblocking: any string ending with the event *dismantle* can not be extended to a marked string. If, instead, state 3 were marked (or, say, the machine featured a controllable event *reassemble* from state 3 to state 2), then the same supervisor would be nonblocking, with $\overline{L_m(V/\mathcal{G})} = K_3 \cap L(\mathcal{G})$. □

Now, when a specification K is *not* (guaranteed to be) controllable, one then looks for controlling the "largest" (in terms of set inclusion) possible sublanguage of K. Interestingly, such sublanguage, called the *supremal controllable sublanguage* of K and denoted $sup\mathsf{C}(K)$, does exist and is in fact unique [28].

Putting it all together, in SCT, we are generally interested in (controlling) the K's sublanguage $K^{\uparrow} = sup\mathsf{C}(K \cap L_m(\mathcal{G}))$, that is, the supremal marked specification. It turns out that, under a plausible assumption, a supervisor does exist for non-empty $K^{\uparrow}$.

Theorem 3 ([27]). *If $\overline{K} \cap L_m(\mathcal{G}) \subseteq K$ and $K^{\uparrow} \neq \varnothing$, there exists a nonblocking supervisor V for $\mathcal{G}$ s.t. $L_m(V/\mathcal{G}) = K^{\uparrow}$.*

The assumption that $\overline{K} \cap L_m(\mathcal{G}) \subseteq K$ states that initial specification K is *closed under marked-prefixes*: every prefix from K representing a complete process is part of K. Theorem 3 will play a key role in our results.

Example 5. Consider again $K_1 = L((on \cdot operate^* \cdot off)^*)$ from Example 3. Its supremal marked specification is $K_1^{\uparrow} = \{\epsilon\}$, that is, the sole empty string. This is because as soon as the event *on* is enabled and the machine moves to 1, the uncontrollable event *break* may occur, thus violating K_1. Therefore, any word leading to state 1 can not be in $K_1^{\uparrow}$. □

3 DES-based Behavior Composition

In this section we show how to relate the notion of a composition controller in behavior composition to that of an adequate supervisor in discrete event system. After all, their operational requirements are similar, namely, to take decisions in a step-by-step fashion in order to keep the system evolutions in a restricted set of "good" traces. Their differences can be summarized as follows:

Composition Controller	Supervisor
given a system history h and a target action a, it outputs one delegation $P(h, a)$	given a plant's string prefix w, it outputs enabled events $V(w)$
such that it is possible to proceed forever	such that we can always 'reach' marked states

Hence, the idea is to mimic $j = P(h, a)$ by means of $j \in V(h \cdot a)$, with $a \in V(h)$. However, there are fundamental differences between the two formalisms that do not allow for a direct, straightforward, translation. As a matter of fact, a naive translation that defines the plant as the cross-product of all available behaviors and the target's language as specification will simply not work, due to several mismatches between DES and behavior composition (see Section 5 for details). In particular, it is well known that, for nondeterministic systems (as is the case with the available system), the notion of language inclusion is weaker than that of simulation.

From now on, let $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n \rangle$ be an available system, where $\mathcal{B}_i = \langle B_i, A, b_{0i}, \delta_i \rangle$ for $i \in \{1, \ldots, n\}$, and $\mathcal{T} = \langle T, A_t, t_0, \delta_t \rangle$ a target behavior (without loss of generality we assume $\mathcal{T}$ to be connected and all B_i's and $\mathcal{T}$ to be mutually disjoint sets). The general approach is to build an adequate plant from $\mathcal{S}$ and $\mathcal{T}$, and define a specification language K, such that controlling K (as per Definition 3) amounts to composing $\mathcal{T}$ in $\mathcal{S}$.

So, let us next build a generator $\mathcal{G}_{\langle \mathcal{S},\mathcal{T} \rangle}$—the *plant* to be controlled—from target $\mathcal{T}$ and system $\mathcal{S}$. The controllable aspect of the plant amounts to behavior delegations: at any point in time, a supervisor can enable or disable an available behavior to execute. On the other hand, the supervisor can control neither the action requests nor the evolution of the behavior selected—they are uncontrollable events. A state in the plant encodes a snapshot of the whole composition process, namely, the state of all behaviors (including the target) together with the current pending target request and current behavior delegation. Only those with no pending request or delegation are considered "marked." Below, we use two auxiliary sets $\mathsf{Indx} = \{1, \ldots, n\}$ and $\mathsf{Succ} = \bigcup_{i \in \{1,\ldots,n\}} B_i$.

Definition 4. Let the *composition plant* $\mathcal{G}_{\langle \mathcal{S},\mathcal{T} \rangle} = \langle \Sigma, G, g_0, \gamma, G_m \rangle$ be defined as follows:

- $\Sigma = \Sigma_c \cup \Sigma_u$, where $\Sigma_c = \mathsf{Indx}$ and $\Sigma_u = A_t \cup \mathsf{Succ}$, is the finite set of controllable (behaviors' indexes) and uncontrollable events (target's actions and behaviors' states).
- $G = T \times B_1 \times \ldots \times B_n \times (A_t \cup \{e\}) \times (\mathsf{Indx} \cup \{0\})$ is the finite set of states of the plant. Additional symbol e denotes no active request, whereas index 0 denotes no active delegation.
- $g_0 = \langle t_0, b_{01}, \ldots, b_{0n}, e, 0 \rangle$ is the initial state of the plant, encoding the initial configuration of the system and target, and the fact that there has been no request event or delegation.
- $\gamma : G \times \Sigma \to G$ of $\mathcal{G}_{\langle \mathcal{S},\mathcal{T} \rangle}$ is the plant's transition function where $\gamma(\langle t, b_1, \ldots, b_n, \mathsf{req}, \mathsf{idx} \rangle, \sigma)$ is equal to:
  - $\langle \delta_t(t, \sigma), b_1, \ldots, b_n, \sigma, 0 \rangle$ if $\mathsf{req} = e$, $\mathsf{idx} = 0$, $\sigma \in A_t$;
  - $\langle t, b_1, \ldots, b_n, \mathsf{req}, \sigma \rangle$ if $\mathsf{req} \in A_t$, $\mathsf{idx} = 0$, $\sigma \in \mathsf{Indx}$;
  - $\langle t, b_1, \ldots, b'_{\mathsf{idx}}, \ldots, b_n, e, 0 \rangle$ if $\sigma \in \delta_{\mathsf{idx}}(b_{\mathsf{idx}}, \mathsf{req})$, $b'_{\mathsf{idx}} = \sigma$.
- $G_m = T \times B_1 \times \ldots \times B_n \times \{e\} \times \{0\}$ (marked states). ■

By inspecting the plant transition function γ we can see that the whole process for one target request involves three transitions in the plant, namely, target action request, behavior delegation, and lastly available system evolution. Initially, and after each target request has been fulfilled, the plant is in a state with no active request (e) and no behavior delegation (0), ready to accept and process a new target request—a *marked* state. Then:

1. given a legal target request (uncontrollable event) $\sigma \in A_t$, the plant evolves to a state recording the request and the corresponding target evolution (case 1 of γ);
2. after that, the plant may evolve relative due to (controllable) delegation events (one per available behavior), to states recording such delegations as well as the current pending action (case 2 of γ); and finally
3. the plant may evolve, in an uncontrollable manner, to states reflecting all possible evolutions of the behavior selected, together with no active request or delegation (case 3 of γ).

Observe that a composition plant $\mathcal{G}_{\langle \mathcal{S},\mathcal{T} \rangle}$, being a generator, is *deterministic*, whereas the available behaviors being modelled may include nondeterministic evolutions. The fact is that such nondeterminism is encoded via uncontrollable events.

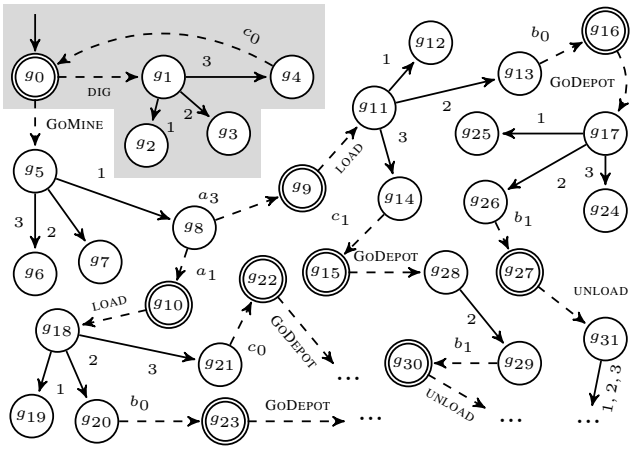

Figure 3: Plant $\mathcal{G}_{\langle \mathcal{S},\mathcal{T} \rangle}$ for the example in Figure 1 (partial). Double circled states are marked, so any word prefix ending in one of these states is marked. Dashed transitions correspond to uncontrollable events, solid ones to controllable events (delegations). State components are listed in the table.

g_0	$\langle t_0, a_0, b_0, c_0\rangle$	e	0	g_8	$\langle t_1, a_0, b_0, c_0\rangle$	GoMine	1	g_{16}	$\langle t_2, a_3, b_0, c_1\rangle$	e	0	g_{24}	$\langle t_3, a_1, b_0, c_0\rangle$	GoDepot	3
g_1	$\langle t_0, a_0, b_0, c_0\rangle$	DIG	0	g_9	$\langle t_1, a_3, b_0, c_0\rangle$	e	0	g_{17}	$\langle t_3, a_3, b_0, c_0\rangle$	GoDepot	0	g_{25}	$\langle t_3, a_3, b_0, c_0\rangle$	GoDepot	1
g_2	$\langle t_0, a_0, b_0, c_0\rangle$	DIG	1	g_{10}	$\langle t_1, a_1, b_0, c_0\rangle$	e	0	g_{18}	$\langle t_2, a_1, b_0, c_0\rangle$	LOAD	0	g_{26}	$\langle t_3, a_3, b_0, c_0\rangle$	GoDepot	2
g_3	$\langle t_0, a_0, b_0, c_0\rangle$	DIG	2	g_{11}	$\langle t_2, a_3, b_0, c_0\rangle$	LOAD	0	g_{19}	$\langle t_2, a_1, b_0, c_0\rangle$	LOAD	1	g_{27}	$\langle t_3, a_1, b_1, c_0\rangle$	e	0
g_4	$\langle t_0, a_0, b_0, c_0\rangle$	DIG	3	g_{12}	$\langle t_2, a_3, b_0, c_0\rangle$	LOAD	1	g_{20}	$\langle t_2, a_1, b_0, c_0\rangle$	LOAD	2	g_{28}	$\langle t_3, a_3, b_0, c_1\rangle$	GoDepot	0
g_5	$\langle t_1, a_0, b_0, c_0\rangle$	GoMine	0	g_{13}	$\langle t_2, a_3, b_0, c_0\rangle$	LOAD	2	g_{21}	$\langle t_2, a_1, b_0, c_0\rangle$	LOAD	3	g_{29}	$\langle t_3, a_3, b_0, c_1\rangle$	GoDepot	2
g_6	$\langle t_1, a_0, b_0, c_0\rangle$	GoMine	3	g_{14}	$\langle t_2, a_3, b_0, c_0\rangle$	LOAD	3	g_{22}	$\langle t_2, a_1, b_0, c_0\rangle$	e	0	g_{30}	$\langle t_3, a_3, b_0, c_1\rangle$	e	0
g_7	$\langle t_1, a_0, b_0, c_0\rangle$	GoMine	2	g_{15}	$\langle t_2, a_3, b_0, c_1\rangle$	e	0	g_{23}	$\langle t_2, a_1, b_0, c_0\rangle$	e	0	g_{31}	$\langle t_4, a_3, b_1, c_0\rangle$	UNLOAD	0

Hence, the final step in an action request delegation process may yield multiple plant states, one per nondeterministic evolution of the selected behavior. Also, such resulting states are to be considered "marked," in that a complete delegation process has been completed. If, however, the chosen behavior is unable to legally execute the active request from its current state, then no transition is defined and the plant (non-marked) state is a dead-end.

Example 6. Figure 3 depicts the (partial) plant for the composition problem of Figure 1. Each complete delegation process of action requests corresponds, in the plant, to three consecutive events in $(A_t \cdot \mathsf{Indx} \cdot \mathsf{Succ})$.

After each uncontrollable event representing a target request, three delegations—to available behaviors $\mathcal{B}_1, \mathcal{B}_2$ and $\mathcal{B}_3$—are *always* possible. For instance, the nodes in the greyed area represent the complete delegation of the digging action from the initial plant (and composition) state g_0. The event DIG represents the action request; that is uncontrollable, and hence always enabled. The resulting state g_1 registers such request. Then, three distinct controllable events embody the three possible delegations, one per available behavior. However, only behavior $\mathcal{B}_3$ can legally perform action DIG from its initial state (see Figure 1) to successor state c_0. Hence, a further uncontrollable event (c_0 itself) is used to model the looping transition evolution of $\mathcal{B}_3$. In general, there could be multiple uncontrollable evolutions if the delegated behavior behaves nondeterministically; see for example, plant state g_8 where behaviour $\mathcal{B}_1$ may evolve in two ways.

Of course, delegations reaching dead-end states are not desirable (e.g., delegation 1 in g_{18}). However, not reaching an immediate dead-end is not enough to capture the composition requirements. Indeed, whereas delegation to $\mathcal{B}_2$ and $\mathcal{B}_3$ will avoid immediate dead-ends in g_{18}, only the latter will be part of a composition solution (see later Figure 4, state 2). □

With the plant built, the question is what language one would like to control. The answer is simple: we aim to control exactly the marked language of the composition plant, that is,

$$K_{\langle \mathcal{S},\mathcal{T}\rangle} = L_m(\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}).$$

In other words, we seek for ways of always controlling the plant so as to eventually be able to reach the end of every request-delegation process. Observe that, contrary to intuition, the target behavior $\mathcal{T}$ is *not* used to derive the language specification, except in that it is embedded into the plant itself. This is not surprising, as the the target is one of the components generating uncontrollable events (the other being the evolution of available behaviors).

We shall claim that the ability to control $K_{\langle \mathcal{S},\mathcal{T}\rangle}$ in plant $\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}$ amounts to the ability to compose $\mathcal{T}$ in system $\mathcal{S}$. To that end, we first show an important technical result stating that set of (P,τ)-induced system histories $\mathcal{H}_{P,\tau}$ is in bijection with the set of traces in $K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$ when P is a composition controller and τ a trace of $\mathcal{T}$. This appears evident when carefully inspecting Figure 3, and is formalized in the following lemma. We use mapping $\mathtt{word}(h) \in (A_t \cdot \mathsf{Indx} \cdot \mathsf{Succ})^{|h|}$ to translate a system history (i.e., a finite trace of the enacted system $\mathcal{B}_{\mathcal{S}}$) into words generated by composition plant $\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}$.

Lemma 1. *Controller P is a composition for target $\mathcal{T}$ in system $\mathcal{S}$* iff *for each target trace τ and system history $h \in \mathcal{H}_{P,\tau}$ we have that $\mathtt{word}(h) \in K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$, where:*

$$\mathtt{word}(\vec{b}_0 \xrightarrow{a_1,j_1} \vec{b}_1 \xrightarrow{a_2,j_2} \ldots \xrightarrow{a_\ell,j_\ell} \vec{b}_\ell) = (a_1 \cdot j_1 \cdot st_{j_1}(\vec{b}_1)) \cdot \ldots \cdot (a_\ell \cdot j_\ell \cdot st_{j_\ell}(\vec{b}_\ell)).$$

Functions $st_i : G \to B_i$[5], with $i \in \mathsf{Indx}$, project the state of i-th behavior in a plant state, that is, $st_i(\langle t, \vec{b}, a, j\rangle) = st_i(\vec{b}) = b_i$. Analogously, for the target, $st_t(\langle t, \vec{b}, a, j\rangle) = t$.

Proof. ($\Rightarrow$) Assume by contradiction that there exists a composition P such that for some target trace τ and induced history $h \xrightarrow{a,j} \vec{b}$, we have $P(h,a) = j$ but $\mathtt{word}(h) \cdot a \cdot j \cdot b' \notin K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$, with $b' = st_j(\vec{b})$. This implies that $\mathtt{word}(h) \cdot a \cdot j \cdot b'$ is not allowed from the initial state g_0 of the plant, according to supervisor V such that $L_m(V/\mathcal{G}) = K^{\uparrow}$, i.e., either

(1) $\mathtt{word}(h) \cdot a \notin L(\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle})$ or

(2) $\mathtt{word}(h) \cdot a \cdot j \notin L(\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle})$ or

(3) $\mathtt{word}(h) \cdot a \cdot j \cdot b' \notin L(\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle})$ or

(4) for all words $w \in L_m(\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle})$ with $w > \mathtt{word}(h) \cdot a \cdot j \cdot b'$ we have $w \notin \overline{K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}}$.

Case (1) is not possible by construction of $\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}$. Indeed, according to γ, it is $w \cdot a \in L(\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle})$ for every

[5] We extend the function st_i to map a plant state to corresponding behavior state.

$w \in L_m(\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle})$ such that $\delta_t(st_t(\gamma(g_0,w)),a)$ is defined in $\mathcal{T}$. Case (3) implies, by definition of γ, that $b'_j \notin \delta_j(b_j,a)$, with $b_j = st_j(\gamma(g_0,w))$. Hence, the action a can not be replicated by behavior $\mathcal{B}_j$ and, as a consequence, the plant's state reached with a is a dead-end. This contradicts the fact that P is a composition for $\mathcal{T}$ in $\mathcal{S}$. Indeed, let $g = \gamma(g_0,w)$; note that this also implies that $st_t(g) \not\preceq_{\text{ND}} \langle st_1(g),\ldots,st_n(g)\rangle$, namely that the simulation is violated, as action a can not be replicated in the enacted system state $\langle st_1(g),\ldots,st_n(g)\rangle$. By following the same argument, we can also exclude case (2). Finally, case (4) implies that for any such word w we have $w \cdot \Sigma_u \cap L(\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}) \notin \overline{K_{\langle \mathcal{S},\mathcal{T}\rangle}}^{\uparrow}$, i.e., there exists a sequence of (uncontrollable) events leading to a state which is not coreachable, i.e., from where a marked state is not reachable. Indeed, remember that $K_{\langle \mathcal{S},\mathcal{T}\rangle} = L_m(\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle})$. Hence, since every action $a \in A_t \subset \Sigma_u$ is always allowed by any supervisor and, by construction of $\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}$, $w \cdot a \in L(\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle})$ for every $w \in L_m(\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle})$, we can apply the same reasoning of (3) and deduce that P is not a composition. That is, there exists a target trace $\tau' = \tau \xrightarrow{a} t_\ell$, with $h \in \mathcal{H}_{P,\tau}$ and $w = \texttt{word}(h)$, not realized by P.

($\Leftarrow$) First of all, since $\overline{K_{\langle \mathcal{S},\mathcal{T}\rangle}} \cap L(\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}) \subseteq K_{\langle \mathcal{S},\mathcal{T}\rangle}$ and by the previous assumption $K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow} \neq \emptyset$, then by Theorem 3 a supervisor V does exist. Hence, $\texttt{word}(h)\cdot a\cdot j\cdot b' \in K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$ iff there exists a supervisor V such that $L_m(V/\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}) = K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$, $a \in V(\texttt{word}(h))$, $j \in V(\texttt{word}(h)\cdot a)$ and $b' \in V(\texttt{word}(h)\cdot a\cdot j)$. Then, remember that $A_t \subset \Sigma_u$ and hence all target action are always allowed by V. Similarly, the event set Succ is uncontrollable as well. Assume by contradiction that $\texttt{word}(h)\cdot a\cdot j\cdot b' \in K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$ but it does not exist any composition P such that $P(h,a) = j$. By definition of composition, this implies that there exists a target trace $\tau = t_0 \xrightarrow{a_1} \cdots t_k$ with $a = a_{|\tau|}$ such that for some history $h \in \mathcal{H}_{P,\tau}$ we have that $\delta_{\mathcal{S}}(last(h),a,j)$ is not defined in the system behavior $\mathcal{B}_{\mathcal{S}}$ built out of $\mathcal{B}_1,\ldots,\mathcal{B}_n$. This means that either $\delta_t(st_t(last(h)),a)$ is not defined or behavior $\mathcal{B}_j$ can not perform this action from its current state $st_j(last(h))$, i.e., $b' \neq \delta_j(st_j(last(h)),a)$. Again, observe how this also implies that $st_t(last(h)) \not\preceq \langle st_1(last(h)),\ldots,st_n(last(h))\rangle$. In other words, according to γ, $\texttt{word}(h)\cdot a\cdot j\cdot b' \notin L(\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle})$. If this is the case, then either $a \notin V(\texttt{word}(h))$ or $j \notin V(\texttt{word}(h)\cdot a)$ or $b' \notin V(\texttt{word}(h)\cdot a\cdot j)$ and we get a contradiction. □

The above lemma is the key to prove our main results of this section, namely, that supervisors able to control the specification $K_{\langle \mathcal{S},\mathcal{T}\rangle}$ in plant $\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}$ correspond *one-to-one* with composition (solution) controllers for building target $\mathcal{T}$ in available system $\mathcal{S}$. To express such results, we first need to relate supervisors and controllers.

Definition 5. Let V be a supervisor for composition plant $\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}$. A controller $P_V : \mathcal{H}\times A \to \{1,\ldots,n\}$ is *induced by* V *iff* $P_V(h,\sigma) \in V(\texttt{word}(h)\cdot\sigma)$, for every $h \in \mathcal{H}$ and $\sigma \in A$. ■

In other words, a P_V is induced by a supervisor V iff its delegations fall into the set of "delegation events" allowed by V. Clearly, a supervisor can induce many controllers.

The main result of this section states that the supremal of the specification is controllable by some supervisor iff a solution to the composition problem exists. Moreover, every such supervisor induces controllers that are in fact compositions.

Theorem 4 (Soundness). *There exists a nonblocking supervisor V such that $L_m(V/\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}) = K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow} \neq \emptyset$* iff *there exists a composition P for $\mathcal{T}$ in $\mathcal{S}$. In particular, every controller P_V induced by V is a composition for $\mathcal{T}$ in $\mathcal{S}$.*

Proof. ($\Rightarrow$) Assume by contradiction that for some controller P_V there exists a target trace $\tau = t_0 \xrightarrow{a_1} \cdots \xrightarrow{a_\ell} t_\ell$ and an induced system history $h \in \mathcal{H}_{P_V,\tau}$ such that either (1) $P_V(h,a_\ell)$ is not defined or (2) $P_V(h,a_\ell) = j$ but $\delta_j(st_j(last(h)),a_\ell)$ is not defined. By Lemma 1, it means that $\texttt{word}(h)\cdot a_\ell\cdot j \notin \overline{K_{\langle \mathcal{S},\mathcal{T}\rangle}}^{\uparrow}$. More precisely, (1) implies that $\texttt{word}(h)\cdot a_\ell \notin L(\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle})$ whereas (2) implies that $a_\ell \notin V(\texttt{word}(h))$ and $j \notin V(\texttt{word}(h)\cdot a_\ell)$. Hence, either $K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow} = \emptyset$, or P_V does not realize the target trace τ and we contradict Lemma 1.

($\Leftarrow$) By Lemma 1, if such P exists then $K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow} \neq \emptyset$. □

Furthermore, (nonblocking) supervisors are "complete" in that they embed every possible composition controller.

Theorem 5 (Completeness). *Given a nonblocking supervisor V such that $L_m(V/\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}) = K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$, every composition P for $\mathcal{T}$ in $\mathcal{S}$ is induced by V.*

Proof. Assume by contradiction that there exists a composition P' which can not be induced by V, that is, it is such that $P'(h,a_\ell) \notin V(\texttt{word}(h)\cdot a_\ell)$ for some target trace $\tau = t_0 \xrightarrow{a_1} \cdots \xrightarrow{a_\ell} t_\ell$ and some induced system history $h \in \mathcal{H}_{P',\tau}$. More precisely, for some $j \in \mathsf{Indx}$, $\texttt{word}(h)\cdot a_\ell\cdot j \notin K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$ but $j \in P'(h,a_\ell)$. It is easy to see that, by Lemma 1, P' can not be a composition. □

Let us call supervisors of this sort *composition supervisors*. These two results demonstrate the formal link between the two synthesis tasks, namely, synthesis of a composition controller and supervisor synthesis.

Recalling that $K_{\langle \mathcal{S},\mathcal{T}\rangle} = L_m(\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle})$ and the definition of compositions, and as already hinted in the proof of Lemma 1, we can also explicitly relate the notions of maximal controllable sublanguage and ND-simulation.

Corollary 1. *Given a plant $\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}$ for $\mathcal{S}$ and $\mathcal{T}$ as above, if $K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow} \neq \emptyset$ then for any word $w \in K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$, we have that $st_t(g) \preceq_{ND} \langle st_1(g),\ldots,st_n(g)\rangle$ where $g = \gamma(g_0,w)$.*

Proof. We proceed by induction on the length of w. The claim holds for $g = g_0$ (by Theorem 4 and Thorem 1). Assume now it holds for w', with $g' = \gamma(g_0,w')$, and consider any word $w = w'\cdot a\cdot j\cdot b_j$ for some action a, delegation j and behavior state b_j. Since $a \in \Sigma_u$, and because $K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$ is controllable, from Definition 3 it follows

that $w \in K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$ implies $w \cdot a \in \overline{K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}}$, and for the same reason also $w{\cdot}a{\cdot}j{\cdot}b_j \in K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$ for some $j \in \mathsf{Indx}$ and $b_j \in B_j$. Which means that for any action a such that $st_t(g') \xrightarrow{a} t$ for some target state t, there exists an index j such that $\langle st_1(g), \ldots, st_n(g)\rangle \xrightarrow{a,j} \vec{b}'$ in $\mathcal{B}_{\mathcal{S}}$, with $st_j(\vec{b}) = b_j$. □

It remains to be seen, though, how to actually *extract* finite representations of composition controllers from DES composition supervisors.

3.1 From Supervisors to Controller Generators

As discussed at the end of Section 2.1, a controller generator (CG) is a finite structure encoding all possible composition solutions—a sort of a universal solution—-that, once computed, can be used at runtime to produce all possible target realizations. Because of that, CGs have been shown to enjoy run-time flexibility and robustness properties, in that the executor can leverage on them to recover or adapt to behavior various types of execution failures (e.g., an available behavior breaking down completely) [11]. Next, we show that it is possible to to extract *the* CG from composition supervisors.

We start by noting that, since both languages $L(\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle})$ and $K_{\langle \mathcal{S},\mathcal{T}\rangle}$ are regular, they are implementable. In fact Wonham and Ramadge [28] have shown that it is possible to compute a generator $\mathcal{R}$ that represents exactly the behavior of controlled system $V/\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}$, for some supervisor V able to control $K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$. Such generator $\mathcal{R}$ will capture not only the control actions of supervisor V, but also all internal (uncontrollable) events of the plant. In a nutshell, extracting the controller generator amounts to projecting out the latter and transforming controllable events into behavior delegations. The whole procedure can be depicted as:

$$\langle \mathcal{S},\mathcal{T}\rangle \xrightarrow{\text{induces}} \boxed{\mathcal{G}_{\langle \mathcal{S},\mathcal{T}\rangle}} \xrightarrow{\text{control}} K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow} \xrightarrow[\text{by}]{\text{represented}} \boxed{\mathcal{R}} \xrightarrow{\text{extraction}} \boxed{cg_{\text{DES}}}$$

From a composition problem, a corresponding plant is first built. If the composition is solvable, the language $K_{\langle \mathcal{S},\mathcal{T}\rangle}{}^{\uparrow} \neq \varnothing$ is controllable (Theorems 4 and 5). In addition, we know that there exists a generator $\mathcal{R}$ representing a composition supervisor.

Given a state y of $\mathcal{R}$, we denote with $[y]$ the tuple $\langle st_t(y), st_1(y), \ldots, st_n(y)\rangle$ extracting the full composition state from $\mathcal{R}$'s state y, where function $st_i(y)$ projects the local state of target and that of the available behaviors (this can be deterministically reconstructed from the event labels in $\mathcal{R}$ in linear time on the size of $\mathcal{R}$).

Definition 6. Let $\mathcal{R} = \langle \Sigma, Y, y_0, \rho, Y_m\rangle$ be the generator representing a supervisor. The *DES controller generator* is a finite-state structure $cg_{\text{DES}} = \langle A_t, \mathsf{Indx}, Q, [y_0], \vartheta, \omega\rangle$, where:

- A_t and Indx are the set of target actions and behavior indexes, as before;

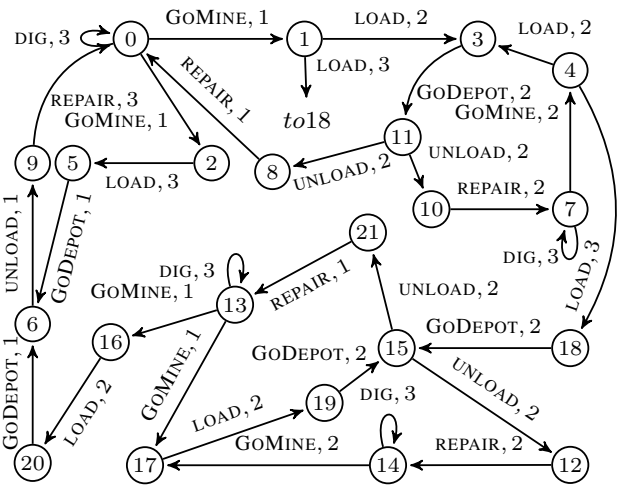

Figure 4: cg_{DES} for the example in Figure 1

- $Q = \{[y] \mid y \in Y,\ y = \rho(y_0, p),\ p \in (A_t \cdot \mathsf{Indx} \cdot \mathsf{Succ})^*\}$ is the set of full composition states reachable from initial generator's state y_0; [6]
- $[y_0]$ is the initial state of cg_{DES};
- $\vartheta : Q \times A_t \times \mathsf{Indx} \times Q$ is the transition relation such that $[y'] \in \vartheta([y], \sigma, j)$ *iff* $y' = \rho(y, \sigma \cdot j \cdot b'_j)$ for some $b'_j \in \mathsf{Succ}$. That is, ϑ outputs a transition corresponding to the delegation of action σ to the j-th module iff there exists a transition, labeled with σ, from its current state (namely, iff there exists a σ-successor b'_j); and
- $\omega : Q \times A_t \to 2^{\mathsf{Indx}}$ is the behavior selection function, such that $\omega(q, \sigma) = \{j \mid \exists q' \in \vartheta(q, \sigma, j)\}$, which just "reads" the function ϑ. ■

A controller generator cg for a composition problem is able to *generate* controllers P such that $P(h, \sigma) \in cg(last(h), \sigma)$, where h is a system history and σ is a target action request (cf. Section 2.1). Similarly, we say here that a DES controller generator cg_{DES} *generates* controllers P such that $P(h, \sigma) \in \omega(last(h), \sigma)$.

Example 7. Figure 4 represents the complete DES CG for the plant in Figure 3. For instance, if the user requests action LOAD from state 1 (namely, $\langle t_1, a_3, b_0, c_0\rangle$), the controller generator allows both behaviors $\mathcal{B}_2$ and $\mathcal{B}_3$ to be scheduled, i.e., $\omega(\langle t_1, a_3, b_0, c_0\rangle, \text{GOMINE}) = \{2, 3\}$. Therefore, any controller P generated by cg_{DES} will be such that $P(h, \text{LOAD}) \in \{2, 3\}$, where h is any system history such that $last(h) = \langle t_1, a_3, b_0, c_0\rangle$.

Note how sequences of transition in the plant (an action request, a delegation and a behaviour evolution) are compressed in cg_{DES} into a single transition. For instance, the sequence $g_0 \xrightarrow{\text{DIG}} g_1 \xrightarrow{3} g_4 \xrightarrow{c_0} g_0$ is combined into the cg_{DES}'s

[6] Recall that Succ is the set of all behaviors states as defined on page 6.

0	$\langle t_0, a_0, b_0, c_0\rangle$	4	$\langle t_1, a_3, b_0, c_0\rangle$	8	$\langle t_4, a_3, b_0, c_0\rangle$	12	$\langle t_4, a_3, b_2, c_1\rangle$	16	$\langle t_1, a_1, b_0, c_1\rangle$	20	$\langle t_2, a_1, b_0, c_1\rangle$
1	$\langle t_1, a_3, b_0, c_0\rangle$	5	$\langle t_2, a_1, b_0, c_1\rangle$	9	$\langle t_4, a_0, b_0, c_1\rangle$	13	$\langle t_0, a_0, b_0, c_1\rangle$	17	$\langle t_1, a_3, b_0, c_1\rangle$	21	$\langle t_2, a_1, b_0, c_1\rangle$
2	$\langle t_1, a_1, b_0, c_0\rangle$	6	$\langle t_3, a_2, b_0, c_1\rangle$	10	$\langle t_4, a_3, b_2, c_0\rangle$	14	$\langle t_0, a_3, b_3, c_1\rangle$	18	$\langle t_2, a_3, b_0, c_1\rangle$		
3	$\langle t_2, a_3, b_0, c_0\rangle$	7	$\langle t_0, a_3, b_3, c_0\rangle$	11	$\langle t_3, a_3, b_1, c_0\rangle$	15	$\langle t_3, a_3, b_1, c_1\rangle$	19	$\langle t_2, a_3, b_0, c_1\rangle$		

transition from state 0 which is labelled $\langle \text{DIG}, 3\rangle$, such that $\omega(0, \text{DIG}) = \{3\}$. □

The following result demonstrates the correctness of our DES-based approach to compute controller generators.

Theorem 6. *A controller P is a composition of $\mathcal{T}$ in $\mathcal{S}$ iff it is generated by DES CG cg_{DES}.*

Proof. ($\Rightarrow$) Assume that there exists a composition P that can not be generated by cg_{DES}. It means that for some target trace τ and induced history $h \in \mathcal{H}_{P,\tau}$ it is $P(h, \sigma) = j \notin cg_{\text{DES}}(last(h), \sigma)$ for some σ. By construction of cg_{DES}, this means that there is no $p = (\sigma \cdot j \cdot b'_j) \in (\Sigma_t \cdot \mathsf{Indx} \cdot \mathsf{Succ})$ such that $\rho(last(h), p)$ is defined in $\mathcal{R}$. By definition of $\mathcal{R}$, it contradicts Theorem 5. ($\Leftarrow$) Assume that there exists a controller P generated by cg_{DES} which is not a composition. Similarly, this contradicts Theorem 4. □

Note how, from Corollary 2 and the definition of cg_{DES}, we get that $\langle t, b_1, \ldots, b_n\rangle \in Q$ iff $t \preceq_{\text{ND}} \langle b_1, \ldots, b_n\rangle$, which matches the definition of controller generator in [11].

Also, the DES-based approach is optimal w.r.t. computational complexity.

Theorem 7. *Computing the DES controller generator cg_{DES} can be done in exponential time in the number of available behaviors, and polynomial in their size.*

The size of the plant $\mathcal{G}_{\langle \mathcal{S}, \mathcal{T}\rangle}$ is indeed exponential in the number of behaviors, and the procedure to synthesize the supervisor (that is, to extract $\mathcal{R}$) is polynomial in the size of the plant and the generator for the specification [28, 13]. It follows then that computing the DES controller generator can be done in exponential time in the number of behaviors, which is the best we can hope for [10].

3.2 Implementation in TCT

We close this section by noting that there are, in fact, several tools available for the automated synthesis of supervisors for a discrete event system. The reduction above allows us to use those tools off-the-shelf. In particular, we have used TCT [31], in which both the plant and the specification are formalized as generators, to extract the generator $\mathcal{R}$ encoding a supervisor for controlling the language $K_{\langle \mathcal{S}, \mathcal{T}\rangle}{}^{\uparrow}$.

In TCT, generators and recognizers are represented as standard DES in the form of a 5-tuple $\langle Size, Init, Mark, Voc, Tran\rangle$: *Size* is the number of states (the standard state set is $\{0, \ldots, Size - 1\}$); *Init* is the initial state (always taken to be 0); *Mark* lists the marker states; *Voc* are the vocal states (not needed here); and *Tran* are the transitions. A transition is a triple $\langle s, e, s'\rangle$ representing a transition from s to s' with label $e \in E$, where E is the set of possible event labels, encoded as integers. To distinguish between controllable and uncontrollable events, the tool assumes that all controllable events are represented with even integers, uncontrollable ones with odd integers.

For example, the generator $\mathcal{G}$ and the specification K_2 from Example 3 are encoded in `ADS` format as follows:

```
G
State size: 4
Marker states: 0
Vocal states:

Transitions:
0 1 1
1 3 1
1 0 2
2 9 3
2 5 1
1 7 0

K
State size: 1
Marker states: 0
Vocal states:

Transitions:
0 1 0
0 3 0
0 0 0
0 5 0
0 7 0
```

Events: 0 = *break*, 1 = *on*, 3 = *operate*, 5 = *repair*, 7 = *off*, 9 = *dismantle*

The encoding is self explanatory and amounts to declaring the plant generator `G` and the generator for the desired specification `K`. A transition line `0 1 1` in `G` encodes the transition $0 \xrightarrow{on} 1$ in Figure 3. Observe how the generator for `K` has only one state, and any event but *dismantle* is represented by a loop on that state. Indeed, recall that this generator captures the specification K_2, which excludes *dismantle* (event 9).
The following steps are required for using the TCT tool to compute the supervisor:

1. Create the plant `G` and the specification `K` in `ADS` format for $\mathcal{G}_{\langle \mathcal{S}, \mathcal{T}\rangle}$ and $K_{\langle \mathcal{S}, \mathcal{T}\rangle}$ (above);
2. Use the `FD` command to convert `DES` files in `ADS` format;
3. Use `supcon(G,K)` command to compute `R`, namely the generator representing the supremal controllable sublanguage of `K`, i.e., $K_{\langle \mathcal{S}, \mathcal{T}\rangle}{}^{\uparrow}$;

4. Use `supreduce(R)` to minimize the generator's size (this step is optional);
5. Compute *control patterns* via `condat(G,R)`, where a control pattern is the set of (controllable) events that must be disabled in each state of the plant.

In this example, the generator `R` is as the plant in Figure 2 but without state 3. Accordingly, the TCT command `condat(G,R)` outputs `2:9`, which (only) disables *dismantle* (event `9`) in state 2 of $\mathcal{G}$.

4 Supremal Realizable Target Fragment

Suppose we are given a target behavior $\mathcal{T}$ and an available system $\mathcal{S}$ such that no exact composition for $\mathcal{T}$ in $\mathcal{S}$ is possible—the target *cannot* be completely realized in the system. A mere "no solution" answer is unsatisfactory in most cases. The need to look beyond exact compositions was first recognized by Stroeder and Pagnucco [26], were they argue that one should look for "approximations" in problem instances not admitting exact compositions. Then, Yadav and Sardina [29] provided the first attempt to define and study properties of such approximations. Subsequently, these optimal approximations were refined and named *supremal realizable target fragments* (SRTF) in [30]. In this section, we show how to adapt the composition plant $\mathcal{G}_{\langle\mathcal{S},\mathcal{T}\rangle}$ to look for SRTFs (rather than exact composition) for the special case of deterministic available systems (i.e., one where all available behaviors are deterministic).

Roughly speaking, an SRTF is a "fragment" of the target behavior which accommodates an exact composition and is closest to the (original) target module. It turns out that there is an exact solution for the original target iff there exists an SRTF that is simulation equivalent to it (a property that can be checked in polynomial time). More surprising is the fact that SRTFs are unique (up to simulation equivalence). Concretely, Yadav and Sardina [29] first proposed to allow for *nondeterministic* target behaviors but model user's requests as target transitions (instead of just actions). By doing that, full controllability of the target module is maintained while allowing approximating the original target as much as possible. The definition of SRTFs, then, relies on the formal notion of simulation [16], already discussed in Section 2.1. A target behavior $\tilde{\mathcal{T}} = \langle \tilde{T}, \tilde{A}_t, \tilde{t_0}, \tilde{\delta_t}\rangle$ is a *realizable target fragment* (RTF) of original target specification $\mathcal{T} = \langle T, A_t, t_0, \delta_t\rangle$ in available system $\mathcal{S}$ iff

- $\tilde{\mathcal{T}}$ is simulated by $\mathcal{T}$ (i.e., $\tilde{\mathcal{T}} \preceq \mathcal{T}$); and
- $\tilde{\mathcal{T}}$ has an exact composition in $\mathcal{S}$.

Then, an RTF $\tilde{\mathcal{T}}$ is *supremal* (SRTF) iff there is no other RTF $\tilde{\mathcal{T}}'$ such that $\tilde{\mathcal{T}} \prec \tilde{\mathcal{T}}'$ (i.e., $\tilde{\mathcal{T}} \preceq \tilde{\mathcal{T}}'$ but $\tilde{\mathcal{T}} \not\preceq \tilde{\mathcal{T}}'$). Intuitively, a supremal RTF is the closest alternative to the original target that can be completely realized. An alternate way of looking at SRTFs is to view them as the (infinite) union of all RTFs [30].

The question we are interested in is as follows: *is it possible to adapt the DES-based composition framework developed above to obtain SRTFs rather than exact compositions?*

We answer this question positively for the case when available behaviors in $\mathcal{S}$ are deterministic. The key idea to synthesizing SRTFs by controlling a DES plant is the fact that we are no longer committed to realize *all* target traces: we only need to realize *as many as possible*. As a consequence, one can see target actions no more as nondeterministic requests over which we have no control, but instead as actions one may decide to fulfill or not, possibly depending on context. Technically this means that events corresponding to user's requests are now assumed *controllable*—the supervisor can enable certain requests and disable others.[7]

So, we start by assuming that system $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n\rangle$ is deterministic and that, following [29], target modules $\mathcal{T}$ may be, in general, nondeterministic: there may be two transitions $\langle t, a, t'\rangle, \langle t, a, t''\rangle \in \delta_t$ such that $t' \neq t''$. To maintain controllability, though, user's requests amount to *target transitions* of the form $\theta = \langle t, a, t'\rangle \in \delta_t$. Still, the task is to implement the action a in the chosen transition θ via behavior delegation.

So, let us define an alternative DES plant $\hat{\mathcal{G}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ suitable for synthesising supervisors encoding SRTFs. Note that, since system $\mathcal{S}$ is deterministic, the whole process for one target request involves now only *two* γ-transitions, both via controllable events, namely, $(\theta \cdot j) \in (\delta_t \cdot \mathsf{Indx})$. Note also that the original target is still assumed to be deterministic (the alternative supremal target may be nondeterministic).

Definition 7. Let $\mathcal{S}$ and $\mathcal{T}$ be a (deterministic) system and a target module, resp., as in Section 3. The *maximal composition plant* is defined as $\hat{\mathcal{G}}_{\langle\mathcal{S},\mathcal{T}\rangle} = \langle \Sigma, G, g_0, \gamma, G_m\rangle$, where:

- $\Sigma = \Sigma_c \cup \Sigma_u$ is the set of events of the plant, where $\Sigma_u = \varnothing$ and $\Sigma_c = \mathsf{Indx} \cup \delta_t$;
- $G = T \times B_1 \times \ldots \times B_n \times (\delta_t \cup \{e\})$ is the set of plant states;
- $g_0 = \langle t_0, b_{01}, \ldots, b_{0n}, e\rangle$ is the initial state of $\hat{\mathcal{G}}_{\langle\mathcal{S},\mathcal{T}\rangle}$;
- $\gamma : G \times \Sigma \to G$ is the plant's transition function where $\gamma(\langle t, b_1, \ldots, b_n, \mathsf{req}\rangle, \sigma)$ is equal to:
  - $\langle t, b_1, \ldots, b_n, \sigma\rangle$ if $\mathsf{req} = e$ and $\sigma \in \delta_t$;

[7] Note that unlike what is often done in a standard DES applications, we do not want to control something in the real world that was not previously controllable, something that usually requires more capabilities (e.g., new actuators). The "real world" remains the same (i.e., the available behaviors), and we now control what the user will be allowed to potentially request (i.e., the final target specification, the SRTF).

– $\langle t', b_1, \ldots, b'_\sigma, \ldots, b_n, e\rangle$ if $\mathsf{req} = \langle t, a, t'\rangle \in \delta_t, \sigma \in \mathsf{Indx}$ and $b'_\sigma = \delta_\sigma(b_\sigma, a)$.

- $G_m = T \times B_1 \times \ldots \times B_n \times \{e\}$. ■

Notably, both target transition requests and behavior delegations are now controllable: the supervisor is allowed to forbid (i.e., disable) target requests.

As before, we just take $\hat{K}_{\langle\mathcal{S},\mathcal{T}\rangle} = L_m(\hat{\mathcal{G}}_{\langle\mathcal{S},\mathcal{T}\rangle})$ as the specification language to control (in the maximal composition plant). To build a SRTF for target $\mathcal{T}$ in system $\mathcal{S}$, we first compute the language $\hat{K}_{\langle\mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$, and then build its corresponding generator $\hat{\mathcal{R}} = \langle\Sigma, Y, y_0, \rho, Y_m\rangle$. Finally, we extract $\hat{\mathcal{R}}$ the alternative, possibly nondeterministic, target behavior $\mathcal{T}^*_{\langle\mathcal{S},\mathcal{T}\rangle} = \langle T^*, A_t, y_0, \delta^*_t\rangle$, where:

- $T^* = \{y \mid y \in Y, p \in (\delta_t \cdot \mathsf{Indx})^*, y = \rho(y_0, p)\}$; and
- $\delta^*_t \subseteq T^* \times A_t \times T^*$ is such that $y' \in \delta^*_t(y, a)$ *iff* $y' = \rho(y, \theta \cdot j)$, where $j \in \mathsf{Indx}$, $\theta \in \delta_t$, and $\theta = \langle t, a, t'\rangle$.

Next, we relate system histories to words of plant $\hat{\mathcal{G}}_{\langle\mathcal{S},\mathcal{T}\rangle}$. Differently from before, however, histories do not hold enough information for this purpose, because the target is now nondeterministic. We then make use of the following definitions, to relate a target trace τ *and* an induced system history $h \in \mathcal{H}_{P,\tau}$ (for some P) to a word in $L(\hat{\mathcal{G}}_{\langle\mathcal{S},\mathcal{T}\rangle})$. Given a target trace $\tau = t_0 \xrightarrow{a_1} \cdots \xrightarrow{a_\ell} t_\ell$, the word $\mathtt{word}(\tau, h)$ corresponding to an induced system history $h = \vec{b}_0 \xrightarrow{a_1, j_1} \cdots \xrightarrow{a_\ell, j_\ell} \vec{b}_\ell$ is

$$\mathtt{word}(\tau, h) := (\langle t_0, a_1, t_1\rangle \cdot j_1) \cdot \ \ldots \ \cdot (\langle t_{\ell-1}, a_\ell, t_\ell\rangle \cdot j_\ell).$$

We now present the key results for our technique.

Lemma 2. *P is a composition for an RTF $\tilde{\mathcal{T}}$ of $\mathcal{T}$ in $\mathcal{S}$ iff $\{\mathtt{word}(\tau, h) \mid \tau \in \tilde{\mathcal{T}} \wedge h \in \mathcal{H}_{P,\tau}\} \subseteq \hat{K}_{\langle\mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$.*

Proof. ($\Rightarrow$) Assume by contradiction that there exists a composition P for $\tilde{\mathcal{T}}$ in $\mathcal{S}$ such that for some target trace $\tau = t_0 \xrightarrow{a_1} t_1 \xrightarrow{a_2} \cdots \xrightarrow{a_\ell} t_\ell$ of $\tilde{\mathcal{T}}$ and induced history $h \in \mathcal{H}_{P,\tau}$, we have $P(h, \theta) = j$ but $\mathtt{word}(\tau, h) \cdot \theta \cdot j \notin \hat{K}_{\langle\mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$, where $\theta = \langle t_\ell, a, t_{\ell+1}\rangle$ is the new target transition been requested. This implies that $\mathtt{word}(\tau, h) \cdot \theta \cdot j$ is not allowed by V, from the initial state g_0. Similarly to Lemma 1, either

(1) $\mathtt{word}(\tau, h) \cdot \theta \notin L(\hat{\mathcal{G}}_{\langle\mathcal{S},\mathcal{T}\rangle})$ or

(2) $\mathtt{word}(\tau, h) \cdot \theta \cdot j \notin L(\hat{\mathcal{G}}_{\langle\mathcal{S},\mathcal{T}\rangle})$ or

(3) for all words $w \in L_m(\hat{\mathcal{G}}_{\langle\mathcal{S},\mathcal{T}\rangle})$ with $w > \mathtt{word}(\tau, h) \cdot \theta \cdot j$ we have $w \notin \overline{\hat{K}_{\langle\mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}}$.

Without loss of generality, assume $\theta = \langle t_\ell, a, t_{\ell+1}\rangle$. If (1) is true, then from the definition of $\hat{\mathcal{G}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ (transition function γ, first item) the transition θ is not in δ_t, thus $\tilde{\mathcal{T}}$ is not an RTF of $\mathcal{T}$, as $\tilde{\mathcal{T}} \not\leq \mathcal{T}$. Similarly, case (2) violates the assumption that P is a composition of $\tilde{\mathcal{T}}$, as behaviour j is not able to perform the action a from its current local state. Indeed, any legal delegation of action a to any behaviour is considered in $\hat{\mathcal{G}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ (transition function γ, second item). Case (3) can be excluded by considering cases (1)(2) by induction on ℓ.

($\Leftarrow$) Assume by contradiction this is not the case. Then there exists a word $\mathtt{word}(\tau, h) \in \hat{K}_{\langle\mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$ for some trace τ of an RTF $\tilde{\mathcal{T}}$ of $\mathcal{T}$ in $\mathcal{S}$ and history $h \in \mathcal{H}_{P,\tau}$ such that P is not a composition of $\tilde{\mathcal{T}}$ in $\mathcal{S}$. It means that for some $\theta = \langle st_t(last(h)), a, t'\rangle$ in $\tilde{\mathcal{T}}$ we have $P(h, \theta) = j$ but behavior j can not perform action a, namely there is no b'_j such that $b'_j \in \delta_j(st_j(last(h)), a)$. Hence, according to $\hat{\mathcal{G}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ (and its transition function γ), $\mathtt{word}(h, \tau) \cdot \theta \cdot j \notin L(\hat{\mathcal{G}}_{\langle\mathcal{S},\mathcal{T}\rangle})$, and we get a contradiction. □

Theorem 8 (Soundness). *Let $\mathcal{T}$ be a target and $\mathcal{S}$ a deterministic system. Then, $\mathcal{T}^*_{\langle\mathcal{S},\mathcal{T}\rangle}$ is a SRTF of $\mathcal{T}$ in $\mathcal{S}$.*

Proof. By construction $\mathcal{T}^*_{\langle\mathcal{S},\mathcal{T}\rangle}$ is an RTF of $\mathcal{T}$ in $\mathcal{S}$. The proof is left to the reader, as it is evident by the fact that $L(\hat{\mathcal{R}}) = \overline{\hat{K}_{\langle\mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}} \subseteq L(\hat{\mathcal{G}}_{\langle\mathcal{S},\mathcal{T}\rangle})$, and transitions in $\hat{\mathcal{G}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ are only defined wrt $\mathcal{T}$'s evolution (function δ_t).

It remains to show that $\mathcal{T}^*_{\langle\mathcal{S},\mathcal{T}\rangle}$ is indeed the maximal one. Suppose $\mathcal{T}^{\uparrow}$ is the SRTF of $\mathcal{T}$ in $\mathcal{S}$ and $\mathcal{T}^*_{\langle\mathcal{S},\mathcal{T}\rangle} \not\leq \mathcal{T}^{\uparrow}$. Therefore, there exists a trace $\tau = t_0 \xrightarrow{a_1} \cdots \xrightarrow{a_\ell} t_\ell \xrightarrow{a_{\ell+1}} t_{\ell+1}$ of $\mathcal{T}^{\uparrow}$ that is not in $\mathcal{T}^*_{\langle\mathcal{S},\mathcal{T}\rangle}$, and the simulation "breaks" at t_ℓ. Formally, for any trace $t^*_0 \xrightarrow{a_1} \cdots \xrightarrow{a_\ell} t^*_\ell$ of $\mathcal{T}^*_{\langle\mathcal{S},\mathcal{T}\rangle}$ with $t^*_0 = y_0$ and the same action sequence $a_1 \cdots a_\ell$ as τ, there is no transition $t^*_\ell \xrightarrow{a_{\ell+1}} t^*_{\ell+1}$: $\mathcal{T}^*_{\langle\mathcal{S},\mathcal{T}\rangle}$ is unable to perform action $a_{\ell+1}$. However, by Lemma 2, the word $\mathtt{word}(\tau, h)$ is in $\hat{K}_{\langle\mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$, and the trace $t^*_0 \xrightarrow{a_1} \cdots \xrightarrow{a_{\ell+1}} t^*_{\ell+1}$ has to be a trace of $\mathcal{T}^*_{\langle\mathcal{S},\mathcal{T}\rangle}$. So we get a contradiction. Thus $\mathcal{T}^{\uparrow}$ does not exist and $\mathcal{T}^*_{\langle\mathcal{S},\mathcal{T}\rangle}$ is an SRTF of $\mathcal{T}$ in $\mathcal{S}$. □

Observe, the controller generator for $\mathcal{T}^*_{\langle\mathcal{S},\mathcal{T}\rangle}$ can be computed *while* building the SRTF itself, by tracking delegations in $\hat{\mathcal{R}}$, similar to DES-based composition. In fact, we have:

Theorem 9 (Completeness). *Let V be a nonblocking supervisor such that $L_m(V/\hat{\mathcal{G}}_{\langle\mathcal{S},\mathcal{T}\rangle}) = \hat{K}_{\langle\mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$. Then, every composition P for $\mathcal{T}^*_{\langle\mathcal{S},\mathcal{T}\rangle}$ in system $\mathcal{S}$ can be induced by V.*

Proof. Assume by contradiction that there exists a composition P' which can not be induced by V, i.e., it is such that $P'(h, \theta) \notin V(\mathtt{word}(\tau, h) \cdot \theta)$ for some target transition $\theta = \langle t^*_{\ell-1}, a_\ell, t^*_\ell\rangle$ and target trace $\tau = t^*_0 \xrightarrow{a_1} \cdots \xrightarrow{a_\ell} t^*_\ell$ in $\mathcal{T}^*_{\langle\mathcal{S},\mathcal{T}\rangle}$ and some induced system history $h \in \mathcal{H}_{P',\tau}$. Assume $P'(h, \theta) = j$. It follows that $\mathtt{word}(\tau, h) \cdot \theta \cdot j \notin \hat{K}_{\langle\mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$, which contradicts Lemma 2. □

In words, every supervisor that can control language $\hat{K}_{\langle\mathcal{S},\mathcal{T}\rangle}{}^{\uparrow}$ in the maximal composition plant $\hat{\mathcal{G}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ encodes all exact compositions of the SRTF $\mathcal{T}^*_{\langle\mathcal{S},\mathcal{T}\rangle}$ built above.

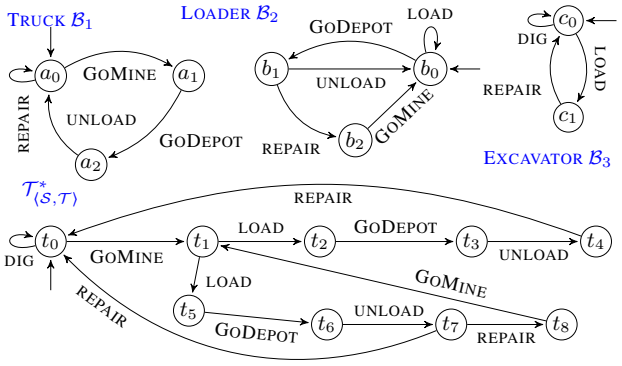

Figure 5: SRTF for a deterministic variant ($\mathcal{T}$ as in Figure 1).

Corollary 2. *Given a plant $\hat{\mathcal{G}}_{\langle\mathcal{S},\mathcal{T}\rangle}$ for $\mathcal{S}$ and $\mathcal{T}$ as above, if $\hat{K}_{\langle\mathcal{S},\mathcal{T}\rangle}{\uparrow} \neq \varnothing$ then for any word $w \in \hat{K}_{\langle\mathcal{S},\mathcal{T}\rangle}{\uparrow}$, we have that $st_t(g) \preceq_{ND} \langle st_1(g), \ldots, st_n(g)\rangle$ where $g = \gamma(g_0, w)$.*

The proof proceeds as in the basic case.

Example 8. Figure 5 depicts the SRTF for a deterministic variant of the problem instance of Figure 1, in which the excavator can not load the truck twice without being repaired. Observe that $\mathcal{T}^*_{\langle\mathcal{S},\mathcal{T}\rangle}$ and $\mathcal{T}$ are simulation equivalent, hence we know that an exact composition exists for the original specification K. However, since actions LOAD and REPAIR are now nondeterministic in $\mathcal{T}^*_{\langle\mathcal{S},\mathcal{T}\rangle}$ (from t_1 and t_7, respectively), one could, in principle, chose to delegate a nondeterministic action to different behaviors based on the transition requested (cf. Figure 4). Hence, the $\mathcal{T}^*_{\langle\mathcal{S},\mathcal{T}\rangle}$ is now more informative than $\mathcal{T}$, in the sense that it allows for more solutions if the user commits to subsequent choices (here, whether to dig next or not). The so-obtained representation of the SRTF is not minimal, but can be easily minimised (e.g., by using the `supreduce` command in TCT — see Section 3.2.) □

5 Conclusions

From an Computer Science perspective, planning, SCT, and behavior composition are all synthesis problems: build a plan, supervisor, or controller, respectively. Observe that, at the core, these problems are concerned with qualitative temporal decision making in dynamic domains and exhibit strong resemblances in how their problem components are modeled (e.g., using transition system-like models) and the solution techniques used (e.g., model checking, search, etc.). In fact, exploration of the relationship between these three synthesis tasks has already gained attention [1, 7, 2].

The behavior composition problem can be considered as a planning problem for a maintenance goal, namely, to *always* satisfy the target's request. There, a plan (i.e., a controller) prescribes behavior delegations rather than domain actions [21]. In particular, various forms of composition problems have been considered under various planning frameworks, including planning as model checking [17], planning in asynchronous domains [7], and nondeterministic planning [21].

With respect to SCT, planning techniques have been used for both synthesis of supervisors [2, 3] as well as for diagnosis problems [14]. However, the link between composition and SCT still remains unexplored. To our knowledge, the only available literature deals with showing the decidability of mediator synthesis for web-service composition by reduction to DES [1]. Such work considers a composition setting involving web services able to exchange messages, and the task is to synthesise a *mediator* able to communicate with them to realize the target specification, instead of an orchestrator (i.e., a controller) that schedules them.

From the outset, it may seem that behavior composition and SCT are tackling the same problem, though maybe from different perspectives: SCT from an Engineering perspective and composition from a Computer Science one. Nonetheless, the inherent control problem in SCT and behavior composition are different in nature. In the latter, one seeks to control the available behaviors, whereas in the former one can prevent (some of the) actions. Consider the simple example shown in Figure 6 with a nondeterministic behavior $\mathcal{B}_1$ and a deterministic behavior $\mathcal{B}_2$. See that both behaviors share the action x; hence, in the enacted system $\mathcal{S}$, x will be nondeterministic for $\mathcal{B}_1$ but not for $\mathcal{B}_2$ (as shown by the indexes used in $\mathcal{S}$). The input in SCT is the whole plant, and it does not have a notion analogous to available behaviors. Therefore, component-based nondeterminism cannot be captured (directly) in a plant, and one has to make delegation events (i.e., indexes) explicit in the plant, as we showed in this paper. Another important mismatch has to do with the semantics of nondeterminism: the nondeterminism of controllable actions in a plant is *angelic*, in the sense that the supervisor can control its evolution. On the other hand, nondeterminism of available behaviors is *devilish*, as it cannot be controlled. This is one of the reasons why, as far as we know, DES frameworks do not have a notion similar to ND-simulation [11]. Indeed, uncertainty is modelled here via (deterministic) uncontrollable events [28], whereas nondeterminism [11] is used to model uncertainty (and partial controllability) in behavior composition. Lastly, the term "composition" itself differs considerably: in SCT it refers to synchronous product between sub-systems, instead of an asynchronous realization in behavior composition literature.

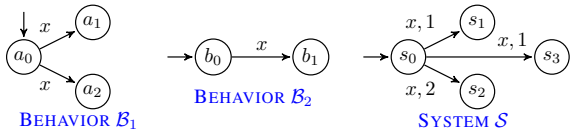

Figure 6: Comparing DES and Behavior Composition.

Notwithstanding all the above differences, this paper shows that a link can indeed be drawn. In particular, we have demonstrated that solving an AI behavior composition problem can be seen as finding a supervisor for a certain plant. In doing so, one can expect to leverage on the solid foundations and extensive work in SCT, as well as on the tools available in those communities. We have shown, for instance, that the DES-based encoding can accommodate (meta-level) constraint on the composition in a straightforward manner. In addition, we detailed how to slightly adapt the encoding to look for "the best possible" target realization when a perfect one does not exist, though only for the case of deterministic systems. For practical applicability, experimental work should follow the work presented here to check whether existing tools in SCT provide any advantages over existing composition techniques via game solvers [9, 12] or automated planners [21].

Once the formal relationship between the two different synthesis tasks has been established, many possibilities for future work open up. In fact, we would like to import notions and techniques common in SCT into the composition setting, such as hierarchical and tolerance supervision/composition [8]. An interesting aspect to look at is how to use the marked language of specification K in order to encode *constraints*. For example, one may want to impose that certain complex (high-level) tasks or processes built from domain action executions may not be started if their termination cannot be guaranteed. So, some goods in a factory production chain should not be cleaned unless it is guaranteed that they will be packaged and disposed afterwards. So far, we have only used the marked language (of the plant) to force complete termination of each action-request and delegation step. On the other direction, probably the most interesting aspect to explore is the use of automated planning systems and game solvers to solve DES problems.

References

[1] Philippe Balbiani, Fahima Cheikh, and Guillaume Feuillade. Composition of interactive web services based on controller synthesis. In *Proc. of the IEEE Congress on Services (SERVICES)*, pages 521–528, 2008.

[2] Michel Barbeau, Froduald Kabanza, and Richard St-Denis. Synthesizing plant controllers using real-time goals. In *Proc. of IJCAI*, pages 791–800, 1995.

[3] Michel Barbeau, Froduald Kabanza, and Richard St-Denis. An efficient algorithm for controller synthesis under full observation. *Journal of Algorithms*, 25(1):144–161, 1997.

[4] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Rick Hull, and Massimo Mecella. Automatic composition of transition-based semantic web services with messaging. In *Proc. of the International Conference on Very Large Databases (VLDB)*, pages 613–624, 2005.

[5] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Massimo Mecella. Automatic service composition based on behavioural descriptions. *International Journal of Cooperative Information Systems*, 14(4):333–376, 2005.

[6] Daniela Berardi, Fahima Cheikh, Giuseppe De Giacomo, and Fabio Patrizi. Automatic service composition via simulation. *International Journal of Foundations of Computer Science*, 19(2):429–452, 2008.

[7] Piergiorgio Bertoli, Marco Pistore, and Paolo Traverso. Automated composition of web services via planning in asynchronous domains. *Artificial Intelligence*, 174(3):316–361, 2010.

[8] Christos G. Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems*. Springer, Secaucus, NJ, USA, 2006. ISBN 0387333320.

[9] Giuseppe De Giacomo and Fabio Patrizi. Automated composition of nondeterministic stateful services. In *Proc. of the International Workshop on Web Services and Formal Methods (WSFM)*, volume 6194 of *LNCS*, pages 147–160. Springer, 2010.

[10] Giuseppe De Giacomo and Sebastian Sardina. Automatic synthesis of new behaviors from a library of available behaviors. In Manuela M. Veloso, editor, *Proc. of IJCAI*, pages 1866–1871, Hyderabad, India, January 2007.

[11] Giuseppe De Giacomo, Fabio Patrizi, and Sebastian Sardina. Automatic behavior composition synthesis. *Artificial Intelligence Journal*, 196:106–142, 2013. doi: 10.1016/j.artint.2012.12.001.

[12] Guiseppe De Giacomo and Paolo Felli. Agent composition synthesis based on ATL. In *Proc. of Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 499–506, Toronto, Canada, 2010. IFAAMAS.

[13] P. Gohari and W. M. Wonham. On the complexity of supervisory control design in the RW framework. *IEEE Transactions on Systems, Man, and Cybernetics*, 30(5):643–652, 2000. ISSN 1083-4419.

[14] Alban Grastien, J Rintanen Anbulagan, Jussi Rintanen, and Elena Kelareva. Diagnosis of discrete-event systems using satisfiability algorithms. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, 2007.

[15] Yoad Lustig and Moshe Y. Vardi. Synthesis from component libraries. In *Proc. of the International Conference on Foundations of Software Science and Computational Structures (FOSSACS)*, volume 5504 of *LNCS*, pages 395–409, 2009.

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

[17] Marco Pistore, Fabio Barbon, Piergiorgio Bertoli, Dmitry Shaparau, and Paolo Traverso. Planning and monitoring web service composition. In *Artificial Intelligence: Methodology, Systems, and Applications*, pages 106–115. Springer, 2004.

[18] Knut Åkesson, Martin Fabian, Hugo Flordal, and Arash Vahidi. Supremica – a tool for verification and synthesis of discrete event supervisors. In *Proc. of the 11th Mediterranean Conference on Control and Automation*, 2003.

[19] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25:206–230, 1987. ISSN 0363-0129.

[20] P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[21] Miguel Ramirez, Nitin Yadav, and Sebastian Sardina. Behavior composition as fully observable non-deterministic planning. In *Proc. of ICAPS*, pages 180–188, 2013.

[22] C. Reiser, A.E.C. da Cunha, and J.E.R. Cury. The environment GRAIL for supervisory control of discrete event systems. In *Proc. of 8th International Workshop on Discrete Event Systems*, pages 390 –391, july 2006.

[23] L. Ricker, S. Lafortune, and S. Gene. DESUMA: A tool integrating GIDDES and UMDES. In *Proc. of 8th International Workshop on Discrete Event Systems workshop*, pages 392 –393, 2006.

[24] Jussi Rintanen. Complexity of planning with partial observability. In *Proc. of ICAPS*, pages 345–354, 2004.

[25] Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. Foundations of Generalized Planning. Technical Report UM-CS-2008-039, Dept. of Computer Science, Univ. of Massachusetts, Amherst, 2008.

[26] Thomas Stroeder and Maurice Pagnucco. Realising deterministic behaviour from multiple non-deterministic behaviours. In *Proc. of IJCAI*, pages 936–941, Pasadena, CA, USA, July 2009. AAAI Press.

[27] W. M. Wonham. Supervisory control of discrete-event systems. Technical Report ECE 1636F/1637S 11-12, University of Toronto, Canada, 2012.

[28] W. M. Wonham and P. J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal on Control and Optimization*, 25(3):637–659, 1987.

[29] N. Yadav and S. Sardina. Qualitative approximate behavior composition. *Logics in Artificial Intelligence*, pages 450–462, 2012.

[30] Nitin Yadav, Paolo Felli, De Giacomo Giuseppe, and Sebastian Sardina. Supremal realizability of behaviors with uncontrollable exogenous events. In Francesca Rossi, editor, *Proc. of IJCAI*, pages 1176–1182, Beijing, China, August 2013. AAAI Press.

[31] Zhonghua Zhang and W. M. Wonham. STCT: An efficient algorithm for supervisory control design. In *Symposium on Supervisory Control of Discrete Event Systems*, pages 249–6399, 2001.