Ensuring the conformance of reactive discrete-event systems by means of supervisory control

Thierry Jéron, Hervé Marchand, Vlad Rusu, Valérie Tschaen

Ensuring the conformance of reactive discrete-event systems by means of supervisory control

Thierry Jéron*, Hervé Marchand*, Vlad Rusu* and Valéry Tschaen*

We study the problem of controlling a plant of a system by means of an automatically computed supervisor, in order to ensure a certain conformance relation between the plant and its formal specification. The supervisor can be seen as a device that automatically fixes errors, which otherwise should have been discovered by testing and fixed by hand. The resulting controlled plant conforms to the specification and is maximal in terms of observable behavior.

1 Introduction

Conformance testing and supervisory control are two approaches for ensuring that computer-operated systems work correctly. *Conformance testing* [Tretmans, 1996] consists in comparing the observable behavior of a plant of a system with respect to those allowed by its formal specification. If a difference (a *non-conformance*) is detected, the plant has to be modified (typically, bugs have to be fixed) and the process is iterated until no more errors are detected. *Supervisory Control* consists in controlling a plant such that the modified (or *controlled*) system satisfies a given property [Ramadge and Wonham, 1989]. Typically, the behaviors of the controlled plant are constrained to remain among those allowed by a formal specification.

In this paper we propose to integrate these two approaches. Specifically, we consider a Supervisory Control Problem (SCP) that consists in ensuring a certain conformance relation, named **ioco**[1], between a plant of a reactive system and its formal specification. This relation was introduced by [Tretmans, 1996] and is employed in existing conformance testing tools [Belinfante et al., 1999, Jard and Jéron, 2002]. Given a model of the plant G, and a formal specification S of its expected behavior, the control problem is the following:

*(***ioco-control***) Given a plant G and an expected specification S, construct the least constraining supervisor $\mathcal{C}$ such that the controlled plant $\mathcal{C}/G$ conforms to the specification S : $\mathcal{C}/G$* **ioco** *S.*

In other words, given a plant G and an expected specification S of the behavior of G, if G does not conform to S, the supervisor forces the controlled plant to conform to its

*IRISA/INRIA Rennes, VerTeCs Team, First.Last@irisa.fr

[1] **ioco** stands for Input/Output conformance relation. Cf. Def. 10 for the formal definition.

specification. Intuitively, this means that all errors that could have been observed by conformance testing, and then fixed by hand, are automatically eliminated by control.

The plant and the specification are here modeled using Input-Output Finite State Machines (IOFSM), a class of finite automata whose alphabet is partitioned into Inputs, Outputs, and internal events. Inputs and outputs are observable by the environment, but internal events are not.

As usual in supervisory control [Ramadge and Wonham, 1989], the alphabet of events must also be partitioned into controllable and uncontrollable, and respectively, observable and unobservable. It is assumed here that inputs (from the environment to the plant) are uncontrollable by the supervisor, whereas internal and a subset of output events are controllable. We also assume that all the events are observable by the supervisor. This is consistent with the idea that the supervisor controls the *plant*, but not the environment. The distinction between internal events and non-internal ones (inputs and outputs) is still essential, because the **ioco** conformance relation describes the observable behavior of a system from the point of view of the environment.

The rest of the paper is organized as follows. In Section 2 the basic model and notations are introduced, and the supervisory control theory and the conformance testing theory are briefly presented. In Section 3 the supervisory control problem for conformance testing is defined and solved. Section 4 reports on related work and concludes.

2 Background

This section serves as a prelude to the rest of the paper. In Section 2.1 the basic model for plants and specifications is defined. Section 2.2 is devoted to a brief presentation of the classical supervisory control problem, and Section 2.3 introduces the theory of conformance testing.

2.1 Model and notations

The basic model employed in the paper is that of Finite State Machines (FSM), defined below. The model is later refined to take into account inputs, outputs, controllable, and uncontrollable events.

Definition 1 (Finite State Machine (FSM)) *A finite state machine is a tuple* $G = (\Sigma, \mathcal{X}, x_o, \delta)$*, where* Σ *is the finite set of events,* $\mathcal{X}$ *is the finite set of states,* x_o *is the initial state, and* $\delta : \Sigma \times \mathcal{X} \rightarrow \mathcal{X}$ *is the (partial) transition function.*

The notation $\delta(\sigma, x)!$ means that there is a transition labeled by the event σ out of state x. $\delta(x)$ denotes the set of enabled events in x, i.e., $\{\sigma \in \Sigma | \delta(\sigma, x)!\}$. For

$s \in \Sigma^*$ a sequence of events, $\delta(s, x)$ denotes the state reached by taking the sequence of events s from state x (if it exists). $\delta^{-1}(x)$ denotes the set of events that lead to x.

The behavior of a plant G is defined by the prefix-closed language $\mathcal{L}(G) = \{s \in \Sigma^* | \delta(s, x_o) \neq \emptyset\}$ generated by G [Cassandras and Lafortune, 1999]. If G is equipped with a set of *final states* $\mathcal{X}_m \subseteq \mathcal{X}$, one may also define the *marked language* $\mathcal{L}_m(G) = \{s \in \Sigma^* \, | \delta(s, x_o) \in \mathcal{X}_m\}$, i.e., the set of trajectories of G ending in $\mathcal{X}_m$. $\mathcal{L}_m(G)$ can be seen as set of trajectories that complete a given task. If $\mathcal{X}_m$ is undefined, we consider that $\mathcal{X}_m = \mathcal{X}$.

We define the notion of sub-machine of a given FSM.

Definition 2 *If $G = (\Sigma, \mathcal{X}, x_o, \delta)$ is an FSM with $\mathcal{X}_m$ its set of marked states, $\Sigma' \subseteq \Sigma$ is a subset of events, and $\mathcal{X}' \subseteq \mathcal{X}$ is a subset of states, then $G_{\downarrow \Sigma', \mathcal{X}'}$ is the* sub-machine *of G restricted to events in Σ', and states in $\mathcal{X}'$, defined as follows. If $x_o \notin X'$, then $G_{\downarrow \Sigma', \mathcal{X}'}$ is empty; otherwise, $G_{\downarrow \Sigma', \mathcal{X}'} = (\Sigma', \mathcal{X}', x_o, \delta_{\downarrow \Sigma', \mathcal{X}'})$ where $\delta_{\downarrow \Sigma', \mathcal{X}'} : \Sigma' \times \mathcal{X}' \to \mathcal{X}'$ is the restriction of δ to Σ' and $\mathcal{X}'$. Its set of marked states is $\mathcal{X}'_m = \mathcal{X}_m \cap \mathcal{X}'$.*

We omit subscripts Σ' if $\Sigma' = \Sigma$ or $\mathcal{X}'$ if $\mathcal{X}' = \mathcal{X}$. Note that $\mathcal{L}(G_{\downarrow \Sigma'}) = \mathcal{L}(G) \cap \Sigma'^*$.

For $\mathcal{X}' \subseteq \mathcal{X}$ and $\Sigma' \subseteq \Sigma$, $pre_{\Sigma'}(\mathcal{X}') = \{x \in \mathcal{X} \mid \exists \sigma \in \Sigma', \exists x' \in \mathcal{X}', x' = \delta(x, \sigma)\}$ denotes the set of predecessors of $\mathcal{X}'$ by events in Σ'. We also denote by

$$reach_{\Sigma'}(\mathcal{X}') = \{x \in \mathcal{X} \mid \exists s \in \Sigma'^*, \exists x' \in \mathcal{X}', x = \delta(x', s)\}$$

the set of states reachable from $\mathcal{X}'$ by sequences of events in Σ' and

$$coreach_{\Sigma'}(\mathcal{X}') = \{x \in \mathcal{X} \mid \exists s \in \Sigma'^*, \exists x' \in \mathcal{X}', x' = \delta(x, s)\}$$

the set of coreachable states of $\mathcal{X}'$ (i.e. the set of states from which $\mathcal{X}'$ is reachable) by sequences of events in Σ'. The subscript Σ' is omitted whenever $\Sigma' = \Sigma$.

For $\Sigma' \subseteq \Sigma$, we consider the natural projection on Σ' $P_{\Sigma \to \Sigma'} : \Sigma^* \to \Sigma'^*$ defined as

$$\begin{array}{lcl} P_{\Sigma \to \Sigma'}(\varepsilon) & = & \varepsilon \text{ where } \varepsilon \text{ is the empty sequence} \\ P_{\Sigma \to \Sigma'}(\sigma) & = & \sigma \text{ if } \sigma \in \Sigma', \text{ and } \varepsilon \text{ otherwise} \\ P_{\Sigma \to \Sigma'}(s.\sigma) & = & P_{\Sigma \to \Sigma'}(s) \cdot P_{\Sigma \to \Sigma'}(\sigma) \text{ for } s \in \Sigma^* \text{ and } \sigma \in \Sigma \end{array} \quad (1)$$

That is, projection erases from a sequence all the events that do not belong to Σ'. This operation is generalized to languages. Let $L \subseteq \Sigma^*$, $P_{\Sigma \to \Sigma'}(L) = \{P_{\Sigma \to \Sigma'}(s) \mid s \in L\}$. We also consider the inverse projection $P^{-1}_{\Sigma \to \Sigma'} : \Sigma'^* \to 2^{\Sigma^*}$ where $s \in P^{-1}_{\Sigma \to \Sigma'}(s')$ iff $P_{\Sigma \to \Sigma'}(s) = s'$. For $L' \subseteq \Sigma'^*$, we denote by $P^{-1}_{\Sigma \to \Sigma'}(L') = \bigcup_{s' \in L'} P^{-1}_{\Sigma \to \Sigma'}(s')$.

Composition of FSM. The *parallel composition* of two FSM performs synchronization on their common shared events:

Definition 3 (Parallel composition) *Let $G_i = (\Sigma^i, \mathcal{X}^i, x_o^i, \delta^i)$,$i = 1, 2$, two FSMs. The parallel composition $G_1 \parallel G_2$ is the FSM $(\Sigma, \mathcal{X}, x_o, \delta)$ such that $\Sigma = \Sigma^1 \cup \Sigma^2, \mathcal{X} = \mathcal{X}^1 \times \mathcal{X}^2$, $x = \langle x_o^1, x_o^2 \rangle$, and δ is defined by: for all $x = \langle x^1, x^2 \rangle \in \mathcal{X}$ and $\sigma \in \Sigma$:*

$$\delta(\sigma, \langle x^1, x^2 \rangle) = \begin{cases} \langle \delta^1(\sigma, x^1), \delta^2(\sigma, x^2) \rangle \text{ if } \sigma \in \Sigma^1 \cap \Sigma^2 \text{ and } \delta^1(\sigma, x^1)! \wedge \delta^2(\sigma, x^2)! \\ \langle \delta^1(\sigma, x^1), x^2 \rangle \text{ if } \sigma \in \Sigma^1 \setminus \Sigma^2 \text{ and } \delta^1(\sigma, x^1)! \\ \langle x^1, \delta^2(\sigma, x^2) \rangle \text{ if } \sigma \in \Sigma^2 \setminus \Sigma^1 \text{ and } \delta^2(\sigma, x^2)! \\ \text{Undefined otherwise} \end{cases}$$

If G_1 and G_2 have sets of marked states $\mathcal{X}_m^1$ and $\mathcal{X}_m^2$, the set of marked states of G is $\mathcal{X}_m^1 \times \mathcal{X}_m^2$.

Definition 3 implies the following relations on languages and marked languages

$$\mathcal{L}(G_1 \parallel G_2) = P^{-1}_{\Sigma \to \Sigma_1}(\mathcal{L}(G_1)) \cap P^{-1}_{\Sigma \to \Sigma_2}(\mathcal{L}(G_2)) \quad (2)$$

$$\mathcal{L}_m(G_1 \parallel G_2) = P^{-1}_{\Sigma \to \Sigma_1}(\mathcal{L}_m(G_1)) \cap P^{-1}_{\Sigma \to \Sigma_2}(\mathcal{L}_m(G_2)) \quad (3)$$

When $\Sigma^1 = \Sigma^2$, $G_1 \parallel G_2$ is exactly the *synchronous product* $G_1 \times G_2$ and we get $\mathcal{L}(G_1 \times G_2) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2)$ and $\mathcal{L}_m(G_1 \times G_2) = \mathcal{L}_m(G_1) \cap \mathcal{L}_m(G_2)$.

2.2 Supervisory Control

The Supervisory Control theory deals with the control of discrete event systems. In this theory, a system (called a *plant*) is modeled by an FSM, and is assumed to have an uncontrolled behavior which may violate some required properties (e.g., safety). Hence, this behavior has to be modified by means of a feedback controller (a Supervisor) in order to achieve the given set of requirements [Ramadge and Wonham, 1989]. For this, the supervisor acts on the plant by forbidding some discrete events and allowing some others. In general, not all events can be disabled by the supervisor. Therefore, some of the events in Σ are said to be uncontrollable (Σ_{uc}), i.e., their occurrence cannot be prevented by a controller, while others are controllable (Σ_c).

Definition 4 (Plant) *A plant is an FSM $G = (\Sigma, \mathcal{X}, x_o, \delta)$, where the alphabet Σ is partitioned into two subsets Σ_{uc}, the set uncontrollable events, and Σ_c, the set controllable events. Moreover, G is equipped with a set of* final states *$\mathcal{X}_m \subseteq \mathcal{X}$.*

The behavior of the plant is then characterized by its two languages $\mathcal{L}(G)$ and $\mathcal{L}_m(G)$, as defined in Section 2.1.

Given $K \subseteq \Sigma^*$ a language over the alphabet Σ, $\overline{K}$ denotes the prefix closure of the language K. An FSM G is said to be *blocking* if $\mathcal{L}(G) \neq \overline{\mathcal{L}_m(G)}$ and *non-blocking* otherwise. It can be shown that (cf. e.g.[Cassandras and Lafortune, 1999]) G is non-blocking if its set of states $\mathcal{X}$ is exactly $reach(x_o) \cap coreach(X_m)$.

A *supervisor* is a function $\mathcal{C} : \mathcal{L}(G) \to 2^{\Sigma}$, delivering the set of events that are allowed in G by the control after a trajectory $s \in \mathcal{L}(G)$. We write $\mathcal{C}/G$ for the

closed loop system, consisting of the initial plant G controlled by the supervisor $\mathcal{C}$. The closed-loop system $\mathcal{C}/G$ is an FSM that can be characterized by the language $\mathcal{L}(\mathcal{C}/G) \subseteq \mathcal{L}(G)$, recursively defined as follows:

1. $\varepsilon \in \mathcal{L}(\mathcal{C}/G)$
2. $s \in \mathcal{L}(\mathcal{C}/G) \ \wedge \ s \cdot \sigma \in \mathcal{L}(G) \ \wedge \ \sigma \in \mathcal{C}(s) \Rightarrow s \cdot \sigma \in \mathcal{L}(\mathcal{C}/G)$

The marked language of $\mathcal{C}/G$ is defined by $\mathcal{L}_m(\mathcal{C}/G) = \mathcal{L}(\mathcal{C}/G) \cap \mathcal{L}_m(G)$.

Not all supervisors are admissible. In particular, a supervisor should not disable uncontrollable events that occur in the plant [Ramadge and Wonham, 1989]. This is formalized by the next definition:

Definition 5 (Admissible Supervisor, Controlled System) *A supervisor $\mathcal{C}$ is* admissible *for a plant G whenever $\mathcal{L}(\mathcal{C}/G) \cdot \Sigma_{uc} \cap \mathcal{L}(G) \subseteq \mathcal{L}(\mathcal{C}/G)$. In this case, the controlled system $\mathcal{C}/G$ is said to be* controllable *with respect to G and Σ_{uc}.*

Supervisory Control Problem ([Ramadge and Wonham, 1989]). The classical supervisory control problem is: *Given a plant G and an FSM S (called the* specification*), build an admissible supervisor $\mathcal{C}$ such that (1): $\mathcal{L}_m(\mathcal{C}/G) \subseteq \mathcal{L}_m(S)$, (2): $\mathcal{C}/G$ is non-blocking, i.e. $\overline{\mathcal{L}_m(\mathcal{C}/G)} = \mathcal{L}(\mathcal{C}/G)$ and (3) $\mathcal{C}$ is the most permissive solution, i.e., for all admissible supervisor $\mathcal{C}'$ satisfying conditions (1) and (2), $\mathcal{L}_m(\mathcal{C}'/G) \subseteq \mathcal{L}_m(\mathcal{C}/G)$ holds.*
In the sequel, we are more interested in the computation of $\mathcal{C}/G$ rather than in the computation of the supervisor $\mathcal{C}$ itself, since one can easily extract $\mathcal{C}$ from $\mathcal{C}/G$.

Given a plant G and a specification S, the standard algorithm SupCont to compute $\mathcal{C}/G$ iterates until stabilization the two following operations, starting from the parallel composition $G \parallel S$: (1) computing the sub-machine Trim restricted to states that are reachable from the initial states and co-reachable from the final states (2) removing the states that violate controllability.

Algorithm SupCont $[G = (\Sigma, \mathcal{X}^{\mathrm{G}}, x_o^{\mathrm{G}}, \delta^{\mathrm{G}}),\ \mathcal{X}_m^{\mathrm{G}} \subseteq \mathcal{X}^{\mathrm{G}},\ S = (\Sigma, \mathcal{X}^{\mathrm{S}}, x_0^{\mathrm{S}}, \delta^{\mathrm{S}}), \Sigma_{uc}] \rightarrow \mathcal{C}/G$

(i) Let $i = 0$ and $G^i = G \parallel S = (\Sigma, \mathcal{X}^i, x_0^i, \delta^i)$, with $\mathcal{X}_{\mathrm{m}}^i = \mathcal{X}_{\mathrm{m}}^{\mathrm{G}} \times \mathcal{X}^{\mathrm{S}}$

(ii) $G'^i = (\Sigma, \mathcal{X}'_i, x_0^i, \delta'^i) = \textsc{Trim}(G^i) = G^i_{\downarrow reach(x_o^i) \cap coreach(\mathcal{X}_m^i)}$

(iii) $G^{i+1} = \textsc{Controllable}(G'^i, G, \Sigma_{uc})$. i.e.

(a) Computing the *forbidden states*:

$$F_s^i = \{x = (x^{\mathrm{G}}, x^{\mathrm{S}}) \in \mathcal{X}'^i \mid \exists \sigma \in \Sigma_{uc},\ \delta^{\mathrm{G}}(\sigma, x^{\mathrm{G}})! \text{ and } \neg(\delta'^i(\sigma, x)!)\ \}$$

(b) Computing the *weakly forbidden states*: $W_s^i = coreach_{\Sigma_{uc}}(F_s^i)$

(c) $G^{i+1} = G'^i_{\downarrow \mathcal{X}'_i \setminus W_s^i}$, the sub-machine of G'^i restricted to states of $\mathcal{X}'^i \setminus W_s^i$

(iv) If $G^{i+1} \neq G'^{i}$ and $G^{i+1} \neq \emptyset$: $i = i + 1$, Goto (ii)

(v) $\mathcal{C}/G = G'^{i}$

It can be shown (see e.g. [Wonham, 2002]) that $\mathcal{C}/G$ is the most permissive solution solving the Supervisory Control Problem.

2.3 Conformance Testing

This section presents the model of IOFSM and the **ioco** conformance relation between them. These notions are used in conformance testing, where the goal is to establish whether a plant G conforms to its formal specification S under the given conformance relation.

2.3.1 Input/Output Finite State Machines

In our framework, the plant and the specification are modeled by Input/Output Finite State Machines (IOFSM). Interactions between the system and its environment are modeled by input and output events, and the internal behavior of the system is modeled by internal events.

Definition 6 (IOFSM) *An IOFSM is an FSM $G = (\Sigma, \mathcal{X}, x_o, \delta)$ whose alphabet Σ is partitioned into three subsets $\Sigma_?$, $\Sigma_!$ and Σ_I of inputs, outputs, and internal events.*

Notations. Given an IOFSM G, we denote by $\Sigma_{?!}$ the set of observable events (i.e. the set of input and output events). For $s \in \mathcal{L}(G)$, we denote $Trace(s) = P_{\Sigma \to \Sigma_{!?}}(s)$ its behavior observable by the environment and $Traces(G) = P_{\Sigma \to \Sigma_{?!}}(\mathcal{L}(G))$.

For $s \in Traces(G)$, and $x \in \mathcal{X}$, x *after* s denotes the set of states in which the system may be after observing the sequence s from state x, i.e.,

$$x \textit{ after } s = \{x' \mid \exists s', s' \in \mathcal{L}(G) \wedge Trace(s') = s \wedge \delta(s', x) = x'\}.$$

We extend this notation to a set of states as follows: $\mathcal{X}'$ *after* $s = \bigcup_{x \in \mathcal{X}'} (x \textit{ after } s)$ and to an IOFSM G by G *after* $s = x_o$ *after* s where x_o is the initial state of G.

As usual for finite-state machines, for an IOFSM G, we can construct a deterministic IOFSM $Det(G)$ without internal events, and with same traces as G, i.e. $Traces(G) = Traces(Det(G)) = \mathcal{L}(Det(G))$.

Definition 7 (Determinization) *Given an IOFSM $G = (\Sigma, \mathcal{X}, x_o, \delta)$, with $\Sigma = \Sigma_? \cup \Sigma_I \cup \Sigma_!$, the* determinization *of G, called $Det(G)$, is defined by $Det(G) = (\Sigma_{?!}, 2^{\mathcal{X}}, x_o \textit{ after } \varepsilon, \delta^d)$ where for $\mathcal{X}_1, \mathcal{X}_2 \in 2^{\mathcal{X}}$ and $\sigma \in \Sigma_{?!}$, $\delta^d(\sigma, \mathcal{X}_1) = \mathcal{X}_2$ iff $\mathcal{X}_2 = \mathcal{X}_1$ after σ.*

If G is equipped with a set of marked states $\mathcal{X}_m$, the set of marked states of $Det(G)$ is $\{F \in 2^{\mathcal{X}} \mid F \cap X_m \neq \emptyset\}$.

In an IOFSM, it is useful to assume that every input event is always enabled:

Definition 8 (input-complete IOFSM) *An IOFSM G with alphabet $\Sigma = \Sigma_? \cup \Sigma_I \cup \Sigma_!$ is input-complete if in each state x, every input event is enabled (possibly after a sequence of internal events), i.e. $\forall x \in X, \Sigma_? \subseteq \delta(x \text{ after } \varepsilon)$.*

It can be shown that if G is input-complete, $Traces(G) \cdot \Sigma_? \subseteq Traces(G)$.

2.3.2 Blocking

We now refine the notion of blocking introduced in Section 2.2 in order to adapt it to the model of IOFSM. This notion occurs in the conformance relation (see Definition 10 below). In the supervisory control theory briefly described in Section 2.2, blocking is defined negatively as "being unable to complete a task" (to reach a marked state [Cassandras and Lafortune, 1999]). This definition is not suitable for conformance testing, where blocking is not always a bad thing. A blocking of the plant is allowed if the specification allows it, and a blocking in the specification may model a reactive system that terminates its execution, or that is blocked waiting for an input from the environment, or that performs an infinite loop of internal computations.

- A *deadlock* state is a state where the system cannot evolve (i.e. $\delta(x) = \emptyset$).
- An *output-lock* state is a state where the system can only move by inputs ($\delta(x) \subseteq \Sigma_?$). Note that this includes deadlocks ($\delta(x) = \emptyset$)
- *a livelock* state is a state from which the system may execute an infinite sequence of internal events. In our finite-state framework, this may only happen if there exists a cycle of states and transitions labeled by internal events (i.e. $\exists \tau_1 \cdot \tau_2 \cdots \tau_n \in \Sigma_I^+ . \delta(\tau_1 \cdots \tau_n, x) = x$). In this case, the states are said to *belong* to the livelock.

For an IOFSM G, we denote by $livelocks(G)$, the set of states that belong to a livelock and $outputlocks(G)$, the set of output-lock/deadlock states, and define $blockings(G) = livelocks(G) \cup outputlocks(G)$ the set of states in which a blocking may occur (cf. left-hand side of figure 1).
In order to distinguish valid blockings (those present in the specification), from invalid ones, blockings must be materialized by adding a new output event λ that manifests itself in blocking states (cf. right-hand side of figure 1).

Definition 9 (Suspension IOFSM) *For an IOFSM $G = (\Sigma, \mathcal{X}, x_o, \delta)$, the* suspension *of G is the IOFSM $\lambda(G) = (\Sigma^\lambda, \mathcal{X}, x_o, \delta_\lambda)$, where $\Sigma^\lambda = \Sigma_? \cup \Sigma_I \cup \Sigma_!^\lambda$, and $\Sigma_!^\lambda = \Sigma_! \cup \{\lambda\}$ (λ is a new output, observable by the environment). The transition relation δ_λ is obtained from δ by adding a self-loop labeled λ on each blocking state, i.e. $\forall x \in blocking(G),\ \delta_\lambda(\lambda, x) = x$ and $\forall x \in \mathcal{X},\ \forall \sigma \neq \lambda,\ \delta(\sigma, x) = \delta_\lambda(\sigma, x)$ otherwise.*

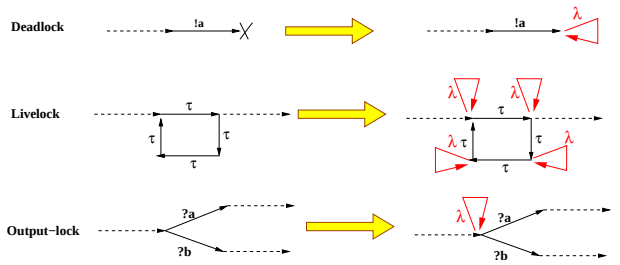

Figure 1: Blockings, and how to materialize them.

As $\lambda \notin \Sigma$, by definition of traces and properties of projections, we get

$$\mathcal{L}(G) = P_{\Sigma^{\lambda} \to \Sigma}(\mathcal{L}(\lambda(G))), \text{ and } Traces(G) = P_{\Sigma^{\lambda}_{!?} \to \Sigma_{!?}}(Traces(\lambda(G)))$$

Given an IOFSM H with alphabet Σ_H, we denote by $\lambda^{-1}(H)$ the sub-machine of H restricted to the alphabet $\Sigma_H \setminus \{\lambda\}$, i.e. $H_{\downarrow \Sigma_H \setminus \{\lambda\}}$. We have $\mathcal{L}(\lambda^{-1}(H)) = \mathcal{L}(H) \cap (\Sigma_H \setminus \{\lambda\})^*$. If $\lambda \notin \Sigma_H$, we get $\lambda^{-1}(\lambda(H)) = H$.

2.3.3 Conformance Relation

A conformance relation defines the set of plants of a system that behave correctly with respect to a given specification. The **ioco** relation defined by [Tretmans, 1996] is the one most employed in practice. Intuitively, a plant G **ioco**-conforms to its specification S if after each observable trace of $\lambda(S)$, G only exhibits outputs and blockings that are also allowed by S.

Definition 10 *Let G the plant and S its expected specification be two IOFSM having the same sets of observable events $\Sigma_{!?}$. Then*

$$G \textbf{ ioco } S \equiv [\forall s \in Traces(\lambda(S)), \quad Out(\lambda(G) \textit{ after } s) \subseteq Out(\lambda(S) \textit{ after } s)] \qquad (4)$$

where, for $\mathcal{X}' \subseteq \mathcal{X}$, the set $Out(\mathcal{X}') = \Sigma_! \cap \bigcup_{x \in \mathcal{X}'} \delta(x)$ is the set of all outputs enabled in some state of $\mathcal{X}'$.

Figure 2 illustrates this notion. G_1 **ioco** S holds because in each state, outputs of G_1 are included in outputs of S. Notice that the initial state of G_1 allows an *input*[2]: $?b$, which is not present in S, but this does not matter because only *outputs* are constrained by **ioco**. Thus, the **ioco** relation tolerates that the specification does not contain information about all inputs, i.e., it allows for incomplete specifications. However $\neg(G_2$ **ioco** $S)$ holds, as the output $!z$ after the input $?a$ is not allowed in the specification, and the blocking (self-loop labeled λ) after $?a{\cdot}!x$ is not specified in S.

In the sequel, we use an alternative definition of **ioco**, given by the following property:

[2]Here, a "?" (resp. a "!") preceding the name of an event means that the event is an input (resp. an output).

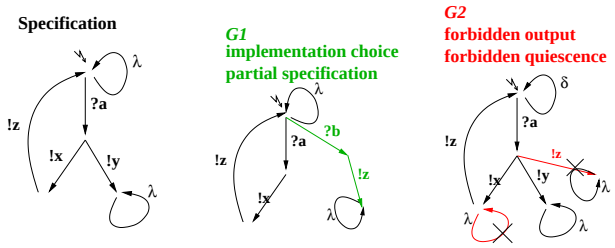

Figure 2: **ioco** by the example

Proposition 1 G **ioco** $S \Leftrightarrow Traces(\lambda(S)).\Sigma_!^\lambda \cap Traces(\lambda(G)) \subseteq Traces(\lambda(S))$.
Proof :

$$\begin{array}{rcl} G \textbf{ ioco } S & \Leftrightarrow & \forall s \in Traces(\lambda(S)), Out(\lambda(G) \textit{ after } s) \subseteq Out(\lambda(S) \textit{ after } s) \\ & \Leftrightarrow & \forall s \in Traces(\lambda(S)),\ \forall \sigma \in \Sigma_!^\lambda, s\sigma \in Traces(\lambda(G)) \Rightarrow s\sigma \in Traces(\lambda(S)) \\ & \Leftrightarrow & Traces(\lambda(S)).\Sigma_!^\lambda \cap Traces(\lambda(G)) \subseteq Traces(\lambda(S)) \end{array}$$

2.4 Reducing ioco to a trace inclusion problem

We now give an alternative, simpler definition of the **ioco** conformance relation that holds if the specification is input-complete. Then we show how to treat the general case, by transforming an arbitrary specification into an input-complete one with the same set of conformant implementations.

Proposition 2 (ioco for input-complete specifications) *Let G, the plant, and S, its expected specification, be two IOFSMs over the same visible alphabet, such that S is input-complete (cf. Definition 8). Then,* G **ioco** $S \Leftrightarrow Traces(\lambda(G)) \subseteq Traces(\lambda(S))$

Proof :($\Rightarrow$) We have to prove that $Traces(\lambda(G)) \not\subseteq Traces(\lambda(S)) \Rightarrow \neg(G \textbf{ ioco } S)$. If $Traces(\lambda(G)) \not\subseteq Traces(\lambda(S))$, there exists $s \in Traces(\lambda(G))$ such that $s \notin Traces(\lambda(S))$. Then s can be decomposed into $s = s' \cdot \sigma \cdot s''$ where s' is the longest prefix of s in $Traces(\lambda(S))$, $\sigma \in \Sigma_{!?}^\lambda$ and $s'' \in \Sigma_{?!}^{\lambda *}$. As $s \in Traces(\lambda(G))$, which is prefix-closed, we also have $s' \cdot \sigma \in Traces(\lambda(G))$.

If $\sigma \in \Sigma_?$, as S is input-complete, $s' \cdot \sigma \in Traces(\lambda(S))$ which contradicts the maximality of s'. Thus $\sigma \in \Sigma_!^\lambda$. Then we have $s' \cdot \sigma \in Traces(\lambda(S)).\Sigma_{?!}^\lambda \cap Traces(\lambda(G))$ but $s' \cdot \sigma \notin Traces(\lambda(S))$, which, according to Prop. 1, proves that $\neg(G \textbf{ ioco } S)$.
($\Leftarrow$) is trivial, even if S is not input-complete. In fact $Traces(\lambda(G)) \subseteq Traces(\lambda(S))$ implies $Traces(\lambda(G)) \cap Traces(\lambda(S)).\Sigma_!^\lambda \subseteq Traces(\lambda(S))$. Hence G **ioco** S by Prop. 1.

$\diamond$

Unfortunately, not all specifications are input-complete. But for any specification S, it is possible to build an input-complete one, denoted $Comp(S)$, without modifying

the set of plants G that conform to S for **ioco**. The idea is first to build $Det(\lambda(S))$, i.e., the determinization of the suspension IOFSM of S, and then to add a new state denoted $Comp$, together with transitions from each state x of $Det(\lambda(S))$ to $Comp$, labeled by all the inputs that do not label any other transition from x to some other state of $Det(\lambda(S))$.

Due to space limitations we do not give the effective construction of $Comp(S)$, which can be found in [Jard et al., 1999]. We just give its properties in terms of suspension traces.

Proposition 3 (input completion) *For an IOFSM $S = (\Sigma, \mathcal{X}, x_o, \delta)$ it is possible to build an input-complete IOFSM $Comp(S)$ such that:*

$$Traces(\lambda(Comp(S))) = Traces(\lambda(S)) \cup [Traces(\lambda(S)).\Sigma_? \setminus Traces(\lambda(S))].(\Sigma^{\lambda}_{!?})^*$$

Using Prop. 3, the following equivalence holds:

Proposition 4 *Let G and S be two IOFSM, then : G **ioco** $S \iff G$ **ioco** $Comp(S)$*

Proof :
($\Leftarrow$) First we prove the property (I): $Traces(\lambda(S)) \cdot \Sigma^{\lambda}_! \cap Traces(\lambda(Comp(S))) \subseteq Traces(\lambda(S))$. Using Prop. 3, we obtain $Traces(\lambda(S)) \cdot \Sigma^{\lambda}_! \cap Traces(\lambda(Comp(S))) = Traces(\lambda(S)) \cdot \Sigma^{\lambda}_! \cap [Traces(\lambda(S)) \cup [Traces(\lambda(S)).\Sigma_? \setminus Traces(\lambda(S))].(\Sigma^{\lambda}_{!?})^*]$, and, by distributing $\cap$ under $\cup$, we get $[Traces(\lambda(S)) \cdot \Sigma^{\lambda}_! \cap Traces(\lambda(S))] \cup [Traces(\lambda(S)) \cdot \Sigma^{\lambda}_! \cap [Traces(\lambda(S)).\Sigma_? \setminus Traces(\lambda(S))].(\Sigma^{\lambda}_{!?})^*]]$.

Now, the second intersection is empty, as $s \in Traces(\lambda(S)) \cdot \Sigma^{\lambda}_!$ implies that all strict prefixes of s are in $Traces(\lambda(S))$, but none of the prefixes of sequences in $[Traces(\lambda(S)).\Sigma_? \setminus Traces(\lambda(S))].(\Sigma^{\lambda}_{!?})^*]$ is in $Traces(\lambda(S))$. Hence, the above property (I) holds.

Assume now that $\neg(G$ **ioco** $S)$.

By Prop. 1, there exists $s \in Traces(\lambda(S))$ and $\sigma \in \Sigma_!$ such that $s \cdot \sigma \in Traces(\lambda(G))$, but $s \cdot \sigma \notin Traces(\lambda(S))$. By Prop. 3 we get $Traces(\lambda(S)) \subseteq Traces(\lambda(Comp(S)))$. Hence (II):$s \in Traces(\lambda(Comp(S)))$.

Thus, if we had $s \cdot \sigma \in Traces(\lambda(Comp(S)))$, as (II) holds, $s \in Traces(\lambda(S))$ and $\sigma \in \Sigma^{\lambda}_!$, (I) implies $s \cdot \sigma \in Traces(\lambda(S))$, which contradicts the hypothesis. Hence, there exists $s \in Traces(\lambda(Comp(S)))$ such that $s \cdot \sigma \in Traces(\lambda(G))$ and $s \cdot \sigma \notin Traces(\lambda(Comp(S)))$, with $\sigma \in \Sigma^{\lambda}_!$ which exactly means $\neg(G$ **ioco** $Comp(S))$.
($\Rightarrow$) Assuming G **ioco** S we need to prove G **ioco** $Comp(S)$ which, by Prop. 1, is

$$(i) : Traces(\lambda(Comp(S)).\Sigma^{\lambda}_! \cap Traces(\lambda(G)) \subseteq Traces(\lambda(Comp(S)))$$

Using Prop. 3 and distribution of concatenation, the left member of the inclusion (i) can be rewritten

$$Traces(\lambda(S)) \cdot \Sigma^{\lambda}_! \cup [Traces(\lambda(S)) \cdot \Sigma_? \setminus Traces(\lambda(S))] \cdot (\Sigma^{\lambda}_{!?})^* \cdot \Sigma^{\lambda}_! \cap Traces(\lambda(G))$$

Using distribution of $\cap$ on $\cup$, this can be rewritten in $A \cup B$, with
$A = Traces(\lambda(S)) \cdot \Sigma_!^\lambda \cap\ Traces(\lambda(G))$, and
$B = [Traces(\lambda(S)) \cdot \Sigma_?\ \setminus\ Traces(\lambda(S))] \cdot (\Sigma_{!?}^\lambda)^* \cdot \Sigma_!^\lambda\ \cap\ Traces(\lambda(G))$.
We first easily get $(\alpha) : A \subseteq Traces(\lambda(S))$. Now in B if we notice $(\Sigma_{!?}^\lambda)^* \cdot \Sigma_!^\lambda \subseteq (\Sigma_{!?}^\lambda)^*$ and eliminate $Traces(\lambda(G))$, we get $(\beta) : B \subseteq [Traces(\lambda(S)) \cdot \Sigma_?\ \setminus\ Traces(\lambda(S))] \cdot (\Sigma_{!?}^\lambda)^*$. Finally inclusions (α) and (β) together give:
$A \cup B \subseteq Traces(\lambda(S)) \cup [Traces(\lambda(S)) \cdot \Sigma_?\ \setminus\ Traces(\lambda(S))] \cdot (\Sigma_{!?}^\lambda)^*$ which equals $Traces(\lambda(Comp(S)))$ by Prop. 3. This ends the proof of inclusion (i) and of the whole proposition. $\diamond$

Prop. 4 allows us to assume that the specification S is input complete, because if it is not, we can replace it with $Comp(S)$ without changing the set of conformant plants.

3 The Supervisory Control Problem for Conformance

Given a plant modeled as an IOFSM G and a specification S also modeled as an IOFSM over the same alphabet, the problem considered here is to control the plant by means of a supervisor $\mathcal{C}$ such that the controlled plant satisfies $\mathcal{C}/G$ **ioco** S. Using the results established in paragraph 2.4 we assume that the expected specification is input-complete. We start by properly defining the control of IOFSM and the supervisory control problem for conformance, called the **ioco-control** problem. We then solve the ioco-control problem in two steps: first, we build a plant that is "almost" conformant with its specification, except for some possibly undesired livelocks, deadlocks, and output locks. Then, the undesired blockings are removed by control. A running example will illustrate the different steps.

3.1 The control of IOFSM

The IOFSM model makes a distinction between inputs, outputs and internal events. In order to perform control on a plant modeled by an IOFSM, we also have to partition its alphabet into controllable and uncontrollable events. Notice that the goal is to control the plant, not the environment. Hence, inputs (from the environment to the plant) are uncontrollable. In contrast, some outputs (as well as all internal events of the plant) are controllable, because the supervisor, by interacting with the plant, may prevent them from occurring[3]. This gives the definition of *IOFSM under control*:

Definition 11 (IOFSM under control) *An* IOFSM under control *is an IOFSM* $(\Sigma, \mathcal{X}, x_o, \delta)$ *whose alphabet* $\Sigma = \Sigma_! \cup \Sigma_? \cup \Sigma_I$ *is partitionned into two sets* Σ_c, Σ_{uc}

[3]The output event λ of a suspension IOFSM (cf. Definition 9) is assumed to be uncontrollable.

satisfying $\Sigma_? \subseteq \Sigma_{uc}$ *(all inputs are uncontrollable) and* $\Sigma_I \subseteq \Sigma_c$ *(all internal events and some outputs are controllable).*

As illustrated in figure 3, in the remainder of this paper we assume that all the events of the plant, including internal events, are observable by the supervisor. The reason is that we assume that the supervisor is a component added "inside" the plant, where everything is observable. Now, the plant together with the supervisor are seen as a "black box" by the environment. We want to ensure, in the least constraining way, that this supervised plant conforms to its specification S for the **ioco** relation. This relation only deals with what is observable by the environment i.e., *traces* of inputs and outputs. This leads to the following formulation of the **ioco-control** problem:

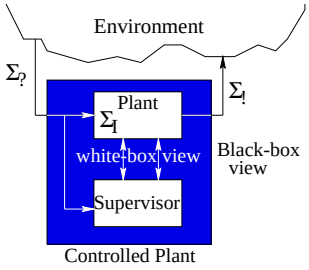

Figure 3: Environment view vs. Supervisor view

Definition 12 (ioco-control) *Given two IOFSM: The plant* G *and the specification* S*, synthesize an admissible supervisor* $\mathcal{C}$ *such that 1)* $\mathcal{C}/G$ **ioco** S*, and 2)* $\mathcal{C}$ *is the most permissive supervisor, i.e., for all admissible supervisors* $\mathcal{C}'$*,* $\mathcal{C}'/G$ **ioco** S *implies* $Traces(\mathcal{C}'/G) \subseteq Traces(\mathcal{C}/G)$*.*

The main difference between **ioco**-control and the classical supervisory control problem (presented in Section 2.2) arises from the notion of blocking. Here, a blocking of the plant will be eliminated by the supervisor only if it is not allowed by the specification. Another difference is that the traces of the resulting controlled plant are not necessarily included in those of S. This allows to synthesize controllers when only a partial specification of the plant is available. Finally, unlike the classical SCP, the notion of maximal permissiveness is defined by trace inclusion (not language inclusion).

In the remainder of this paper, for control purposes, we assume that the specification is given by an input-complete and deterministic IOLTS S^λ over the alphabet $\Sigma_? \cup \Sigma_!^\lambda$; i.e. the expected specification does not model the internal behavior of the plant. However, we assume that the allowed blockings are already taken into account (there is no need to compute the suspension IO-FSM) Moreover, we assume that $\lambda(\lambda^{-1}(S^\lambda)) \subseteq S^\lambda$. This simply means that all the blocking of the specification are taken into account plus some others that may model internal livelocks.

3.2 Reducing ioco-control to the avoidance of blockings

Now, given a plant G and a specification S^λ, we first constrain the observable behaviors of the plant (except for blockings) to remain in those allowed by the specification. To do so, we consider the composition $G_o = G \parallel S^\lambda$. The result is an IOFSM that accepts all the traces of G allowed by S^λ, while preserving the internal behavior of G after a given observable trace. Moreover, G_o captures all the blockings that are admissible in S^λ and that may happen in G. This is summarized by the following result.

Proposition 5 *Let $G_o = G \parallel S^\lambda$, then*

$$\mathcal{L}(\lambda^{-1}(G_o)) \subseteq \mathcal{L}(G) \text{ and } Traces(\lambda^{-1}(G_o)) \subseteq \mathcal{L}(\lambda^{-1}(S^\lambda))$$

Proof :

$$\begin{aligned}
\mathcal{L}(\lambda^{-1}(G_o)) &= \mathcal{L}(\lambda^{-1}(G||S^\lambda))) = \mathcal{L}(G||S^\lambda)) \cap \Sigma^* \text{ using properties of } \lambda^{-1} \\
&= P^{-1}_{\Sigma^\lambda \to \Sigma}(\mathcal{L}(G)) \cap P^{-1}_{\Sigma^\lambda \to \Sigma^\lambda_{!?}}(\mathcal{L}(S^\lambda)) \cap \Sigma^* \text{ using properties of } || \\
&= \mathcal{L}(G) \cap [P^{-1}_{\Sigma^\lambda \to \Sigma^\lambda_{!?}}(\mathcal{L}(S^\lambda)) \cap \Sigma^*] \quad (\alpha), \text{as } \mathcal{L}(G) = P^{-1}_{\Sigma^\lambda \to \Sigma}(\mathcal{L}(G)) \cap \Sigma^*
\end{aligned}$$

This proves $\mathcal{L}(\lambda^{-1}(G_o)) \subseteq \mathcal{L}(G)$. We also have $Traces(\lambda^{-1}(G_o)) \subseteq \mathcal{L}(\lambda^{-1}(S^\lambda))$ as

$$\begin{aligned}
Traces(\lambda^{-1}(G_o)) &= P_{\Sigma^\lambda \to \Sigma^\lambda_{!?}}(\mathcal{L}(\lambda^{-1}(G_o))) \\
&\subseteq P_{\Sigma^\lambda \to \Sigma^\lambda_{!?}}(P^{-1}_{\Sigma^\lambda \to \Sigma^\lambda_{!?}}(\mathcal{L}(S^\lambda) \cap \Sigma^*), \text{ using } (\alpha) \\
&\subseteq \mathcal{L}(S^\lambda) \cap P_{\Sigma^\lambda \to \Sigma^\lambda_{!?}}(\Sigma^*) = \mathcal{L}(\lambda^{-1}(S^\lambda)) \qquad \diamond
\end{aligned}$$

Knowing that S^λ is input-complete, the fact that $Traces(\lambda^{-1}(G_o)) \subseteq \mathcal{L}(\lambda^{-1}(S^\lambda))$ ensures that, except for blockings (that we have modeled by the λ event), G_o conforms to S^λ (Proposition 2). Moreover, we have the following result, which says that G_o is a good starting point for solving the **ioco-control** problem, in the sense that G_o is larger than any solution of the problem:

Lemma 1 *Let G^{ioco} be an IOFSM ensuring the conformance w.r.t. S^λ such that $\mathcal{L}(G^{ioco}) \subseteq \mathcal{L}(G)$, and let $G_o = G \parallel S^\lambda$. Then, $\mathcal{L}(\lambda(G^{ioco})) \subseteq \mathcal{L}(G_o)$.*

Proof : Let $\Lambda(G)$ be the IOFSM obtained by adding a λ self-loop on each state of G. By Definition of $\parallel$, we have $G \parallel S^\lambda = \Lambda(G) \parallel S^\lambda$ and

$$\mathcal{L}(G \parallel S^\lambda) = \mathcal{L}(\Lambda(G) \parallel S^\lambda) = \mathcal{L}(\Lambda(G)) \ \cap P^{-1}_{\Sigma^\lambda \to \Sigma^\lambda_{?!}}(\mathcal{L}(S^\lambda)) \qquad (5)$$

Moreover, G^{ioco} **ioco** S^{λ}, which, according to Prop. 4, implies that

$$Traces(\lambda(G^{ioco})) \subseteq Traces(S^{\lambda}) = \mathcal{L}(S^{\lambda})$$

Let $s \in \mathcal{L}(\lambda(G^{ioco}))$, then, $P_{\Sigma^{\lambda} \to \Sigma^{\lambda}_{!?}}(s) \in Traces(\lambda(G^{ioco})) \subseteq \mathcal{L}(S^{\lambda})$. This implies that $s \in P^{-1}_{\Sigma^{\lambda} \to \Sigma^{\lambda}_{?!}}(\mathcal{L}(S^{\lambda}))$. Now, as $\mathcal{L}(G^{ioco}) \subseteq \mathcal{L}(G)$, $\mathcal{L}(\lambda(G^{ioco})) \subseteq \mathcal{L}(\Lambda(G))$. We then have $s \in \mathcal{L}(\Lambda(G)) \cap P^{-1}_{\Sigma^{\lambda} \to \Sigma_{?!}}(\mathcal{L}(S^{\lambda}))$. And finally based on (5), $\mathcal{L}(\lambda(G^{ioco}) \subseteq \mathcal{L}(G \parallel S^{\lambda})$ ◇

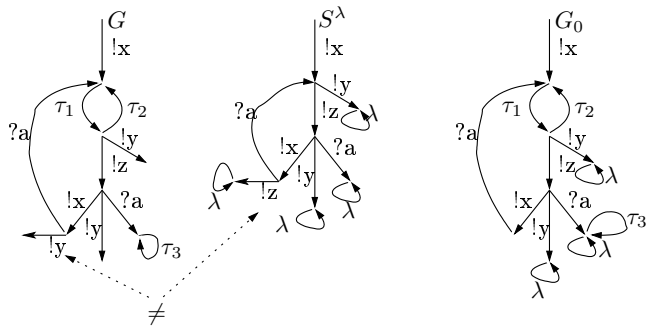

Figure 4: First step: plant and specification composition

Example *Consider the plant G ($\Sigma_{uc} = \{?a, !x\}$) and the specification S^{λ} given in figure 4. G_0 is the result of their composition. As the event $!y$ after $!x.\tau_1.!z.!x$, present in G, is not allowed by S^{λ}, it has been cut by the composition operation. (Here, S^{λ} is not input-complete, but input-completeness does not matter in this example.)*

At this point, the only remaining non-conformances of G_o with respect to S^{λ} are blockings. The states of G_o with a λ self-loop correspond to blockings of G that are also allowed by the specification; these states do not pose problems, and may be kept by the supervisor in the final, controlled plant. However, some other blocking states may still exist in G_o. They correspond to blockings of the plant that do not exist in the specification and have to be removed. This is done in two phases:

1) First, the livelocks of G_o that are not labeled by a λ self-loop are removed (Section 3.3).

2) In a second phase, we eliminate the output-locks as well as the deadlocks (Section 3.4).

3.3 Eliminating undesired livelocks

Let $G_o = G \parallel S^{\lambda}$ the IOFSM computed after the previous phase. We denote by $\mathcal{G}_o = \{G^i | i = 1, 2, \ldots\}$ the finite set of maximal submachines of the machine G_o, whose alphabets are all equal to Σ_I, which include at least one cycle, and such that from each state of G^i, there is no transition labeled λ in G_o. Formally, if $G_o = (\Sigma \cup \{\lambda\}, \mathcal{X}_o, x_o, \delta_o)$, let

$$I_o = \{x_o^i \in \mathcal{X} \quad | \quad (x_o^i = x_o \vee \delta^{-1}(x_o^i) \cap \Sigma_{?!} \neq \emptyset) \\ \wedge \forall x \in reach_{\Sigma_I}(x_o^i), \neg\delta_o(\lambda, x) = x \\ \wedge \exists x \in reach_{\Sigma_I}(x_o^i), \exists s_i \in \Sigma_I^+.\ \delta_o(s_i, x) = x\} \tag{6}$$

Then,

$$\mathcal{G}_o = \{G^i = (\Sigma_I, \mathcal{X}^i, x_o^i, \delta^i) | x_o^i \in I_o\} \tag{7}$$

such that for all $x_o^i \in I_o$,

$$G^i = G_{o\downarrow\Sigma_I, reach_{\Sigma_I}(x_o^i)} \tag{8}$$

The set of marked states of each of these submachines is respectively defined by

$$\mathcal{X}_m^i = \{x \in \mathcal{X}^i | \exists x' \in \mathcal{X}_o, \exists \sigma \in \Sigma \setminus \Sigma_I . \delta_o(x, \sigma)!\} \tag{9}$$

that is, the set of states from which there exists a visible event whose firing exits the machine G_i.

These submachines are interesting because a cycle in any of them corresponds to a livelock in the plant G_o that is not present in the specification S^λ (a livelock present in G_o and accepted by S^λ would have been indicated by a λ self-loop). These livelocks violate conformance and have to be removed.

Let $Acyclic(G^i)$ be the acyclic graph containing all the acyclic elementary paths of G_i, and $\mathcal{C}_l/G^i = \textsc{SupCont}(G^i, \mathcal{X}_m^i, Acyclic(G^i))$ the result obtained by applying the SupCont algorithm, defined in Section 2.2, using this graph as a control objective, and $\mathcal{X}_m^i$ as marked states.

It is worthwhile noting that the way we are removing the livelocks by simply cutting the cycles of internal events makes that the obtained supervisor is not the most permissive one (according to the language inclusion relation). It would be sufficient to unfold the original graph and to apply our techniques on this new graph in order to obtain a more permissive controller. However, the next lemma ensures us that it is maximal if we express the maximality w.r.t. the traces inclusion relation used by the **ioco** relation (cf. Def. 10). Indeed one important aspect of our algorithm is that it preserves the reachability of all the final states that basically corresponds to the places where an observation can be performed. Hence by first unfolding the graph and by controlling the unfolded graph, the same final states would be reachable and therefore the same set of observable events would be admissible.

Lemma 2 *With the above notations, $\mathcal{C}_l/G^i$ has the property that all states in $\mathcal{X}_m^i$ are reachable from the respective initial state x_o^i.*

Proof : (Sketch) We have assumed that all the events in Σ_I are controllable (cf. Definition 11). Then, note that the computation of $\mathcal{C}_l/G^i$ here consists essentially in

"breaking" loops. Since all transitions are labeled by internal events, and all internal events are controllable, a loop can be "broken" by preventing an arbitrary event from that loop to occur. Hence, the controlled system $\mathcal{C}_l/G^i$ contains all the sequences in $Acyclic(G^i)$. Now, a state is reachable if it is reachable by an acyclic sequence. In particular, all states in $\mathcal{X}_m^i$ are reachable in $\mathcal{C}_l/G^i$ from the initial state x_o^i. $\diamond$

Finally, let $G_o(G^i \rightarrow \mathcal{C}_l/G^i)$ denote the operation that consists in replacing in G_o, the submachine G^i by the controlled system $\mathcal{C}_l/G^i$, and $\bigcup_{G^i \in \mathcal{G}_o} G_o(G^i \rightarrow \mathcal{C}_l/G^i)$ denote the union of all these machines, understood as the point-wise union of 4-tuples of sets of the form $(\Sigma, \mathcal{X}, x_o, \delta)$, which share the same initial state x_o. The algorithm that removes undesired livelocks is the following:

Algorithm LIVE-LOCK-FREE$(G, S^\lambda) \rightarrow \mathcal{C}_l/G_o$

(i) $G_o = G \parallel S^\lambda$, and $\mathcal{G}_o$ is defined by Equations (6), (7), and (8)

(ii) $\forall G^i \in \mathcal{G}_o$, $\mathcal{C}_l/G^i = \text{SUPCONT}(Acyclic(G^i), \mathcal{X}_m^i, G^i)$

(iii) $G'_o = \bigcup_{G^i \in \mathcal{G}_o} G_o(G^i \rightarrow \mathcal{C}_l/G^i)$

(iv) $\mathcal{C}_l/G_o = G'_o$

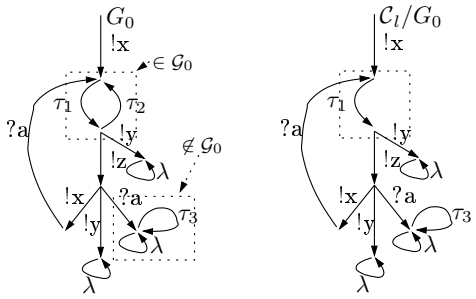

Figure 5: Second step: Removing livelocks

Example *Considering again our example (cf. figure 4), the two internal events cycles are shown in figure 5. Only one of them has no λ transition and thus belongs to $\mathcal{G}_0$. This cycle is broken by cutting τ_2.*

Proposition 6 *Let G^{ioco} be an IOFSM that conforms to S^λ and controllable with respect to G, such that $\mathcal{L}(G^{ioco}) \subseteq \mathcal{L}(G_o)$. Let also $\mathcal{C}_l/G_o$ be the result of Algorithm* LIVE-LOCK-FREE. *Then, $Traces(\lambda(G^{ioco})) \subseteq Traces(\mathcal{C}_l/G_o)$.*

Proof : We only give here the sketch of the proof. Assume that there exists a trace $s \in Traces(\lambda(G^{ioco})) \setminus Traces(\mathcal{C}_l/G_o)$. Since (cf. Lemma 1) $Traces(\lambda(G^{ioco})) \subseteq Traces(G_o)$, we also have $s \in Traces(G_o) \setminus Traces(\mathcal{C}_l/G_o)$. The trace s can be decomposed as $s' \cdot \sigma \cdot s''$, where s' is the longest prefix of s such that $s' \in Traces(\mathcal{C}_l/G_o)$.

Hence, σ is a visible event, i.e., $\sigma \in \Sigma^{\lambda}_{?!}$, and is unreachable in $\mathcal{C}_l/G_o$ as well, that is,

$$(\dagger): \quad s' \cdot \sigma \notin Traces(\mathcal{C}_l/G_o)$$

Then, let $s'^{-1} \in P^{-1}_{\Sigma^{\lambda} \to \Sigma^{\lambda}_{?!}}(s') \cap \mathcal{L}(G_o)$ be a sequence of G_o whose projection on visible events is s'. Thus, σ is a visible event that was fireable in G_o after the sequence s'^{-1}, but has become unreachable in $\mathcal{C}_l/G_o$ after the sequence s'^{-1}, by effect of the supervisor $\mathcal{C}_l$.

Let now $s'_\tau : \tau_1 \cdot \tau_2 \ldots \tau_k$ be the longest suffix of s'^{-1} that consists only of internal events, and $t'_\tau : t_1 \cdot t_2 \ldots t_k$ be the corresponding sequence of transitions. By construction, the only way for σ to become unreachable in $\mathcal{C}_l/G_o$ is by having the supervisor $\mathcal{C}_l$ eliminate (a suffix of) t'_τ from G_o.

By construction of the supervisor, this may only happen when t'_τ contains a loop, and is itself contained in some machine G_i from the set $\mathcal{G}_o$ defined by Equations (6), (7), and (8). Then, t'_τ is nonempty, and, by maximality of t'_τ and definition of G_i, the initial state of G_i is the origin of t_1.

Moreover, the origin of the transition labeled by σ must be a *marked* state of G_i, i.e., a state that belongs to the corresponding set $\mathcal{X}^i_m$ defined by Equation (9). Indeed, these are the only states of G_i from which a visible event can be fired.

Since the origin l of σ was reachable in G_o, but has become unreachable in $\mathcal{C}_l/G_i$, this implies that the state $l \in \mathcal{X}^i_m$ has become unreachable due to the event of the supervisor $\mathcal{C}_l$. However, the initial state of G_i is still reachable in $\mathcal{C}_l/G_o$, and, by construction, all the marked states that were reachable in G_i from the initial state of G_i, are still reachable in $\mathcal{C}_l/G_i$, and therefore, are reachable in $\mathcal{C}_l/G_o$. Hence, σ is fireable in $\mathcal{C}_l/G_o$ after s'^{-1}, in contradiction with (†): the proof is done. $\diamond$

3.4 Eliminating undesired deadlocks and output-locks

The controlled plant $\mathcal{C}_l/G$ computed so far is such that its traces and livelocks are among those of the specification S^{λ}. In order to obtain a controlled plant that fully conforms to the specification, we still need to remove from $\mathcal{C}_l/G$ the deadlocks and output-locks that are not present in S^{λ}. This is done by the following operation.

Definition 13 *Let $H = (\Sigma, \mathcal{X}, x_o, \delta)$ be an IOFSM, then* CUT(H) *is the result of the following fix-point computation*

$$\begin{cases} H_o & = & H \\ H_{i+1} & = & H_{i \downarrow \mathcal{X}_i \setminus outputlock(H_i)} \text{ where } \mathcal{X}_i \text{ is the set of states of } H_i \end{cases} \tag{10}$$

Intuitively, the CUT operation removes from an IOFSM the states that correspond either to a deadlock (i.e when $\delta(x) = \emptyset$) or to an output-lock ($\delta(x) \subseteq \Sigma_?$). When applied to the machine $\mathcal{C}_l/G$ that was computed in the previous section, the CUT operation eliminates precisely the deadlocks and output-locks that are not present in the specification S^{λ}. Then, the following result holds:

Proposition 7 *Let* $H = \text{LIVE-LOCK-FREE}(G, S^\lambda)$ *be the IOFSM computed in Section 3.3. Then* $\lambda^{-1}(\text{CUT}(H))$ *is the most permissive IOFSM (in the sense of trace inclusion) such that* $\lambda^{-1}(\text{CUT}(H))$ **ioco** S^λ.

Proof : Let $H = \text{LIVE-LOCK-FREE}(G, S^\lambda) = \mathcal{C}_l/G$ and $H' = \text{CUT}(H)$. The CUT operation eliminates states that have unexpected deadlocks and output-locks. In these states, by construction, the λ event is not fireable. Thus, the CUT operation applied to H does not eliminate any λ events from H, which implies that $\lambda(\lambda^{-1}(H'))) \subseteq H'$, thus, $\mathcal{L}(\lambda(\lambda^{-1}(H'))) \subseteq \mathcal{L}(H')$. Moreover, since $Traces(H) \subseteq \mathcal{L}(S^\lambda)$, we have $Traces(\lambda(\lambda^{-1}(H'))) \subseteq Traces(H') \subseteq Traces(H) \subseteq \mathcal{L}(S^\lambda) = Traces(S^\lambda)$. By Prop. 2, $\lambda^{-1}(H')$ **ioco** S^λ.
Next, we have to prove that $\lambda^{-1}(\text{CUT}(H))$ is the most permissive IOFSM such that $\lambda^{-1}(\text{CUT}(H))$ **ioco** S^λ. The proof proceeds by induction over the fix-point (10) of Definition 13. Let G^{ioco} be an IOFSM defined as in Prop. 6. According to Prop. 6, we have $Traces(\lambda(G^{ioco})) \subseteq Traces(H_o)$. We assume that $Traces(\lambda(G^{ioco})) \subseteq Traces(H_i)$, and prove that $Traces(\lambda(G^{ioco})) \subseteq Traces(H_{i+1})$. For this, assume that there exists a trace $s \in Traces(\lambda(G^{ioco})) \setminus Traces(H_{i+1})$. Since $Traces(\lambda(G^{ioco})) \subseteq Traces(H_i)$, we have $s \in Traces(H_i)$. Let $s^{-1} \in P^{-1}_{\Sigma^\lambda \to \Sigma^\lambda_{!?}}(s) \cap \mathcal{L}(H_i)$ be an arbitrary sequence of events (including internal events) whose projection on the observable events is s. Let $x \in \mathcal{X}_i$ be the state reached via this sequence. Since $s \notin Traces(H_{i+1})$, this means $s^{-1} \notin \mathcal{L}(H_{i+1})$, hence, $x \notin \mathcal{X}_{i+1}$, that is, $x \in \mathcal{X}_i \setminus \mathcal{X}_{i+1}$. Then, by Definition 13 of the CUT operation this means that $\delta_i(x) \subseteq \Sigma_?$, i.e., x is a deadlock or output-lock state of H_i. Now, the sequence s^{-1} was chosen arbitrarily among those whose projection on the observable behavior is s. Hence, all such sequences lead to a deadlock or an output-lock state; formally, $\delta_i(H_i \textit{ after } s) \subseteq \Sigma_?$. Since $s \in Traces(\lambda(G^{ioco}) \subseteq Traces(H_i)$, this means that $\delta(\lambda(G^{ioco}) \textit{ after } s) \subseteq \Sigma_?$. This is absurd, since there cannot be any deadlocks or output locks in $\lambda(G^{ioco})$, as all have been tagged with λ self-loops, cf. Definition 9.

Thus, for all i, $Traces(\lambda(G^{ioco})) \subseteq Traces(H_i)$. In particular, once the fix-point (10) is reached (which is guaranteed to happen, as each iteration decreases the number of states), we have $Traces(\lambda(G^{ioco})) \subseteq Traces(\text{CUT}(H))$ and finally $Traces(G^{ioco}) \subseteq Traces(\lambda^{-1}\text{CUT}(H))$. Hence $\lambda^{-1}(\text{CUT}(H))$ is the most permissive IOFSM that conforms to S^λ. ◇

The previous result ensures that $\lambda^{-1}(\text{CUT}(H))$ is the most permissive IOFSM that conforms to the specification S^λ. However, this new controlled plant may violate the controllability condition, as the CUT operation may have removed uncontrollable transitions. We can extract the largest controllable submachine out of $\lambda^{-1}(\text{CUT}(H))$, but the result may now violate the conformance relation. In fact, we need to iterate the computation of the greatest controllable submachine of $\lambda^{-1}(\text{CUT}(H))$, and the

computation of the greatest conformant submachine, until a fixpoint is reached. This is similar to what happens when computing a supervisor that is both admissible and non-blocking in the classical supervisory control theory. This leads us to the following general algorithm:

Algorithm IOCO-CONTROLLABLE$(G, S^\lambda) \rightarrow \mathcal{C}/G$

(i) G_i =LIVELOCK-FREE(G, S^λ) and $i = 0$

(ii) $G'_i = \text{CUT}(G_i)$

(iii) $G_{i+1} = \text{CONTROLLABLE}(G'_i, \lambda(G), \Sigma^\lambda_{uc})$.

(iv) If $G_{i+1} \neq G'_i$, $i = i + 1$, Goto (ii)

otherwise $\mathcal{C}/G = \lambda^{-1}(G'_i)$.

Our final result is then:

Proposition 8 *$\mathcal{C}/G$ is the most permissive IOFSM with respect to trace inclusion, such that $\mathcal{C}/G$* **ioco** *S^λ and $\mathcal{C}/G$ is controllable w.r.t. G and Σ_{uc}.*

Proof : At each iteration, after point (ii), $\lambda^{-1}(G'_i)$ **ioco** S^λ (the proof is similar to the one of Prop. 7); and after point (iii), $\lambda^{-1}(G_{i+1})$ is controllable w.r.t. G and Σ_{uc} (this is due to the fact that, by construction, G_{i+1} is controllable w.r.t. $\lambda(G)$ and Σ^λ_{uc}, and operation $\lambda^{-1}(.)$ preserves controllability). When the fix-point is reached (which is guaranteed to happen, as each iteration decreases the number of states), we have $G'_i = G_{i+1} = \mathcal{C}/G$, thus, the final result satisfies both required properties. The proof of maximality for $\mathcal{C}/G$ is analogous to the proof showing that iterating the computation of the greatest non-blocking submachine, and the computation of the greatest controllable sub-machine, generates the largest sub-machine having both properties [Wonham, 2002]. ◇

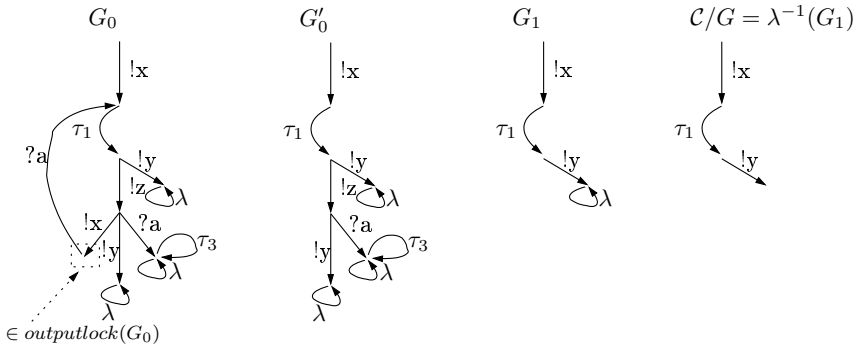

Figure 6: Final step: Deadlocks and output-locks elimination

Example *Figure 6 depicts the application of* IOCO-CONTROLLABLE *to our example.* G_0 *corresponds to the result of the previous step, that is* $\mathcal{C}_l/G_0$ *of figure 5. First, the* CUT *operation cut the* $!x$ *event to remove the output-lock. Then,* CONTROLLABLE *cut the* $!z$ *event (remember that* $!x \in \Sigma_{uc}$*). Finally, the* λ *event is removed by* $\lambda^{-1}()$ *in order to obtain the most permissive IOFSM such as stated in Proposition 8.*

4 Conclusion

The problem addressed here is how to force a plant in order to make it conformant with a reference specification. The proposed solution is to control the plant by means of a supervisor, and to compute the supervisor using control synthesis techniques. Here, conformance to the specification constitutes the control objective. To our knowledge it is the first time this kind of objective is considered. An interesting extension of this work is to consider that the supervisor has only a partial view of the plant. Another point of interest concerns the realization of the supervisor itself (or of the controlled system). Typically, the supervisor will be implemented using traditional programming techniques, which may introduce errors and thus requires another testing phase. The result of the synthesis (i.e., events that were removed for obtaining conformance) may suggest interesting test sequences, which lead to critical points in the final controlled system. Finally, alternatives to the conformance relation **ioco** will also be considered.

References

[Belinfante et al., 1999] Belinfante, A., Feenstra, J., de Vries, R., Tretmans, J., Goga, N., Feijs, L., and Mauw, S. (1999). Formal test automation: a simple experiment. In International Workshop on the Testing of Communicating Systems (IWTCS'99), pages 179–196.

[Cassandras and Lafortune, 1999] Cassandras, C. and Lafortune, S. (1999). *Introduction to Discrete Event Systems.* Kluwer Academic Publishers.

[Jard and Jéron, 2002] Jard, C. and Jéron, T. (2002). TGV: theory, principles and algorithms. In *The Sixth World Conference on Integrated Design & Process Technology (IDPT'02) Pasadena, California.*

[Jard et al., 1999] Jard, C., Jéron, T., Tanguy, L., and Viho, C. (1999). Remote testing can be as powerful as local testing. In Wu, J., Chanson, S., and Gao, Q., editors, *Formal methods for protocol engineering and distributed systems, FORTE XII/ PSTV XIX' 99, Beijing, China*, pages 25–40. Kluwer Academic Publishers.

[Ramadge and Wonham, 1989] Ramadge, P. J. and Wonham, W. M. (1989). The control of discrete event systems. *Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems*, 77(1):81–98.

[Tretmans, 1996] Tretmans, J. (1996). Test generation with inputs, outputs and repetitive quiescence. *Software—Concepts and Tools*, 17(3):103–120.

[Wonham, 2002] Wonham, W. M. (2002). Notes on control of discrete-event systems. Technical Report ECE 1636F/1637S, Department of Electrical and Computer EngineeringUnivertsity of Toronto.