A look at the control of asynchronous automata

Anca Muscholl, Igor Walukiewicz and Marc Zeitoun *

LaBRI
Bordeaux Universtity, France

1 Introduction

In the simplest case, the controller synthesis problem asks to find a model for a given specification. So it is just the satisfiability problem. In a more refined version one is given a system, referred to as a plant, and is asked to find a controller such that the controlled system satisfies a given specification. Here, we are interested in the case when both the plant and the controller are distributed systems. More precisely, when they are modelled by asynchronous automata.

There are numerous versions of the distributed synthesis problem [PR90, LW90, RW92, KV01, MT01, AVW03, MW03, GLZ04, MTY05]. We focus on two variants that are most rooted in the theory of Mazurkiewicz traces. They correspond to two different, and quite intuitive, ways of controlling a distributed system. The interest in these two control problems is also motivated by the fact that their decidability is still open. This contrasts with most of the other settings where one very quickly hits the undecidability barrier [PR90, MT01, AW07].

In the two problems we consider, the goal is to control the behaviour of an asynchronous automaton. This is an automaton with some fixed number of components, or processes, executing in parallel. Each input letter has a preassigned set of processes it acts on. In this way, if two letters have disjoint sets of assigned processes we can consider that they can occur in parallel. The objective is to control an asynchronous automaton so that every possible run satisfies a given specification. The control consists in forbidding some actions of the automaton, but not every action can be forbidden, and the control has also to ensure that the system does not block completely.

In the context of asynchronous automata, it is not reasonable to consider a controller that at every moment of the execution has a complete knowledge of the state of all the processes. This would amount to eliminating all concurrency in the automaton and controlling the resulting finite automaton. Control of finite automata is well studied and much simpler than distributed control [KG95, CL99]. It is much more interesting to try to control an asynchronous automaton without forcing an ad hoc sequentialisation of its behaviours. A

*Work supported by projects DOTS (ANR-06-SETI-003) and DocFlow (ANR-06-MDCA-05).

natural assumption is that the controller has to use the same communication architecture as the plant.

Here, we consider the following two ways of controlling an asynchronous automaton. In the *process-based* version, each process declares which actions it permits to execute. Then, an action can execute if all the processes it involves permit it. In the *action-based* version, each action is declared executable or not by regarding all its processes. This gives more power to the controller as it can look at several processes at the same time.

The process-based version of control was introduced by Madhusudan, Thiagarajan and Yang [MTY05]. The action-based variant was proposed by Gastin, Lerman and Zeitoun [GLZ04]. Madhusudan et al. show the decidability of the control problem for asynchronous automata communicating connectedly (roughly speaking, this requires that if two processes do not synchronise during a long amount of time, then they won't synchronise ever again). The proof proceeds by coding the problem into monadic second-order theory of event structures and showing that this theory is decidable when the criterion holds. Gastin et al. show the decidability of action-based control for asynchronous automata, with a restriction on the form of dependencies between letters of the input alphabet.

Although these two forms of control have been studied for some time, they have never been put side by side. In this paper we have reworded their definitions in order to underline their similarities. Based on this reformulation, we give a reduction from process-based to action-based control. The resemblance of the two definitions allows us also to examine to which extent the above mentioned results can be transferred form one setting to the other.

2 Preliminaries

2.1 Traces and event structures

A *trace alphabet* is a pair (Σ, D), where Σ is a finite set of *actions* and $D \subseteq \Sigma \times \Sigma$ is a *dependence relation* that is reflexive and symmetric. We will use $I = \Sigma \times \Sigma \setminus D$ to denote the *independence relation*.

This alphabet induces a congruence relation $\sim_I$ over the set of words Σ^* over Σ. This is the smallest congruence such that $ab \sim_I ba$ for all $a, b \in I$. An equivalence class of $\sim_I$ is called a (Mazurkiewicz) *trace*. We will use t for denoting traces and write $[u]_I$ for the trace containing $u \in \Sigma^*$ (or simply $[u]$ if I is clear from the context).

A trace t is *prime* if all words it contains end with the same letter. A *prefix* of a trace t' is any trace $t = [v]$ where v is a prefix of some $u \in t'$. Write $t \leq t'$ when t is a prefix of t'; in this case we also say that t' is an *extension* of t. For any two traces t, t' that have some common extension, we write $t \sqcup t'$ for their least common

extension. For a detailed introduction to the theory of traces, see the book [DR95].

Let L be a prefix closed set of traces. In the same way as a prefix closed set of words forms a tree, a prefix closed set of traces forms an *event structure* that we denote by $ES(L)$. The latter is a tuple $\langle E, \leq, \#, \lambda \rangle$, where:

- $E = \{e \in L : e \text{ prime trace}\}$,
- $e \leq e'$ if e is a prefix of e',
- $e \# e'$ if e and e' do not have a common extension in L,
- $\lambda(e)$ is the last letter of a word in e.

Observe that if $\lambda(e)$ and $\lambda(e')$ are dependent then either $e \# e'$, or the two events are comparable with respect to the $\leq$ relation.

For an event $e \in E$ we use $e{\downarrow} = \{e' : e' \leq e\}$ for the set of events below e. More generally, for a set $C \subseteq E$ we write $C{\downarrow}$ for the set $\bigcup_{e \in C} e{\downarrow}$. A *configuration* C of an event structure is a conflict-free, downward-closed subset $C \subseteq E$. That is, $C{\downarrow} = C$ and no events $e, e' \in C$ satisfy $e \# e'$. Notice that every configuration C corresponds to a trace. We can even have a bijection between configurations of $ES(L)$ and traces in L in the case when L is both prefix and extension closed; the latter means that for every two traces $t, t' \in L$ having some common extension, also $t \sqcup t' \in L$.

2.2 Asynchronous automata

Let $\mathbb{P}$ be a finite set of *processes*. Consider an alphabet Σ and a function $loc : \Sigma \to (2^{\mathbb{P}} \setminus \emptyset)$. A (deterministic) *asynchronous automaton* is a tuple

$$\mathcal{A} = \langle \{S_p\}_{p \in \mathbb{P}}, s_{in}, \{\delta_a\}_{a \in \Sigma} \rangle ,$$

where

- S_p is a finite set of (local) states of process p,
- $s_{in} \in \prod_{p \in \mathbb{P}} S_p$ is a (global) initial state,
- $\delta_a : \prod_{p \in loc(a)} S_p \stackrel{\cdot}{\to} \prod_{p \in loc(a)} S_p$ is a transition relation; so on a letter $a \in \Sigma$ it is a partial function on tuples of states of processes in $loc(a)$.

The location mapping loc defines in a natural way an independence relation I: two actions $a, b \in \Sigma$ are independent if they use different processes, i.e., if $loc(a) \cap loc(b) = \emptyset$. An asynchronous automaton can be seen as a sequential automaton with the state set $S = \prod_{p \in \mathbb{P}} S_p$ and transitions $s \xrightarrow{a} s'$ if $((s_p)_{p \in loc(a)}, (s'_p)_{p \in loc(a)}) \in \delta_a$, and $s_q = s'_q$ for all $q \notin loc(a)$. By $L(\mathcal{A})$ we denote the set of words labelling runs of this sequential automaton that start from the initial

state. This definition has an important consequence. If $(a, b) \in I$ then the same state is reached on the words ab and ba. More generally, whenever $u \sim_I v$ and $u \in L(\mathcal{A})$ then $v \in L(\mathcal{A})$, too. This means that $L(\mathcal{A})$ is trace closed. By definition, $L(\mathcal{A})$ is also prefix closed, thus it defines an event structure $ES(L(\mathcal{A}))$, which we also denote as $ES(\mathcal{A})$.

3 Two variants of distributed control problems

As in the standard setting of Ramadge and Wonham [RW89] for finite sequential automata, we can divide the input alphabet of an asynchronous automaton into controllable and uncontrollable actions. We can then ask if there is a way of choosing controllable actions so that a given specification is satisfied. The important parameter here is the mechanism of controlling actions. A most general solution is to ask for a global controller that makes decisions by looking at the global state of the system. This approach completely ignores the distributed aspect of the system. A more interesting approach is to ask for a controller that respects the concurrency present in the automaton. There are several ways of formalising this intuition. Here we present two. The first one was proposed by Madhusudan et al. [MTY05], the second one by Gastin et al. [GLZ04].

3.1 Process-based control

A *plant* is a deterministic asynchronous automaton together with a partition of the input alphabet into actions controlled by the *system* and actions controlled by the *environment*: $\Sigma = \Sigma^{sys} \cup \Sigma^{env}$.

A plant defines a game arena, with plays corresponding to initial runs of $\mathcal{A}$. Since $\mathcal{A}$ is deterministic, we can view a play as a word from $L(\mathcal{A})$. Let $\mathit{Plays}(\mathcal{A})$ denote the set of traces associated with words from $L(\mathcal{A})$.

The p-view of a play u, denoted $\mathit{view}_p(u)$, is the smallest trace prefix of u containing all the p-events of u (a p-event is an occurrence of some $a \in \Sigma$ with $p \in \mathit{loc}(a)$). We write $\mathit{Plays}_p(\mathcal{A})$ for the set of plays that are p-views:

$$\mathit{Plays}_p(\mathcal{A}) = \{\mathit{view}_p(u) : u \in \mathit{Plays}(\mathcal{A})\}\,.$$

A *strategy* for a process p is a function $f_p : \mathit{Plays}_p(\mathcal{A}) \to 2^{\Sigma_p}$; where $\Sigma_p = \{a \in \Sigma : p \in \mathit{loc}(a)\}$ is the set of actions of process p. A *process-based strategy* is a family of strategies $\{f_p\}_{p\in\mathbb{P}}$, one for each process.

The set of plays respecting a strategy $\sigma = \{f_p\}_{p\in\mathbb{P}}$, denoted $\mathit{Plays}(\mathcal{A}, \sigma)$, is the smallest set containing the empty play ε, and such that for every $u \in \mathit{Plays}(\mathcal{A}, \sigma)$:

- if $a \in \Sigma^{env}$ and $ua \in \mathit{Plays}(\mathcal{A})$ then ua is in $\mathit{Plays}(\mathcal{A}, \sigma)$.
- if $a \in \Sigma^{sys}$ and $ua \in \mathit{Plays}(\mathcal{A})$ then $ua \in \mathit{Plays}(\mathcal{A}, \sigma)$ provided that $a \in f_p(\mathit{view}_p(u))$ for all $p \in \mathit{loc}(a)$.

Intuitively, the definition says that actions of the environment are always possible, whereas actions of the system are possible only if they are allowed by the strategies of all the involved processes.

Before defining specifications, we need to make it precise what are infinite plays that are consistent with a given strategy σ. Let X be an infinite set of traces from $\mathit{Plays}(\mathcal{A}, \sigma)$ such that for any two traces t, t' in X, there is some trace t'' in X with $t \leq t''$ and $t' \leq t''$ (such a set is called directed). The limit of an infinite, directed set X, denoted $\sqcup X$, is the least ω-trace with $t \leq \sqcup X$ for every $t \in X$ (for the formal definition of ω-traces see [DR95]). We write $\mathit{Plays}^{\omega}(\mathcal{A}, \sigma)$ for the set of ω-traces of the form $\sqcup X$, where X is a maximal, directed subset of $\mathit{Plays}(\mathcal{A}, \sigma)$.

Specifications for controllers are given by ω-regular languages of traces. Such languages can be defined as ω-regular word languages L that are closed under the equivalence $\sim_I$: if $u_i \sim_I v_i$ for all i, then $u_0 u_1 \cdots \in L$ iff $v_0 v_1 \cdots \in L$. A process-based *controller* satisfying a specification *Spec* is a process-based strategy σ, such that $\mathit{Plays}^{\omega}(\mathcal{A}, \sigma) \subseteq \mathit{Spec}$.

In order to avoid trivial solutions we need also to impose another restriction. A strategy is *non-blocking* if every play in $\mathit{Plays}(\mathcal{A}, \sigma)$ having an extension in $\mathit{Plays}(\mathcal{A})$ also has an extension in $\mathit{Plays}(\mathcal{A}, \sigma)$. Intuitively, the system is not allowed to stop the execution of the plant by forbidding all possible actions.

The *process-based control problem* is to determine for a given plant and specification if there is a process-based controller for the plant that is non-blocking and satisfies the specification.

Madhusudan et. al. introduced the notion of connectedly communicating plants, meaning that in every prime trace of $L(\mathcal{A})$, each of its events has at most $|\mathcal{A}|$ many concurrent events. They showed:

Theorem 1. *[Madhusudan & Thiagarajan & Yang] The process-based control problem is decidable for connectedly communicating plants.*

3.2 Action-based control

As in the process-based version, a plant is a deterministic[†] asynchronous automaton $\mathcal{A}$ with a partition of the input alphabet. The difference is that with an action-based control we need to define the

[†] In [GLZ04], non-deterministic plants were admitted. Moreover, non-blocking controlled behaviour was not required (but the specification could enforce it). These minor differences with our presentation are motivated in order to present a unified framework.

views of actions from Σ. The a-view of a play u, denoted $view_a(u)$, is the smallest trace prefix of u that contains all p-events of u, for every $p \in loc(a)$. Notice that a trace u in $Plays_a(\mathcal{A})$ needs not be prime, in general, but ua is. We write $Plays_a(\mathcal{A})$ for the set of plays which are a-views:

$$Plays_a(\mathcal{A}) = \{view_a(u) : u \in Plays(\mathcal{A})\}.$$

A *strategy* for an action $a \in \Sigma^{sys}$ is a function $g_a : Plays_a(\mathcal{A}) \to \{tt, ff\}$. An *action-based strategy* is a family of strategies $\{g_a\}_{a \in \Sigma^{sys}}$, one for each action in Σ^{sys}.

The set of plays respecting a distributed strategy $\rho = \{g_a\}_{a \in \Sigma^{sys}}$ is the smallest set containing ε, and such that if $u \in Plays(\mathcal{A}, \rho)$ we also have:

- if $a \in \Sigma^{env}$ and $ua \in Plays(\mathcal{A})$ then ua is in $Plays(\mathcal{A}, \rho)$.
- if $a \in \Sigma^{sys}$ and $ua \in Plays(\mathcal{A})$ then $ua \in Plays(\mathcal{A}, \rho)$ provided that $g_a(view_a(u)) = tt$.

The definitions of $Plays^{\omega}(\mathcal{A}, \rho)$ and of non-blocking controllers are exactly as before. The notion of a controller satisfying a specification is also the same.

The *action-based control problem* is to determine for a given plant and specification if there is a non-blocking action-based controller for the plant satisfying the specification.

Decidability of the action-based control problem for plants over trace alphabets of a special form has been shown, first for reachability conditions [GLZ04], later for all ω-regular specifications [Ler05]. A trace alphabet (Σ, D) can be seen as a graph with letters from Σ as nodes and the edges given by D. Since D is symmetric this graph is undirected; D is also reflexive but we will not put self loops in the graph. A *co-graph* is a graph that can constructed from singletons using parallel and sequential products. Another characterisation is that it is a graph without an induced P_4 subgraph, i.e. without a graph of the form $x_1 - x_2 - x_3 - x_4$ as an induced subgraph.

Theorem 2. *[Gastin & Lerman & Zeitoun] The action-based control problem for plants over trace alphabets that are co-graphs and ω-regular specifications is decidable.*

3.3 An example

We give a small example showing the difference between process-based and action-based control. Consider a system of two processes $\mathbb{P} = \{1, 2\}$ with actions $\Sigma = \{a_1, b_1, a_2, b_2, c, d\}$ distributed as follows:

$$loc(c) = loc(d) = \{1, 2\} \quad loc(a_p) = loc(b_p) = \{p\} \quad \text{for } p = 1, 2$$

Intuitively, it means that actions c and d are common to two processes, while a_p and b_p are local to process p. The plant will be very simple. It will allow each process to do either a_p or b_p followed by one common action which can be either c or d. Common actions are controllable by the system, while a_p and b_p are environment actions. Formally we have

$$\mathcal{A} = \langle \{S_p\}_{p\in\mathbb{P}}, s_{in}, \{\delta_a\}_{a\in\Sigma}\rangle$$

with $S_p = \{s_p, t_p, r_p\}$, $s_{in} = (s_1, s_2)$ and the transition relation defined by:

$$\delta_{a_p}(s_p) = \delta_{b_p}(s_p) = t_p \quad \delta_c(t_1, t_2) = \delta_d(t_1, t_2) = (r_1, r_2)$$

Consider a specification saying that if the two processes do action a then the system should do an action c, otherwise it should do d. Formally the specification language L is a prefix and trace closure of $\{a_1a_2c, a_1b_2d, a_2b_1d, b_1b_2d\}$. It means that, for example a_1 and b_2a_1d also belong to L. We will show that there is no process-based strategy for the system while there is an action-based one.

A process-based strategy for system $\mathcal{A}$ is a pair of functions $f_p : \mathit{Plays}_p(\mathcal{A}) \to 2^{\Sigma_p}$; for $p = 1, 2$. In our case $\mathit{Plays}_p(\mathcal{A}) = (a_p + b_p)(\varepsilon + c + d)$. Among this four strings it is only important to define f_p on a_p and b_p. If the environment chooses to do a_1 and b_2 then the system should reply with d. For this we should have $d \in f_1(a_1)$ and $d \in f_2(b_2)$. By similar reasoning we should have $d \in f_1(b_1)$ and $d \in f_2(a_2)$. But then the play a_1a_2d would also be consistent with the strategy. As this is a string not in L, no process-based strategy can satisfy the specification.

An action-based strategy is a pair of functions $g_x : \mathit{Plays}_x(\mathcal{A}) \to \{tt, ff\}$ for $x \in \{c, d\}$; one for each action of the system. In this case we have that $\mathit{Plays}_c(\mathcal{A}) = \mathit{Plays}_d(\mathcal{A})$ is the trace closure of $(a_1 + b_1)(a_2 + b_2)$. We can define

$$g_c(a_1a_2) = g_c(a_2a_1) = tt \quad \text{and } ff \text{ otherwise}$$
$$g_d(a_1a_2) = g_d(a_2a_1) = ff \quad \text{and } tt \text{ otherwise}$$

It is easy to verify that all the plays respecting this strategy satisfy the specification.

We can see with this example that the difference between process and action-based control lies in the information available when the decision is taken. The intuitive reason is that an action-based strategy has more information available, since it can read the state of every process involved in an action in order to decide whether to allow it or not. In contrast, process-based strategies look at each process separately. The example shows that this difference is crucial.

4 Reduction from process-based to action-based control

Action-based strategies are more powerful than process-based ones. It is easy to see that a process-based controller for a given $(\mathcal{A}, \mathit{Spec})$ can be converted into an action-based one. The converse is not true, as shown by the example above. Below, we show that the process-based control problem can be reduced to the action-based problem.

Fix a plant $\mathcal{A}$ and a specification Spec. We will construct $\overline{\mathcal{A}}$ and $\overline{\mathit{Spec}}$ such that: there is a process-based controller for $(\mathcal{A}, \mathit{Spec})$ iff there is an action-based controller for $(\overline{\mathcal{A}}, \overline{\mathit{Spec}})$. Informally, the reduction works as follows: for each process we use additional local system actions, one for every subset of Σ^{sys}. The execution of such an action corresponds to a declaration of the value of the process-based strategy at this process. Then we will have another set of actions, this time for the environment, that will allow to choose one of the declared actions.

We set:

$$\begin{aligned}
\overline{S}_p &= S_p \cup (S_p \times 2^{\Sigma^{sys}}) \cup (S_p \times \Sigma^{sys}),\\
\overline{\Sigma}^{sys} &= \Sigma^{sys} \cup (\mathbb{P} \times 2^{\Sigma^{sys}}) \cup \{\top\},\\
\overline{\Sigma}^{env} &= \Sigma^{env} \cup (\mathbb{P} \times \Sigma^{sys}) \cup \{\bot\}.
\end{aligned}$$

We also need to specify the locations of the new letters. For all $p \in \mathbb{P}$, $a \in \Sigma^{sys}$ and $\theta \in 2^{\Sigma^{sys}}$ we set:

$$\begin{aligned}
&\mathit{loc}\big((p,\theta)\big) = \mathit{loc}\big((p,a)\big) = \{p\},\\
&\mathit{loc}(\top) = \mathit{loc}(\bot) = \mathbb{P}.
\end{aligned}$$

The idea is that by playing its local action (p, θ_p), process p declares that it is willing to enable any action of θ_p. Later on, the environment will have to pick some action a present in all sets θ_p with $p \in \mathit{loc}(a)$. Thus, a move on action $a \in \Sigma^{sys}$ in $\mathcal{A}$ is simulated in $\overline{\mathcal{A}}$, up to commutation of independent actions, by a sequence of the form $(p_1, \theta_1), \ldots, (p_i, \theta_i), (p_1, a), \ldots, (p_i, a), a$, where $\mathit{loc}(a) = \{p_1, \ldots, p_i\}$ and $a \in \theta_j$ for all j. Actions $\bot$ and $\top$ will be used to "punish" the system or the environment for an "unfair" behaviour captured by the following definitions:

- We say that a global state $\prod_{p\in\mathbb{P}}(s_p, \theta_p)$ of $\overline{\mathcal{A}}$ is *s-blocking* if there is an action of $\mathcal{A}$ possible from $\prod_{p\in\mathbb{P}} s_p$ but for every such action a we have $a \notin \theta_p$ for some $p \in \mathit{loc}(a)$.
- We say that a global state $\prod_{p\in\mathbb{P}}(s_p, a_p)$ of $\overline{\mathcal{A}}$ is *e-blocking* if there is an action of $\mathcal{A}$ possible from $\prod_{p\in\mathbb{P}} s_p$ but for every such action a we have $a \neq a_p$ for some $p \in \mathit{loc}(a)$.

An s-blocking state represents a situation when some actions are

possible in the plant, but all of them are blocked due to the choice made by the system. The e-blocking state is similar, but here the reason is that the environment has not made a consistent choice of a next action.

Now we define the transition function of $\overline{\mathcal{A}}$:

$$\begin{aligned}
\overline{\delta}_{(p,\theta)}(s_p) &= \{(s_p,\theta)\} && \text{if } s_p \in S_p,\ \theta \in 2^{\Sigma^{sys}}\\
\overline{\delta}_{(p,a)}((s_p,\theta)) &= \{(s_p,a)\} && \text{if } s_p \in S_p,\ a \in \theta\\
\overline{\delta}_a(\prod_{p\in loc(a)}(s_p,a)) &= \delta_a(\prod_{p\in loc(a)} s_p) && \text{if } a \in \Sigma^{sys}\\
\overline{\delta}_a(s) &= \delta_a(s) && \text{if } a \in \Sigma^{env} \text{ and } s \in \prod_{p\in loc(a)} S_p\\
\overline{\delta}_\top(s) &= s && \text{if } s \in \prod_{p\in\mathbb{P}}(S_p \times \Sigma^{sys}) \text{ is e-blocking}\\
\overline{\delta}_\bot(s) &= s && \text{if } s \in \prod_{p\in\mathbb{P}}(S_p \times 2^{\Sigma^{sys}}) \text{ is s-blocking}
\end{aligned}$$

The specification $\overline{Spec}$ contains an ω-trace t if: (i) it does not have an occurrence of $\bot$, and (ii) either it has an occurrence of $\top$, or its projection $t_{|\Sigma}$ on Σ is in *Spec*. In general, a projection of a trace may not be a trace. For example, the words $a\bot b$ and $b\bot a$ are not trace equivalent, but their projections are if a and b are independent. Fortunately, in our case $u_{|\Sigma}$ is a trace when u has neither $\top$ nor $\bot$.

Lemma 3. *If* $(\mathcal{A}, Spec)$ *has a process-based controller then* $(\overline{\mathcal{A}}, \overline{Spec})$ *has an action-based controller.*

Proof. Given a process-based controller $\sigma = \{f_p\}_{p\in\mathbb{P}}$ for $\mathcal{A}$, we construct an action-based controller $\rho = \{g_a\}_{a\in\overline{\Sigma}^{sys}}$ for $\overline{\mathcal{A}}$ as follows:

$$\begin{aligned}
&g_{(p,\theta)}(v) = tt \text{ iff } \theta = f_p(v_{|\Sigma}), \text{ for all } v \in Plays_{(p,\theta)}(\overline{\mathcal{A}}),\\
&g_a(v) = tt \text{ for every } v \text{ in } Plays_a(\overline{\mathcal{A}}),\\
&g_\top(v) = tt \text{ for every } v \text{ in } Plays_\top(\overline{\mathcal{A}}).
\end{aligned}$$

We want to show that if $v \in Plays(\overline{\mathcal{A}}, \rho)$ and contains neither $\top$ nor $\bot$, then $v_{|\Sigma} \in Plays(\mathcal{A}, \sigma)$. This will show the same statement for infinite executions: any $v \in Plays^\omega(\overline{\mathcal{A}}, \rho)$ either contains $\top$ or $\bot$, or $v_{|\Sigma} \in Plays^\omega(\mathcal{A}, \sigma)$. We will then get that ρ satisfies $\overline{Spec}$ if σ satisfies *Spec*. We also need to show that ρ is non-blocking and that it avoids $\bot$.

Let $state(\overline{\mathcal{A}}, v)$ denote the global state reached by $\overline{\mathcal{A}}$ after reading v. Similarly for $state(\mathcal{A}, v)$. We write $\prod_{p\in\mathbb{P}} s_p \sim \prod_{p\in\mathbb{P}} \overline{s}_p$ if for every

$p \in \mathbb{P}$, either $s_p = \overline{s}_p$, or s_p is the first component of $\overline{s}_p$. By induction on the length of $v \in \mathit{Plays}(\overline{\mathcal{A}}, \rho)$ we show that if v contains neither $\top$ nor $\bot$ then $v_{\,|\Sigma} \in \mathit{Plays}(\mathcal{A}, \sigma)$ and $\mathit{state}(\mathcal{A}, v_{|\Sigma}) \sim \mathit{state}(\overline{\mathcal{A}}, v)$.

If v ends with a letter (p, a) or (p, θ) then the statement is immediate from the induction assumption. For the case of $v = ua$ with $a \in \Sigma^{env}$ we use the induction hypothesis $\mathit{state}(\overline{\mathcal{A}}, u) \sim \mathit{state}(\mathcal{A}, u_{\,|\Sigma})$. From $u, ua \in \mathit{Plays}(\overline{\mathcal{A}}, \sigma)$ we deduce that the transition of $\overline{\mathcal{A}}$ from $\mathit{state}(\overline{\mathcal{A}}, u)$ is possible. The definition of $\overline{\mathcal{A}}$ tells us that on environment actions the transitions of $\overline{\mathcal{A}}$ are the same as that of $\mathcal{A}$. Hence the transition of $\mathcal{A}$ on a is possible from $\mathit{state}(\mathcal{A}, u_{\,|\Sigma})$. Thus $ua \in \mathit{Plays}(\mathcal{A}, \sigma)$, and the constraint on states holds, since the same transition is taken by the two automata. Next, we consider $v = ua$ with $a \in \Sigma^{sys}$. As $\mathit{Plays}(\overline{\mathcal{A}}, \rho)$ is trace closed, we can assume that u ends with the sequence $(p_{i_1}, \theta_{i_1}), \ldots, (p_{i_k}, \theta_{i_k}), (p_{i_1}, a), \ldots, (p_{i_k}, a)$, where $\mathit{loc}(a) = \{p_{i_1}, \ldots, p_{i_k}\}$. From the definition of the strategy ρ it follows that $f_{p_{i_j}}(\mathit{view}_{p_{i_j}}(u_{|\Sigma})) = \theta_{i_j}$ and $a \in \theta_{i_j}$ for all $j = 1, \ldots, k$. This means that $u_{|\Sigma}a \in \mathit{Plays}(\mathcal{A}, \sigma)$ and $\mathit{state}(\overline{\mathcal{A}}, v) \sim \mathit{state}(\mathcal{A}, v_{|\Sigma})$.

We need also to ensure that if σ is non-blocking then ρ is also non-blocking and that $\bot$ is not reachable. Take $v \in \mathit{Plays}(\overline{\mathcal{A}}, \rho)$ and suppose that $v_{|\Sigma}$ has a prolongation in $\mathit{Plays}(\mathcal{A}, \sigma)$. From the above we know that $\mathit{state}(\overline{\mathcal{A}}, v) \sim \mathit{state}(\mathcal{A}, v_{|\Sigma})$. The first case is when $\mathit{state}(\overline{\mathcal{A}}, v)$ has at least one component in a state from $\mathcal{A}$, say $s_p \in S_p$ on the p-th component. In this case it is possible to extend v either by an environment action or by the system action (p, θ_p), where $\theta_p = f_p(\mathit{view}_p(v_{|\Sigma}))$. The next case is when all components are of the form (s_p, θ_p), observe that we have to have $\theta_p = f_p(\mathit{view}_p(v_{|\Sigma}))$ as any execution needs to respect the strategy ρ. As σ is non-blocking, $\overline{\mathcal{A}}$'s transition on $\bot$ is not applicable (the state $(s_p, \theta_p)_{p\in\mathbb{P}}$ is not s-blocking). Next, suppose that there is at least one component in a state of the form (s_p, θ_p) with $\theta_p \neq \emptyset$. Then (s_p, a) with $a \in \theta_p$ is a possible next action. Finally, assume that all components are in states of the form (s_p, a_p) or $(s_p, \emptyset)$. Then either a letter from Σ^{sys} or $\top$ is possible.

It remains to show the converse.

Lemma 4. *If there is an action-based controller for $(\overline{\mathcal{A}}, \overline{\mathit{Spec}})$ then there is a process-based controller for $(\mathcal{A}, \mathit{Spec})$.*

Proof. We first need an observation about action-based controllers. We call a strategy $\rho = \{g_a\}_{a\in\overline{\Sigma}^{sys}}$ *deterministic* if for all $p \in \mathbb{P}$ and $v \in \mathit{Plays}_{(p,\theta)}(\overline{\mathcal{A}})$, there is at most one $\theta \in 2^{\Sigma^{sys}}$ such that $g_{(p,\theta)}(v)$ is true. We claim that if there is a controller then there is a deterministic one. To see this suppose that we have a controller ρ such that $g_{(p,\theta_1)}(v)$ and $g_{(p,\theta_2)}(v)$ hold for some v, p, and two distinct θ_1, θ_2. Then we modify the strategy into ρ' where we set $g_{(p,\theta_2)}(v)$

to false. Clearly $Plays(\overline{\mathcal{A}}, \rho')$ is not bigger than $Plays(\overline{\mathcal{A}}, \rho)$. So all the sequences admitted by ρ' are in $\overline{Spec}$. The new strategy is also non-blocking if ρ was non-blocking.

Let us now suppose that $\rho = \{g_a\}_{a \in \overline{\Sigma}^{sys}}$ is a deterministic strategy. Thanks to determinism, for every $p \in \mathbb{P}$ we can introduce an auxiliary function F_p defined by $F_p(v) = \theta$ iff $g_{(p,\theta)}(v)$ holds. For a trace u in $Plays(\mathcal{A})$ we define a trace $\overline{u}$ in the following way:

$$\overline{ua} = \overline{u}a \qquad \text{for } a \in \Sigma^{env},$$
$$\overline{ua} = \overline{u}\big(p_1, F_{p_1}(view_{p_1}(\overline{u}))\big) \cdots \big(p_k, F_{p_k}(view_{p_k}(\overline{u}))\big)(p_1, a) \cdots (p_k, a)a$$
$$\text{for } a \in \Sigma^{sys} \text{ with, for easier notation, } loc(a) = \{p_1, \ldots, p_k\}.$$

Naturally the definition is the same also for actions that have the sets of locations of a different form than $\{p_1, \ldots, p_k\}$.

We assume that $\overline{ua}$ is undefined if so is one of the expressions of the right hand side. We construct the process-based strategy $\sigma = \{f_p\}_{p \in \mathbb{P}}$:

$$f_p(u) = F_p(view_p(\overline{u})) \quad \text{if } \overline{u} \text{ defined and in } Plays(\overline{\mathcal{A}}).$$

By a simple induction on the size of u, one can show that if $u \in Plays(\mathcal{A}, \sigma)$ then $\overline{u} \in Plays(\overline{\mathcal{A}}, \rho)$ and $state(\mathcal{A}, u) = state(\overline{\mathcal{A}}, \overline{u})$. This implies that if all infinite plays consistent with ρ satisfy $\overline{Spec}$, then all infinite plays consistent with σ satisfy $Spec$. It remains to check that σ is non-blocking if ρ is. We take $u \in Plays(\mathcal{A}, \sigma)$ and the corresponding word $\overline{u} \in Plays(\overline{\mathcal{A}}, \rho)$. Suppose that $\mathcal{A}$ can do an action from $state(\mathcal{A}, u)$. As ρ is non-blocking and winning (in particular, ρ can avoid $\bot$), $\overline{u}$ has an extension $\overline{u}vb \in Plays(\overline{\mathcal{A}}, \rho)$ with $b \in \Sigma$ and $v \in (\overline{\Sigma} \setminus \Sigma)^*$. Then $ub \in Plays(\mathcal{A}, \sigma)$.

With these two lemmas we get:

Theorem 5. *For every $(\mathcal{A}, Spec)$ one can construct $(\overline{\mathcal{A}}, \overline{Spec})$ with the property that: there is a process-based controller for $(\mathcal{A}, Spec)$ iff there is an action-based controller for $(\overline{\mathcal{A}}, \overline{Spec})$.*

This theorem says that in general it is easier to solve process control problems. Unfortunately, it is not universally applicable. For instance, we cannot use Theorem 2 to get decidability of the process-based control for trace alphabets that are co-graphs. Our reduction extends the alphabet, and this extension does not preserve the property of being a co-graph. For example, in the original alphabet we can have just two actions with locations $\{p_1, p_2\}$ and $\{p_2, p_3\}$. In the extended alphabet we add actions on each process so we will have an action with location $\{p_1\}$ and one with location $\{p_3\}$. This results in a P_4 induced subgraph.

5 Control problems and MSOL over event structures

In this section we encode the two control problems into the satisfiability problem of monadic second order logic (MSOL) over event structures. This encoding gives a decidability of the control problem for plants $\mathcal{A}$ such that $ES(\mathcal{A})$ has decidable MSOL theory. This is for example the case for connectedly communicating processes. As we will see, while sufficient, the decidability of the MSOL theory of $ES(\mathcal{A})$ is not a necessary condition for the decidability of control problems.

Monadic second order logic over event structures has two binary relations $<$ and $\#$, and unary relations R_a, one for each letter in the alphabet. The interpretation of a relation R_a is the set of events labelled with a. The interpretation of $<$ and $\#$ is, as expected: the partial order, and the conflict relation between events.

5.1 Encoding for the process-based control

We describe in this section the reduction of the process-based control problem to the satisfiability problem of MSOL over event structures, as provided in [MTY05]. For a given specification *Spec* we want to write a formula φ_{Spec} such that for every plant $\mathcal{A}$: the process-based control problem for $(\mathcal{A}, Spec)$ has a solution iff $ES(\mathcal{A}) \vDash \varphi_{Spec}$.

As a preparation, let us see how one can talk about plays from $\mathcal{A}$ inside $ES(\mathcal{A})$. A play is a trace and it corresponds to a configuration inside the event structure. Such a configuration can be described by its maximal events. We can also talk about $view_p(u)$ inside the event structure, as it is the smallest configuration containing all events from u that are labelled with letters having p in their domain.

A process-based strategy $\sigma = \{f_p\}_{p \in \mathbb{P}}$ can be encoded into an event structure with the help of second-order variables Z_p^a, for every $p \in \mathbb{P}$ and $a \in \Sigma^{sys}$ such that $p \in loc(a)$. Observe that elements of $ES(\mathcal{A})$ are prime traces, and that $Plays_p(\mathcal{A})$ are exactly the prime traces that end with an action from p. We define Z_p^a as the set of all $u \in Plays_p(\mathcal{A})$ such that $a \in f_p(u)$. An event (prime trace) $u = u'a$ may of course belong to more than one Z_p^a, ranging over processes $p \in loc(a)$. Clearly there is a bijection between strategies and such assignments to variables Z_p^a.

Now we need to define $Plays(\mathcal{A}, \sigma)$ inside the event structure. The first observation is that this set is determined by the prime traces in it. Indeed, a configuration is in the set iff all its prime subconfigurations are in the set. Then we write a constraint that an event (prime trace) ua with $u \in Plays(\mathcal{A}, \sigma)$ and $a \in \Sigma^{sys}$ is in the set iff for all $p \in loc(a)$ we have $view_p(u) \in Z_a^p$. Similarly for $a \in \Sigma^{env}$.

Finally, let us see how to talk about infinite traces in $Plays^{\omega}(\mathcal{A}, \sigma)$ and about satisfying the specification *Spec*. As for finite plays, an infinite configuration X in $Plays^{\omega}(\mathcal{A}, \sigma)$ is determined by its prime subconfigurations. Moreover, we need to state that such a set X is maximal (no event $u \in Plays(\mathcal{A}, \sigma)$ can be added to X such that a larger configuration is obtained). Using [EM96], we can assume that the specification *Spec* is given by a monadic second-order formula ψ_{Spec} over ω-traces. Then it suffices to ensure that any set X of events that describes an infinite trace in $Plays^{\omega}(\mathcal{A}, \sigma)$ satisfies ψ_{Spec}. We also need to say that a strategy is non-blocking. For this it suffices to say that every configuration inside $Plays(\mathcal{A}, \sigma)$ that is not maximal in the event structure has a proper extension in $Plays(\mathcal{A}, \sigma)$.

5.2 Encoding for the action-based control

We show now how to adapt the encoding presented in the previous section to action-based control:

Proposition 6. *Given a ω-regular trace specification Spec, one can write an MSOL formula φ_{Spec} such that for every plant $\mathcal{A}$, the action-based control problem for $(\mathcal{A}, Spec)$ has a solution iff $ES(\mathcal{A}) \models \varphi_{Spec}$.*

Notice first that the a-views of configurations (or of traces in $Plays(\mathcal{A})$) can be described in a similar way as p-views. Namely, $view_a(u)$ is the smallest configuration containing every event of every process $p \in loc(a)$.

The encoding of an action-based strategy $\rho = \{g_a\}_{a \in \Sigma^{sys}}$ is even simpler than in the process-based case. We use second-order variables Z^a, $a \in \Sigma^{sys}$, with the following interpretation: an event e with label a belongs to Z^a iff the prime trace ua associated with e satisfies $g_a(u) = tt$. Now we describe the set of finite plays $Plays(\mathcal{A}, \rho)$. As before, a configuration is in $Plays(\mathcal{A}, \rho)$ iff all its prime subconfigurations are in this set. We write that a prime configuration ua belongs to $Plays(\mathcal{A}, \rho)$ iff $u \in Plays(\mathcal{A}, \rho)$ and moreover $ua \in Z_a$ if $a \in \Sigma^{sys}$.

For the encodings of $Plays^{\omega}(\mathcal{A}, \rho)$ and *Spec* we proceed exactly as in the process-based case.

Madhusudan et al. [MTY05] showed that if $\mathcal{A}$ is a connectedly communicating asynchronous automaton, then $ES(\mathcal{A})$ has a decidable MSOL theory. Together with the encoding above this gives decidability of the action-based control for such plants.

Proposition 7. *The action-based control problem for connectedly communicating plants is decidable.*

We can now observe that the decidability of the MSOL theory of $ES(\mathcal{A})$ is not a necessary condition for the decidability of controller problems, neither for the process nor for the action variant. Indeed it suffices to take a three letter alphabet $\{a, b, c\}$ with dependencies

induced by the pairs (a, c) and (b, c). This alphabet is a co-graph so by Theorem 2 the action-based control problem is decidable. It is not difficult to check that the extended alphabet used in the translation from Section 4 is still a co-graph. Hence, by Theorem 2, the process-based control problem for this three letter alphabet is also decidable. Now, consider $\mathcal{A}$ that generates all the traces over this alphabet. The MSOL theory of $ES(\mathcal{A})$ is undecidable as it contains a grid as a subgraph.

References

[AVW03] A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *Theoretical Computer Science*, 303(1):7–34, 2003.

[AW07] A. Arnold and I. Walukiewicz. Nondeterministic controllers of nondeterministic processes. In J. Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata*, volume 2 of *Texts in Logic and Games*, pages 29–52. Amsterdam University Press, 2007.

[CL99] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[DR95] V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, 1995.

[EM96] W. Ebinger and A. Muscholl. Logical definability on infinite traces. *Theoretical Computer Science*, 154(3):67–84, 1996.

[GLZ04] P. Gastin, B. Lerman, and M. Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In *FSTTCS*, volume 3328 of *Lecture Notes in Computer Science*, pages 275–286, 2004.

[KG95] R. Kumar and V. K. Garg. *Modeling and control of logical discrete event systems*. Kluwer Academic Pub., 1995.

[KV01] O. Kupferman and M.Y. Vardi. Synthesizing distributed systems. In *Proc. 16th IEEE Symp. on Logic in Computer Science*, 2001.

[Ler05] B. Lerman. *Vérification et Spécification des Systèmes Distribués*. PhD thesis, Université Denis Diderot - Paris VII, 2005. `http://tel.archives-ouvertes.fr/tel-00322322/fr/`.

[LW90] F. Lin and M. Wonham. Decentralized control and coordination of discrete-event systems with partial observation. *IEEE Transactions on automatic control*, 33(12):1330–1337, 1990.

[MT01] P. Madhusudan and P.S. Thiagarajan. Distributed control and synthesis for local specifications. In *ICALP'01*, volume 2076 of *Lecture Notes in Computer Science*, pages 396–407, 2001.

[MTY05] P. Madhusudan, P. S. Thiagarajan, and S. Yang. The MSO theory of connectedly communicating processes. In *FSTTCS*, volume 3821 of *lncs*, pages 201–212, 2005.

[MW03] S. Mohalik and I. Walukiewicz. Distributed games. In *FSTTCS'03*, volume 2914 of *Lecture Notes in Computer Science*, pages 338–351, 2003.

[PR90] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *31th IEEE Symposium Foundations of Computer Science (FOCS 1990)*, pages 746–757, 1990.

[RW89] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(2):81–98, 1989.

[RW92] K. Rudie and W. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Trans. on Automat. Control*, 37(11):1692–1708, 1992.