Nondeterministic controllers of nondeterministic processes

André Arnold and Igor Walukiewicz

[1] LaBRI
CNRS and University Bordeaux I
33405 Talence, France

Abstract

The controller synthesis problem as formalized by Ramadge and Wonham consists of finding a finite controller that when synchronized with a given plant results in a system satisfying a required property. In this setting, both a plant and a controller are deterministic finite automata, while synchronization is modelled by a synchronous product. Originally, the framework was developed only for safety and some deadlock properties. More recently, Arnold et. al. have extended the setting to all mu-calculus expressible properties and proposed a reduction of the synthesis problem to the satisfiability problem of the mu-calculus. They have also presented some results on decidability of distributed synthesis problem where one requires to find several controllers that control the plant at the same time. The additional difficulty in this case is that each controller is aware of a different part of the whole system.

In this paper, an extension of the setting to nondeterministic processes is studied. In other words, the case when both a system and a controller can be presented by a nondeterministic automaton. This extension is motivated by examples in control where a continuous quantity is measured and digitized thereby introducing imprecision and uncertainty. It is shown that nondeterminism of the plant can be handled at no extra cost, both for centralized and decentralized control. Centralized synthesis remains decidable even for nondeterministic controllers. In contrast, very few cases of decentralized control are decidable when controllers are allowed to be nondeterministic. A classification of decidable/undecidable variants of this problem is given.

1 Introduction

At the end of the eighties, Ramadge and Wonham introduced the theory of control of discrete event systems (see the survey [13] and the books [6]

and [3]). In this theory a process (also called a *plant*) is a deterministic non-complete finite state automaton over an alphabet A of events, which defines all possible sequential behaviours of the process. The goal is to find for a given plant another process, called *controller*, such that a synchronous product of the plant and the controller satisfies desired properties. The usual properties considered are for instance, that some dangerous states are never reachable, or that one can always go back to the initial state of the plant. In *decentralized* control one looks for a fixed number of controllers that control the plant simultaneously.

In the setting described above one assumes that both a plant and controllers are deterministic automata. This paper examines what changes when assumption on determinism is dropped. It is shown that nondeterminism in a plant can be handled at no cost, while nondeterminism in controllers may lead to undecidability in the case of decentralized control.

The synthesis problem would be interesting neither form theoretical nor from practical point of view if there were no additional restrictions on controllers. In the most standard form a restriction is determined by two subsets A_{unc} and A_{uobs} of the alphabet A with the associated requirement that:

(C) For any state q of the controller, and for any uncontrollable event a, there is a transition from q labelled by a.

(O) For any state q of the controller, and for any unobservable event a, if there is a transition from q labelled by a then this transition is a loop over q.

In other words, a controller must react to any uncontrollable event and cannot detect the occurrence of an unobservable event.

In [1] an extension of this setting was proposed that handles specifications expressed in the mu-calculus, or rather in its extension called modal loop mu-calculus. This allowed a more general formulation of the synthesis problem:

> (**CC**) Given a plant P and two formulas α and β, does there exist a controller R satisfying β such that $P \times R$ satisfies α?

With formulas α and β one can express properties (C) and (O) but also much more, as for example that an action becomes unobservable after a failure has occurred, or that always one of two actions is controllable but never both at the same time.

The problem (**CC**) can be solved thanks to the division operation [1]. For a process P and a formula α there is a formula α/P such that: $R \vDash \alpha/P$ iff $P \times R \vDash \alpha$. This way a process R is a solution to (**CC**) if and only if $R \vDash (\alpha/P) \wedge \beta$. As $(\alpha/P) \wedge \beta$ is a formula of the modal loop mu-calculus

the synthesis problem reduces to the constructive satisfiability problem: construct a model for a formula whenever a model exists. The latter is decidable and a witnessing model, which is a controller, can be constructed.

Ramadge and Wonham have considered also the problem of synthesis of decentralized controllers: a plant can be supervised by several independent controllers (instead of only one). But each controller has its own set of controllable and observable events. Hence the decentralized control problem is to find $R_1, \ldots, R_n$ such that the supervised system $P \times R_1 \times \cdots \times R_n$ satisfies the specification S and for each i, R_i satisfies (C_i) and (O_i). More generally, in our setting, a decentralized control problem is:

> (**DC**) Given a plant P and modal-loop mu-calculus formulas α, β_1, $\ldots$,β_n, do there exist controllers R_i satisfying β_i (for $i = 1, \ldots, n$) such that $P \times R_1 \times \cdots \times R_n$ satisfies α?

In [1] it is shown how to solve a decentralized control problem when at most one of the formulas α_i restrains visibility of a controller. If one allows to put visibility restrictions on at least two controllers then the existence of a solution to the problem is undecidable.

Till now, all the constructions assumed that processes are deterministic automata. This may be a limiting assumption if, for example, a plant is a model of a continuous system. In this case a continuous domain of values must be sampled to a discrete one. Hence, the same measurement can correspond to different values that may have different effect on the behaviour of the plant. For similar reasons, the result of actions of controllers may be also not completely determined.

In this paper, we show that in the case of centralized synthesis the approach via division operation still works. We do this by generalizing the division operation described above to a division by a nondeterministic process. This shows that nondeterminism in a plant can be handled at no cost. Next, we study decidability of (**DC**) problem. Thanks to the division, allowing nondeterministic plant does not make the problem more complex. By contrast, if we allow at least two controllers to be nondeterministic, then the problem becomes undecidable even for formulas in the standard mu-calculus. We study also the case when at most one of the controllers can be nondeterministic, obtaining a full characterisation of decidability of (**DC**) problem.

The paper is organized as follows. In the next section we introduce processes and automata on processes. This will be a rather rich version of alternating automata that has not only loop testing, $\circlearrowleft_a$, but also indistinguishability testing $\downdownarrows_{a,b}$. Intuitively, the first constraint will be used to say that a controller cannot observe a, and the second that it cannot make a difference between a and b. These kinds of automata were introduced in [1], and $\downdownarrows_{a,b}$ test was added in [2]. In Section 3 we give basic properties

of these automata, like closure under boolean operations and decidability of emptiness. Section 4 presents an operation of division of an automaton by a process. This operation is used in the following section to solve centralized synthesis problem and to eliminate the plant from formalization of decentralized synthesis problem. Main results of the paper are given in this section. The proofs of decidability and undecidability results announced here are given in Sections 6 and 7 respectively.

2 Processes and automata

2.1 Processes

Let A be a finite alphabet of *actions*. A *process* is a finite graph with a distinguished node and with edges labelled by actions:

$$P = \langle S, A, s^0 \in S, e \subseteq S \times A \times S \rangle$$

We will usually refer to nodes as *states*. We will write $s \xrightarrow{a} s'$ instead of $(s, a, s') \in e$, and will say that there is a transition labelled a form a state s to a state s'. A process is *deterministic* if e is a partial function $e : S \times A \to S$. We will write $out_P(s, a)$ for the set of states reachable from s by a transition labelled a: $out_P(s, a) = \{s' : s \xrightarrow{a} s'\}$.

A *product* of two processes over the same alphabet is standard

$$P \times R = \langle S_P \times S_R, A, (s^0_P, s^0_R), e_{P\times R} \rangle$$

where $((s_P, s_R), a, (s'_P, s'_R)) \in e_{P\times R}$ if $(s_P, a, s'_P) \in e_P$ and $(s_R, a, s'_R) \in e_R$.

2.2 Games

As our specification language we will use a rich variant of alternating automata that we will introduce in the next subsection. It will be very convenient to describe its semantics in terms of games, so we recall necessary definitions here.

A *game* G is a tuple $\langle V_E, V_A, T \subseteq (V_E \cup V_A)^2, Acc \subseteq V^\omega \rangle$ where Acc is a set defining the *winning condition*, and $\langle V_E \cup V_A, T \rangle$ is a graph with the vertices partitioned into those of Eve and those of Adam. We say that a vertex v' is a *successor* of a vertex v if $T(v, v')$ holds.

A *play* between Eve and Adam from some vertex $v \in V = V_E \cup V_A$ proceeds as follows: if $v \in V_E$ then Eve makes a choice of a successor, otherwise Adam chooses a successor; from this successor the same rule applies and the play goes on forever unless one of the parties cannot make a move. The player who cannot make a move looses. The result of an infinite play is an infinite path $v_0 v_1 v_2 \ldots$ This *path is winning* for Eve if it belongs to Acc. Otherwise Adam is the winner.

A *strategy* σ for Eve is a function assigning to every sequence of vertices $\vec{v}$ ending in a vertex v from V_E a vertex $\sigma(\vec{v})$ which is a successor of v.

A *play respecting* σ is a sequence $v_0 v_1 \dots$ such that $v_{i+1} = \sigma(v_i)$ for all i with $v_i \in V_E$. The *strategy σ is winning for Eve* from a vertex v iff all the plays starting in v and respecting σ are winning. A *vertex is winning* if there exists a strategy winning from it. The strategies for Adam are defined similarly. A strategy is *positional* if it does not depend on the sequence of vertices that were played till now, but only on the present vertex. So such a strategy can be represented as a function $\sigma : V_E \to V$ and identified with a choice of edges in the graph of the game.

In this paper the winning conditions $Acc \subseteq V^\omega$ will be *regular conditions*. That is conditions defined in monadic second-order logic on sequences. An important special case is a *parity condition*. It is a condition determined by a function $\Omega : V \to \{0, \dots, d\}$ in the following way:

$$Acc = \{v_0 v_1 \dots \in V^\omega : \limsup_{i \to \infty} \Omega(v_i) \text{ is even}\}$$

Hence, in this case, each position is assigned a natural number and we require that the largest among those appearing infinitely often is even. This condition was discovered by Mostowski [9] and is the most useful form of regular conditions. It guarantees existence of positional strategies [4, 10, 8]. It is closed by negation (the negation of a parity condition is a parity condition). It is universal in the sense that every game with a regular condition can be reduced to a game with a parity condition [9].

The main results about games that we need are summarized in the following theorem.

Theorem 2.1 ([7, 4, 10])**.** Every game with regular winning conditions is determined, i.e., every vertex is winning for one of the players. It is algorithmically decidable who is a winner from a given vertex in a finite game. In a parity game a player has a positional strategy winning from each of his winning vertices.

2.3 Automata

We will need automata that work on graphs. These automata should cope with multiple outgoing edges with the same label. Moreover, we would like to equip them with tests of some simple structural graph properties. They will be able to check that an edge on a given letter is a self-loop or that the edges on two different letters lead to the same states. To incorporate all these tests it will be simpler to define automata which use a kind of modal formulas over the set of states in a process. Thus we start with defining these formulas.

Let A be an alphabet and let Q be a finite set. The set of *modal formulas* over A and Q, denoted $\mathcal{F}(A, Q)$, is the smallest set closed under the following rules:

- tt, ff, q, $\circlearrowleft_a$, $\overline{\circlearrowleft}_a$, $\downdownarrows_{a.b}$ $\overline{\downdownarrows}_{a,b}$ are formulas, for any $q \in Q$ and $a, b \in A$.
- $\alpha \vee \beta$ and $\alpha \wedge \beta$ are formulas for all $\alpha, \beta \in \mathcal{F}(A, Q)$.
- $\langle a \rangle \alpha$ and $[a]\alpha$ are formulas for all $\alpha \in \mathcal{F}(A, Q)$ and $a \in A$.

An *automaton* over a set of actions A is a tuple:

$$\mathcal{A} = \langle Q, A, q^0 \in Q, \delta : Q \to \mathcal{F}(A, Q), Acc \subseteq Q^\omega \rangle$$

where Q is a finite set of states, A is a finite alphabet, and Acc is an accepting condition that is a regular set of infinite sequences of states.

The acceptance of a process P by an automaton $\mathcal{A}$ is defined in terms of strategies in a game $G(P, \mathcal{A})$ that we describe now. Let $\mathcal{F}^{\mathcal{A}}$ be the smallest set of formulas closed under taking subformulas, and containing all formulas in the range of δ together with tt and ff. We have

$$G(P, \mathcal{A}) = \langle V_E, V_A, T, Acc_G \rangle$$

where

- $V_E = S \times \mathcal{F}_E^{\mathcal{A}}$, and $\mathcal{F}_E^{\mathcal{A}}$ is the set of formulas form $\mathcal{F}^{\mathcal{A}}$ of one of the forms: $ff, \circlearrowleft_a, \downdownarrows_{a,b}, q, \langle a \rangle \alpha$, $\alpha \vee \beta$.
- $V_A = S \times \mathcal{F}^{\mathcal{A}} - V_E$.
- From (s, tt) and (s, ff) no move is possible.
- From $(s, \circlearrowleft_a)$ there is a move to (s, tt) if $out_P(s, a) = \{s\}$ and to (s, ff) otherwise.
- From $(s, \downdownarrows_{a,b})$ there is a move to (s, tt) if $out_P(s, a) = out_P(s, b)$ and to (s, ff) otherwise.
- Similarly for $(s, \overline{\circlearrowleft}_a)$ and $(s, \overline{\downdownarrows}_{a,b})$ but with roles of (s, tt) and (s, ff) interchanged.
- From $(s, \alpha \wedge \beta)$ as well as from $(s, \alpha \vee \beta)$ there are moves to (s, α) and to (s, β).
- From $(s, \langle a \rangle \alpha)$ and from $(s, [a]\alpha)$ there are moves to (t, α) for every $t \in out(s, a)$.
- Finally, from (s, q) there is a move to $(s, \delta(q))$.
- The winning condition Acc_G contains sequences such that when looking only at the elements of Q appearing in the sequence we obtain an element of Acc.

We say that P *satisfies* $\mathcal{A}$, in symbols $P \vDash \mathcal{A}$, if Eve has a winning strategy in $G(P, \mathcal{A})$ from (s^0, q^0), which is the pair consisting from the initial states of P and $\mathcal{A}$, respectively. As our automata are very close to formulas we prefer to talk about satisfiability instead of acceptance. We will still use some automata terminology though. For example, the *language recognized by an automaton* $\mathcal{A}$ will be the class of processes that satisfy $\mathcal{A}$.

Our automata are a variant of alternating automata. In particular the formulas used in the transition function are "closed" under disjunction and conjunction. Using standard constructions on alternating automata we get.

Proposition 2.2. The class of languages recognized by automata is closed under sum, intersection and complement.

This proposition allows to write $\mathcal{A} \wedge \mathcal{C}$ to denote an automaton which recognizes $L(\mathcal{A}) \cap L(\mathcal{C})$.

Definition 2.3. An automaton is called *simple* if formulas in its transition function use none of $\circlearrowleft_a, \overline{\circlearrowleft}_a, \downdownarrows_{a,b}, \overline{\downdownarrows}_{a,b}$.

A simple automaton is nothing else but a μ-calculus formula in a different presentation. Using the results on the μ-calculus we have.

Theorem 2.4 ([4, 14])**.** It is decidable if for a given simple automaton $\mathcal{A}$ there is a process P such that $P \vDash \mathcal{A}$. Similarly, if we require P to be deterministic. In both cases a process P can be constructed if the answer is positive.

Theorem 2.5 ([11, 5])**.** Over deterministic systems which are trees, the expressive power of simple automata is equivalent to that of monadic second-order logic. Over all processes: a property is expressible by a simple automaton iff it is expressible in monadic second-order logic and bisimulation invariant.

3 Satisfiability

The basic question one can ask about our automata model is whether for a given automaton $\mathcal{A}$ there is a process that satisfies it. From the previous section we know that there is an algorithm answering this question in the case of simple automata. We will now reduce the general case to that of simple automata. For this, we will encode information about loops and parallel tests in additional transitions. This way for a process P we will define a process $\mathit{Code}(P)$. It will then turn out that behaviour of an automaton over P can be simulated by a behaviour of a simple automaton over $\mathit{Code}(P)$.

A *type* of a state s of a process P is:

$$\mathit{type}(s) = \{\circlearrowleft_a \colon \mathit{out}(s,a) = \{s\}\} \cup \{\downdownarrows_{a,b} \colon \mathit{out}(s,a) = \mathit{out}(s,b)\}$$

Let $Types(A)$ be the set of all possible types over an alphabet A.

Note that if $\tau \in Types(A)$ and $\downdownarrows_{a,b} \in \tau$ then $\circlearrowleft_a \in \tau$ implies $\circlearrowleft_b \in \tau$, and also $\downdownarrows_{a,c} \in \tau$ implies $\downdownarrows_{b,c} \in \tau$.

Fix an alphabet A and some arbitrary ordering $<_A$ on it. For a process P over an alphabet A we define its code $Code(P)$ over an alphabet $A \cup Types(A)$. For each state s of P we do the following:

- Add a transition on action $\tau = type(s)$, the target of this transition is some arbitrary fixed state (say, the initial state).
- Remove transitions on a if $\circlearrowleft_a \in type(s)$ or $\downdownarrows_{a,b} \in type(s)$ for some $b <_A a$.

Let $\mathcal{C}$ be a simple automaton expressing the conditions:

- For every state there is transition on exactly one letter from $Types(A)$.
- For every $a \in A$, there is no transition on a if $\circlearrowleft_a \in \tau$ or $\downdownarrows_{a,b} \in \tau$ for some $b <_A a$.

Lemma 3.1. The process $Code(P)$ satisfies $\mathcal{C}$ and has no loops $s \xrightarrow{a} s$ Moreover, $Code(P)$ is deterministic if P is. If R is a process satisfying $\mathcal{C}$ without loops $s \xrightarrow{a} s$ then there is a unique process P such that $Code(P)$ is isomorphic to R.

The next step is to transform an automaton over an alphabet A into an "equivalent" automaton over an alphabet $A \cup Types(A)$. Take an automaton

$$\mathcal{A} = \langle Q, A, q^0, \delta : Q \to \mathcal{F}(A, Q), Acc \subseteq Q^\omega \rangle$$

We first define transformation, $Code(\alpha)$, on formulas from $\mathcal{F}(A, Q)$:

- $Code(q) = q$.
- $Code(\circlearrowleft_a) = \bigvee\{\langle\tau\rangle tt : \tau \in Types(A), \circlearrowleft_a \in \tau\}$. Similarly for $\downdownarrows_{a,b}$.
- $Code(\overline{\circlearrowleft}_a) = \bigvee\{\langle\tau\rangle tt : \tau \in Types(A), \circlearrowleft_a \notin \tau\}$. Similarly for $\overline{\downdownarrows}_{a,b}$.
- $Code(\alpha \vee \beta) = Code(\alpha) \vee Code(\beta)$, and similarly for the conjunction.
- $Code(\langle a\rangle\alpha) = \bigvee\{\langle\tau\rangle tt \wedge Code(\langle a\rangle\alpha, \tau) : \tau \in Types(A)\}$ where
  $Code(\langle a\rangle\alpha, \tau) = \begin{cases} Code(\alpha) & \text{if } \circlearrowleft_a \in \tau \\ \langle a\rangle Code(\alpha) \vee \bigvee\{\langle b\rangle Code(\alpha) : \downdownarrows_{a,b} \in \tau\} & \text{otherwise} \end{cases}$
- $Code([a]\alpha) = \bigvee\{\langle\tau\rangle tt \wedge Code([a]\alpha, \tau) : \tau \in Types(A)\}$; where the definition of $Code([a]\alpha, \tau)$ is as above but replacing $\langle a\rangle$ by $[a]$, $\langle b\rangle$ by $[b]$, and disjunctions by conjunctions.

Then automaton $Code(\mathcal{A})$ is the same as $\mathcal{A}$ except for the transition function δ_{Code}. We put $\delta_{Code}(q) = Code(\delta(q))$. The following lemma follows directly from definitions.

Lemma 3.2. For every process P and automaton $\mathcal{A}$ over an alphabet A:

$$P \vDash \mathcal{A} \quad \text{iff} \quad Code(P) \vDash Code(\mathcal{A})$$

Observe that $Code(\mathcal{A})$ is simple, i.e., does not use neither $\circlearrowleft$ nor $\downdownarrows$ constraints. Using $Code(\mathcal{A})$ we can transfer results from simple automata to the general case.

Theorem 3.3. It is decidable if for a given automaton $\mathcal{A}$ there exist a process P such that $P \vDash \mathcal{A}$. Similarly, if we ask for P being deterministic. In both cases, if the answer is positive then a process satisfying $\mathcal{A}$ can be constructed.

Proof. Consider $Code(\mathcal{A})$. As $Code(\mathcal{A}) \wedge \mathcal{C}$ is a simple automaton, by Theorem 2.4 we can test if there exists a process $R \vDash Code(\mathcal{A}) \wedge \mathcal{C}$. Unfolding the loops of R we can construct a process R' without loops such that $R' \vDash Code(\mathcal{A}) \wedge \mathcal{C}$. Lemma 3.1 gives us a process P such that $Code(P)$ is isomorphic to R', hence $Code(P) \vDash Code(\mathcal{A})$. By Lemma 3.2 we have $P \vDash \mathcal{A}$. This construction works also when we require P to be deterministic.

Conversely, if P is a (deterministic) process that satisfies $\mathcal{A}$ then the (deterministic) process $Code(P)$ satisfies $Code(\mathcal{A})$, by Lemma 3.2, and $\mathcal{C}$, by Lemma 3.1. Q.E.D.

4 Quotient for extended automata

In this section we present an operation that will permit us to reduce synthesis problems to the satisfiability problems. Consider an extended automaton $\mathcal{A} = \langle Q, A, q_0, \delta, Acc\rangle$ and a process $P = \langle S, A, s^0, e\rangle$ over a common alphabet A. Our goal is to construct an automaton $\mathcal{A}/P$ such that for every process R:

$$R \vDash \mathcal{A}/P \quad \text{if and only if} \quad P \times R \vDash \mathcal{A}$$

We first define a division α/s for α a formula from $\mathcal{F}(A, Q)$, and s a state of P. The result is a formula from $\mathcal{F}(A, Q \times S)$:

$$q/s = (q, s)$$

$$(\alpha \vee \beta)/s = \alpha/s \vee \beta/s \qquad (\alpha \wedge \beta)/s = \alpha/s \wedge \beta/s$$

$$(\langle a\rangle\alpha)/s = \langle a\rangle \bigvee\{\alpha/s' : s \xrightarrow{a} s'\} \qquad ([a]\alpha)/s = [a] \bigwedge\{\alpha/s' : s \xrightarrow{a} s'\}$$

Now

$$\mathcal{A}/P = \langle Q \times S, A, (q^0, s^0), \delta_/ : Q \times S \to \mathcal{F}(Q \times S), \Omega\rangle$$

where $\delta_/(q, s) = \delta(q)/s$; recall that $\delta(q) \in \mathcal{F}(Q)$, so $\delta(q)/s \in F(Q \times S)$.

Theorem 4.1. Let A be an alphabet. For every extended automaton $\mathcal{A}$ and every process P, both over A, there is an automaton $\mathcal{A}/P$ such that for every process R over A:

$$R \vDash \mathcal{A}/P \quad \text{if and only if} \quad P \times R \vDash \mathcal{A}$$

Proof. Fix a process R. We examine the games $G_\times = G(P \times R, \mathcal{A})$ and $G_/ = G(R, \mathcal{A}/P)$. We want to show how a move of one of the players in $G_\times$ from a position of the form $((s,r),\alpha)$ can be mimicked by, possibly several, moves of the same player in $G_/$ from $(r, \alpha/s)$. For example, suppose that a position has the form $((s,r), \alpha \vee \beta)$ and that Eve chooses $((s,r),\alpha)$. From a position $(r,(\alpha \vee \beta)/s))$ this move can be mimicked by going to $(r, \alpha/s)$. Slightly more complicated is the case of $((s,r), \langle a\rangle\alpha)$. In this case Eve can choose $((s',r'),\alpha)$ for $s \xrightarrow{a} s'$ and $r \xrightarrow{a} r'$. From $(r, (\langle a\rangle\alpha)/s)$ this move can be mimicked by first choosing $(r', \bigvee\{\alpha/s'' : s \xrightarrow{a} s''\})$ and then $(r', \alpha/s')$; this is possible as $(\langle a\rangle\alpha)/s = \langle a\rangle \bigvee\{\alpha/s'' : s \xrightarrow{a} s''\}$. The cases of $\alpha \wedge \beta$ and $[a]\alpha$ are dual.

These observations show that any play in $G_\times$ can be mimicked in $G_/$, so the same player has a winning strategy from $((s^0,r^0),q^0)$ in $G_\times$ and from $(r^0,(q^0,s^0))$ in $G_/$.

Q.E.D.

5 Solving controller synthesis problems

Equipped with the operation of division we can reduce the control problem to the satisfiability problem.

5.1 Centralized control

As we have argued in the introduction, the centralized controller synthesis problem can be formulated as:

> For a given process P and two automata $\mathcal{A}$, $\mathcal{B}$ over an alphabet A, find R such that:
> $$P \times R \vDash \mathcal{A} \quad \text{and} \quad R \vDash \mathcal{B}$$

We denote by $Sol(P, \mathcal{A}, \mathcal{B})$ the set of solutions to the problem. The following is a direct corollary of Theorem 4.1

Corollary 5.1. For every process R:

$$R \in Sol(P, \mathcal{A}, \mathcal{B}) \text{ if and only if } R \vDash (\mathcal{A}/P) \wedge \mathcal{B}.$$

This means that solving a synthesis problem amounts to checking emptiness of the automaton $(\mathcal{A}/P) \wedge \mathcal{B}$. Theorem 2.1 then states that this problem is decidable both for the general, nondeterministic, case as well as for the case of deterministic processes.

5.2 Decentralized control

The decentralized controller synthesis problem is:

> For a given process P and automata $\mathcal{A}$, $\mathcal{B}_1, \ldots, \mathcal{B}_n$ over an alphabet A, find $R_1, \ldots, R_n$ such that:
>
> $$P \times R_1 \times \cdots \times R_n \vDash \mathcal{A} \quad \text{and} \quad R_i \vDash \mathcal{B}_i \text{ for all } i = 1, \ldots, n$$

Thanks to Theorem 4.1 we can take $\mathcal{A}/P$ and remove P from the left hand side. This shows that we can as well consider the following simpler formulation of the problem where P is not mentioned.

> For a given automata $\mathcal{A}$, $\mathcal{B}_1, \ldots, \mathcal{B}_n$ over an alphabet A, find $R_1, \ldots, R_n$ such that:
>
> $$R_1 \times \cdots \times R_n \vDash \mathcal{A} \quad \text{and} \quad R_i \vDash \mathcal{B}_i \text{ for all } i = 1, \ldots, n$$

This last problem was studied in [1, 2] in the case when $R_1, \ldots, R_n$ are required to be deterministic. In particular the problem was shown decidable when all but one $\mathcal{B}_i$ are simple. We will see later that the same problem is undecidable in the nondeterministic case. To better understand the decidable/undecidable borderline we propose a classification of decentralized synthesis problems with respect to restrictions on $R_1, \ldots, R_n$.

A pair $(pt, st) \subseteq \{det, nondet\} \times \{simple, full\}$ describes requirements on processes and specifications. A choice of a number of components and a type for each component determines a distributed control problem:

> $DS(n, (pt_1, st_1), \ldots, (pt_n, st_n))$:
> Given automata $\mathcal{A}$, $\mathcal{B}_1, \ldots, \mathcal{B}_n$ such that $\mathcal{B}_i$ is simple if $st_i = simple$, find $R_1, \ldots, R_n$ such that R_i is deterministic if $pt_i = det$, satisfying $P \times R_1 \times \cdots \times R_n \vDash \mathcal{A}$ and $R_i \vDash \mathcal{B}_i$ for all $i = 1, \ldots, n$.

The following theorem gives a complete classification of these problems with respect to decidability.

Theorem 5.2. The problem $DS(\mathcal{A}, (pt_1, st_1), \ldots, (pt_n, st_n))$ is decidable iff

- There is at most one i such that $st_i = full$.
- There is at most one j such that $pt_j = nondet$ and moreover $j \neq i$.

The proof of this theorem will be given in the two following sections. In terms of the decentralized control problem formulated at the beginning of the section, we get that the problem is decidable if at most one of $\mathcal{B}_i$ is not simple and at most one R_j is allowed to be nondeterministic (moreover $j \neq i$). Probably the most important difference with respect to the deterministic case considered in [1] is that now P can be nondeterministic.

6 The decidable sub-case of decentralized control

We would like to show the right to left implication of Theorem 5.2.

The solution in the case when all $R_1, \ldots, R_n$ are required to be deterministic uses the following extension of the quotient operation :

Theorem 6.1 ([1])**.** For every automaton $\mathcal{A}$ and every simple automaton $\mathcal{B}$ there is an automaton $\mathcal{A}/\mathcal{B}$ such that for every deterministic process P:

$$P \vDash \mathcal{A}/\mathcal{B} \quad \text{iff} \quad \exists R.\ R \text{ deterministic},\ R \vDash \mathcal{B}, \text{ and } P \times R \vDash \mathcal{A}$$

Here we show existence of a variant of this division operation. The difference is that the existentially quantified process R need not be deterministic.

Theorem 6.2. For every automaton $\mathcal{A}$ and every simple automaton $\mathcal{B}$, there is an automaton $\mathcal{A}/_{ndet}\mathcal{B}$ such that for every deterministic process P:

$$P \vDash \mathcal{A}/_{ndet}\mathcal{B} \quad \text{iff} \quad \exists R.\ R \vDash \mathcal{B} \text{ and } P \times R \vDash \mathcal{A}$$

Before giving the proof of this theorem let us show how it can be used to prove right to left direction of Theorem 5.2.

Let us assume that $st_n = full$ and $pt_1 = nondet$. First, we find a deterministic process $R_n \vDash (\mathcal{A}/_{ndet}\mathcal{B}_1/\mathcal{B}_2 \ldots/\mathcal{B}_{n-1}) \wedge \mathcal{B}_n$. If none exists then the problem has no solutions. Otherwise, by Theorem 3.3, we can construct required R_n. Equipped with it we can find a deterministic process $R_{n-1} \vDash (\mathcal{A}/R_n/_{ndet}\mathcal{B}_1/\mathcal{B}_2 \ldots/\mathcal{B}_{n-2}) \wedge \mathcal{B}_{n-1}$. This construction can be repeated, giving $R_{n-1}\ldots$, until we construct a deterministic $R_2 \vDash (\mathcal{A}/R_n/R_{n-1}/\ldots/R_3/_{ndet}\mathcal{B}_1) \wedge \mathcal{B}_2$. Once $R_2, \ldots, R_n$ are fixed, we can look for, this time nondeterministic, process $R_1 \vDash (\mathcal{A}/R_n/R_{n-1}/\ldots/R_2) \wedge \mathcal{B}_1$. By the above two theorems on division operations, $R_1, \ldots, R_n$ is a solution to the problem. The theorems also guarantee that all solutions to the problem can be obtained this way.

The rest of this section presents the proof of Theorem 6.2. We want to transform the property of a deterministic process P:

$$\exists R. \quad R \vDash \mathcal{B} \quad \text{and} \quad P \times R \vDash \mathcal{A} \tag{1.1}$$

to an equivalent formulation that is expressible by an automaton. This will be our automaton $\mathcal{A}/\mathcal{B}$.

The first step is to introduce well-typed processes and restrict our problem only to this kind of processes. Given a process P over an alphabet A, a *well-typed* process, $wt(P)$, is a process over the alphabet $A \cup \mathcal{P}(A)$ that is obtained form P by adding a new state $\top$, and precisely one action from each state as follows: to a state s of P we add a transition to $\top$ on $out^P(s) \in \mathcal{P}(A)$, where $out^P(s)$ is the set of actions possible form s,

$out^P(s) = \{b : out^P(s,b) \neq \varnothing\}$. It should be clear that there is an automaton checking if a processes is of the form $wt(P)$. It is also easy, given an automaton $\mathcal{C}$, to construct an automaton $\mathcal{C}'$ such that for all processes P over an alphabet A

$$wt(P) \vDash \mathcal{C} \quad \text{iff} \quad P \vDash \mathcal{C}'$$

This means that in the following we can consider only processes of the form $wt(P)$. We call these processes *well-typed*.

The restriction to well-typed processes is important for the first simplification step. We want to find an automaton $\mathcal{D}$ such that (1.1) is equivalent to

$$\exists R. \quad P \times R \vDash \mathcal{D} \tag{1.2}$$

For this we construct an automaton $\mathcal{B}'$ and show that (1.1) is equivalent to $\exists R'.\ P \times R' \vDash \mathcal{B}' \wedge P \times R' \vDash \mathcal{A}$. Having this, we can just take $\mathcal{B}' \wedge \mathcal{A}$ for $\mathcal{D}$. We call a process over $A \cup \mathcal{P}(A)$ *typed* if every state has precisely one transition on a letter from $\mathcal{P}(A)$. Compared with well-typed processes, we do not put any restriction what a γ is. We also define a *safe extension* of a typed process R to be a process obtained form R by adding some states and transitions provided that if (s,b,t) is an added transition and s is a state from R then t must be an added state and b must not appear in the label of the unique action from $\mathcal{P}(A)$ possible from s. With these definitions we can say what the automation $\mathcal{B}'$ is. We want $\mathcal{B}'$ to accept a process if it is typed, and moreover it has a safe extension that is accepted by $\mathcal{B}$. It remains to argue that $\mathcal{B}'$ has the desired property. For one direction suppose that we have R' with $P \times R' \vDash \mathcal{B}'$ and $P \times R' \vDash \mathcal{A}'$. If $P \times R' \vDash \mathcal{B}'$ then, by the definition of $\mathcal{B}'$, there is a safe extension R of $P \times R'$ that satisfies $\mathcal{B}$. By the definition of the safe extension, and the fact that P is well-typed we have that $P \times R' = P \times R$. So $P \times R \vDash \mathcal{A}$. Now consider the opposite direction. Take R which is assumed to exists and add to R a state $\top$ as well as transitions to $\top$ from each state of R on every letter from $\mathcal{P}(A)$. As $\mathcal{B}$ does not talk about the actions from $\mathcal{P}(A)$ then $R' \vDash \mathcal{B}$. We have $P \times R' \vDash \mathcal{B}'$ because $P \times R'$ is typed and R' is a safe extension of $P \times R'$. We also have $P \times R' \vDash \mathcal{A}$ as $\mathcal{A}$ does not talk about actions from $\mathcal{P}(A)$.

The above argument reduces our task to the problem of expressing by an automaton the property (1.2) of well-typed P. First, we will consider a simpler property where the branching of the process R we quantify over is bounded by k, i.e. for every $s \in R$ and a, $|out(s,a)| \leq k$.

$$\exists R. \quad \text{branching}(R) \leq k \quad \text{and} \quad P \times R \vDash \mathcal{D} \tag{1.3}$$

This formulation will allow us to use the division operation for the deterministic case, i.e, Theorem 6.1. Consider processes over an alphabet $A_{[k]} = A \times \{1, \ldots, k\}$. A deterministic process P' over an alphabet $A_{[k]}$ represents a nondeterministic process $red(P')$ over an alphabet A where each

action (a,i), for $i=1,\ldots,k$, is mapped to a. Every nondeterministic process of branching bounded by k can be represented in such a way (in general not uniquely). From automaton $\mathcal{D}$ it is easy to construct an automaton $\mathcal{D}_{[k]}$ which accepts a process P' over $A_{[k]}$ iff $red(P')$ is accepted by $\mathcal{D}$. Consider $\mathcal{D}_{[k]}/tt$ where tt is an automaton accepting all the processes over $\mathcal{A}_{[k]}$. By Theorem 6.1 we have

$$P' \vDash \mathcal{D}_{[k]}/tt \quad \text{iff} \quad \exists R'.\ P' \times R' \vDash \mathcal{D}_{[k]};$$

Here, all the processes are over $A_{[k]}$. For a deterministic process P over A we can define $P_{[k]}$ to be a deterministic process over $A_{[k]}$ where there is an edge (b,i), for $i=1,\ldots,k$, between two nodes iff in P there is an edge b between them. For an automaton $\mathcal{D}'$ over $A_{[k]}$ is easy to construct an automaton $red(\mathcal{D}')$ such that for all deterministic processes P over A

$$P \vDash red(\mathcal{D}') \quad \text{iff} \quad P_{[k]} \vDash \mathcal{D}'$$

With this we get

$$P \vDash red(\mathcal{D}_{[k]}/tt) \quad \text{iff} \quad P_{[k]} \vDash \mathcal{D}_{[k]}/tt \quad \text{iff} \quad \exists R'.\ P_{[k]} \times R' \vDash \mathcal{D}_{[k]}$$

where R' and $P_{[k]}$ are over the alphabet $A_{[k]}$. By definition, the last formula is equivalent to $\exists R'.red(P_{[k]} \times R') \vDash \mathcal{D}$. As P is deterministic $red(\ P_{[k]} \times R') = P \times red(R')$. It is easy to see that (1.3) is equivalent to $\exists R'.\ P \times red(R') \vDash \mathcal{D}$ and in consequence to $P \vDash red(\mathcal{D}_{[k]}/tt)$. So, for $\mathcal{A}/_{ndet}\mathcal{B}$ we could take $red(\mathcal{D}_{[k]}/tt)$ if only we could find a bound on k.

We are left to show that we can bound the branching in our problem (1.2), so that we can fix k. The following proposition gives the desired bound.

Lemma 6.3. Let P be a deterministic process and let $\mathcal{A}$ be an automaton with parity acceptance conditions. If there is (possibly nondeterministic) process R such that:

$$P \times R \vDash \mathcal{A}$$

then there is R' with the same property which has the branching degree $|A||\mathcal{A}|$

Proof. Take R such that $P \times R \vDash \mathcal{A}$. Then Eve has a positional winning strategy (cf. Theorem 2.1) in the game $G(P \times R, \mathcal{A})$. This strategy is a function $\sigma : (P\times R)\times\mathcal{F}_E^{\mathcal{A}} \to (P\times R)\times\mathcal{F}^{\mathcal{A}}$ which to pairs of the form $(s, \alpha\vee\beta)$ assigns either (s,α) or (s,β); and to pairs of the form $(s,\langle b\rangle\alpha)$ assigns a pair (s',α) for some $s' \in out_{P\times R}(s,b)$. This function has the property that all the plays respecting suggestions of this function are winning for Eve.

Take some state s of $P \times R$. Let $us(s,b)$, the set of useful successors, be the set of all successors t of s such that $(t,\alpha) = \sigma(s,\langle b\rangle\alpha)$ for some formula

$\langle b\rangle\alpha$. Because the number of formulas of this kind is bounded by the size of $\mathcal{A}$, so is the size of $us(s,b)$.

The intention is that we would like to prune $P \times R$ so that on action b from s only edges to $us(s,b)$ remain. This may not be correct as the following example shows. Suppose that $us(s,b) = us(s,c)$, while $out_{P\times R}(s,b) \neq out_{P\times R}(s,c)$. Now, the result of $\downdownarrows_{b,c}$ test will be different in $P \times R$ and in the pruned structure. Hence, it may happen that $\mathcal{A}$ does not accept the pruned structure.

In order to avoid the problem mentioned in the above paragraph we extend $us(s,b)$ to $us'(s,b)$. For every state s and action b, let $us'(s,b)$ be a set satisfying the following.

- $us(s,b) \subseteq us'(s,b)$.
- if $s \vDash \circlearrowleft_b$ then $s \in us'(s,b)$.
- if $s \vDash \overline{\circlearrowleft}_b$ then either $us'(s,b') = \varnothing$, or $s' \in us'(s,b')$ for some $s' \neq s$ and $s' \in out_{P\times R}(s,b)$.
- if $s \vDash \downdownarrows_{b,c}$ then $us'(s,b) = us'(s,c)$.
- if $s \vDash \overline{\downdownarrows}_{b,c}$ and $out_{P\times R}(s,b) \not\subseteq out_{P\times R}(s,c)$ then $s' \in us'(s,b)$ for some arbitrary chosen $s' \in out_{P\times R}(s,b) - out_{P\times R}(s,c)$.

It is easy to see that $us'(s,b)$ can be chosen in such a way that it is at most $|A|$-times bigger than $us(s,b)$.

Now take $P \times R$ and delete all edges (s,b,t) such that $t \notin us'(s,b)$. Let us call the resulting process R'. In R', strategy σ is still a winning strategy because we have only limited choices for Adam. Hence, Eve wins in $G(R',\mathcal{A})$, and in consequence $R' \vDash \mathcal{A}$. We have that $P \times R' \vDash \mathcal{A}$, as $P \times R' = R'$, since P is deterministic. By construction, the branching of R' is bounded by the maximal possible size of $us'(s,b)$ which is $|A||\mathcal{A}|$.

Q.E.D.

Remark 6.4. If the restriction of determinism of P is dropped than the division $\mathcal{A}/_{ndet}\mathcal{B}$ does not exist even when $\mathcal{A}$ and $\mathcal{B}$ are simple. For example, take $\mathcal{A}$ which says that all maximal paths are of the form a^*b, and if a state has an successor on action a then it does not have one on action b. Consider $\mathcal{A}/_{ndet}\mathcal{A}$. Condition $P \vDash \mathcal{A}/_{ndet}\mathcal{A}$ means that there is R such that $P \times R \vDash \mathcal{A}$ and $R \vDash \mathcal{A}$. If P had two paths a^ib and a^jb of different length then in $P \times R$ we would have a path that does not finish with b. This implies that $P \vDash \mathcal{A}/_{ndet}\mathcal{A}$ iff there is k such that all the paths in P have the form a^kb. So the set of processes satisfying $\mathcal{A}/_{ndet}\mathcal{A}$ is not regular. Observe that in this argument it did not matter whether we restrict to R being deterministic or not.

Remark 6.5. Even when restricting to deterministic processes, automaton $\mathcal{A}/\mathcal{B}$ may not exist if $\mathcal{B}$ is not simple. In [1] it is shown that decentralized control problem is undecidable for $n = 2$ if both $\mathcal{B}_1$ and $\mathcal{B}_2$ are automata with $\circlearrowleft_a$ constraints. In [2] undecidability is shown when both automata used $\downdownarrows_{a,b}$ constraints, or when one uses $\circlearrowleft_a$ constrains and the other $\downdownarrows_{a,b}$ constraints.

7 Undecidable cases of decentralized control

In this subsection we show left to right direction of Theorem 5.2. It will be enough to study the version of the control problem for two processes:

(ABC) Given automata $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ over the same action alphabet A, do there exist, possibly nondeterministic, processes P, R such that $P \vDash \mathcal{A}$, $R \vDash \mathcal{B}$ and $P \times R \vDash \mathcal{C}$.

First, we will show that the problem is undecidable even when $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are simple automata. This will give the proof of Theorem 5.2 for the case when there are at least two processes that can be nondeterministic. Next, we will consider the case when at most one of the processes can be nondeterministic. We will show that the above problem is undecidable when only R can be nondeterministic, and when $\mathcal{B}$ can use either $\circlearrowleft$ constraints or $\downdownarrows$ constrains. This not only will imply the remaining part of Theorem 5.2, but will also show that restricting our automata uniquely to $\circlearrowleft$ constraints or to $\downdownarrows$ constraints does not change the decidability classification.

Before showing these results we would like to introduce a syntactic extension of our setting which will make the presentation easier. We will suppose that we have propositional letters labelling states of processes. So each process comes not only with an alphabet of actions but also with an alphabet Λ of propositions:

$$P = \langle A, \Lambda, S, s^0, e \subseteq S \times A \times S, \lambda : S \to \Lambda \rangle$$

Automata are also extended to reflect this, so the transition function can test what is a label of the current state:

$$\mathcal{A} = \langle Q, A, \Lambda, q^0, \delta : Q \times \Lambda \to F(A, Q), Acc \subseteq Q^\omega \rangle$$

There are many possible definitions of a product of two processes with state labels. Here we choose the one that will suit our needs. Given two processes over the same action alphabet, but possibly different proposition alphabets:

$$P = \langle A, \Lambda_P, S_P, s^0_P, e_p, \lambda_P \rangle \qquad R = \langle A, \Lambda_R, S_R, s^0_R, e_R, \lambda_R \rangle$$

we define their product as:

$$P \otimes R = \langle A, \Lambda_P \times \Lambda_R, S_P \times S_R, (s_P^0, s_R^0), e_\otimes, \lambda_\otimes \rangle$$

where $\lambda_\otimes(s_P, s_R) = (\lambda_P(s_P), \lambda_R(s_R))$ and, as before, $((s_p, s_R), a, (s'_P, s'_R)) \in e_\otimes$ iff $(s_P, a, s'_P) \in e_P$ and $(s_R, a, s'_R) \in e_R$.

It is quite straightforward to see how to simulate propositional letters by actions. One can add propositional letters to the action alphabet and require that from each state there is a transition on exactly one propositional letter; the target of this transitions is of no importance.

The problem with this coding is that the standard product does not reflect our $\otimes$-product. In order to recover the $\otimes$-product, we first make the alphabets Λ_P and Λ_R disjoint. Let $\widehat{P}$, $\widehat{R}$ denote respective plants with encoding of propositions as described above. We add to every state of $\widehat{P}$ an action on every letter from Λ_R and to every state of $\widehat{R}$ an action on every letter of Λ_P. This way we have that $\widehat{P} \times \widehat{R}$ is the encoding of $P \otimes R$: from every state of $\widehat{P} \times \widehat{R}$ we have a successor on exactly one letter from Λ_P and on one letter from Λ_R.

After these remarks it should be clear that instead of the problem (ABC) we can consider the problem ($\text{ABC}_\otimes$) where the processes are allowed to have propositions and $\otimes$ is used in place of ordinary product.

> **($\text{ABC}_\otimes$)** Given automata $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ over the same action alphabet A, and over proposition alphabets Λ_a, Λ_b and $\Lambda_a \times \Lambda_b$ respectively, do there exist processes P, R such that
> $$P \vDash \mathcal{A},\ R \vDash \mathcal{B} \text{ and } P \otimes R \vDash \mathcal{C}.$$

Thus, the following proposition implies undecidability of the problem (ABC).

Proposition 7.1. The problem ($\text{ABC}_\otimes$) is undecidable.

Proof. We will present a reduction of the halting problem. Let us fix a deterministic Turing machine together with an alphabet Γ needed to encode its configurations. We write $c \vdash c'$ to say that a configuration c' is a successor of a configuration c. Without a loss of generality we assume that the machine loops from the accepting configuration. We will use just one action letter, so we will not mention it in the description below. The alphabet of propositions will contain Γ and special symbols: l and $\#$. The nodes labelled by l will be called *l-nodes*; similarly for $\#$-nodes, and γ-nodes for $\gamma \in \Gamma$. We will say that a node is a *Γ-node*, if it is a γ-node for some γ. We will also talk about an l-successor of a node, this a successor that is an l-node. Finally, when we will say that there is a path $\gamma_1 \dots \gamma_n$ in a process, this would mean that there is a sequence of nodes, that is a path, and such that the propositional letters associated to nodes form the sequence $\gamma_1 \dots \gamma_n$.

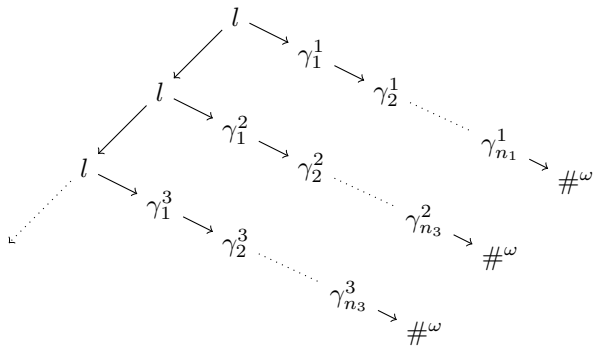

FIGURE 1. Intended shape of a process satisfying **AB1**, **AB2**, **AB3**.

We want to construct $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ so that the problem ($\mathrm{ABC}_\otimes$) has a solution iff the machine accepts when started from the empty tape. Consider the following three conditions that will be used for specifications $\mathcal{A}$ and $\mathcal{B}$:

AB1 Every l-node has an l-successor and a Γ-successor. Every Γ-node has either only Γ-nodes or only #-nodes as successors.

AB2 From every Γ-node, every path reaches a #-node.

AB3 Every #-node has only #-nodes as successors.

The intended shape of a process satisfying these conditions is presented in Figure 1. These conditions do not imply that the shape is exactly as presented in the figure. For example, they do not guarantee that there is only one infinite path labelled with l.

The constraints on the product of two processes are listed below. They are formulated in terms of the product alphabet.

C1 Every (l,l)-node has an (l,l)-successor and an (γ,γ)-successor for some $\gamma \in \Gamma$. Moreover all its successors are labelled either by (l,l), (l,γ), (γ,l) or (γ,γ).

C2 Every maximal path starting with $(l,l)^i(\gamma,\gamma)$ has a form $(l,l)^i\Delta^+(\#,\#)^\omega$ where $\Delta = \{(\gamma,\gamma) : \gamma \in \Gamma\}$.

C3 Every maximal path that starts with $(l,l)^i(\gamma_1,l)(\gamma_2,\gamma_1')$ for some γ_1, γ_2, $\gamma_1' \in \Gamma$ has the form: $(l,l)^i(\gamma_1,l)(\gamma_2,\gamma_1')\dots(\gamma_k,\gamma_{k-1}')(\#,\gamma_k)(\#,\#)^\omega$. Moreover $\gamma_1 \dots \gamma_k \vdash \gamma_1' \dots \gamma_k'$, or the two are identical if $\gamma_1 \dots \gamma_k$ is an accepting configuration.

C4 For every path labelled $(l,l)(\gamma_1,\gamma_2)\dots(\gamma_k,\gamma_k)(\#,\#)^\omega$, the sequence $\gamma_1\dots\gamma_k$ represents the initial configuration for the Turing machine. An accepting state of the machine appears in the tree.

Let $\mathcal{C}$ be the automaton expressing the conjunction of the conditions C1-C4. We claim that with this choice of automata the problem (ABC$_\otimes$) has a solution iff the Turing machine halts on the initial configuration.

We first consider an easy direction. Suppose that the Turing machine halts on the initial configuration. Then we construct P and R as in the Figure 1, where for every i the sequence of Γ letters after l^i is the i-th configuration of the machine (we assume that all configurations are represented by words of the same length). This way P and R satisfy conditions AB1-3. It is straightforward to verify that $P \otimes R$ satisfies the conditions C1-C4.

For the other direction, suppose that P and R are as required. We will show that the machine has an accepting computation from the initial configuration.

First, we show that the conditions we have imposed limit very much possible nondeterminism in P and R. Take any n and a path labelled $l^n\gamma_1\dots\gamma_{k_n}\#^\omega$ in P as well as a path labelled $l^n\gamma'_1\dots\gamma'_{m_n}\#^\omega$ in R. These paths exist by conditions AB1-AB3. In $P\times R$ these two paths give a path that starts with $(l,l)^n(\gamma_1,\gamma'_1)$. The condition AB1 implies that $\gamma_1=\gamma'_1$. Consequently, the condition AB2 implies that $k_n=m_n$ and $\gamma_i=\gamma'_i$ for all $i=1,\dots,k_n$. This allows us to define $u_n=\gamma_1\dots\gamma_{k_n}$. To summarize, in P all paths of the form $l^n\Gamma^+\#^\omega$ have the same labels: $l^nu_n\#^\omega$. Similarly for paths in R.

It remains to show that u_n is the n-th configuration of the computation of the Turing machine. By condition A3, we know that u_1 is the initial configuration. Consider now a path in $P\otimes R$ labelled with

$$(l,l)^n(\gamma_1,l)(\gamma_2,\gamma'_1)\dots(\gamma_k,\gamma'_{k-1})(\#,\gamma'_k)(\#,\#)^\omega$$

This path exists as it is a product of a path in P starting with $l^n\gamma_1$ and a path in R starting with $l^{n+1}\gamma'_1$. We have that $u_n=\gamma_1\dots\gamma_k$ and $u_{n+1}=\gamma'_1\dots\gamma'_k$. By the condition A2 we get $u_n\vdash u_{n+1}$. Q.E.D.

This finishes the case of Theorem 5.2 when at least two processes can be nondeterministic. It remains to consider the case when only one of the processes, say R can be nondeterministic, and when specification $\mathcal{B}$ of R is not simple. We will show that in this case the problem is undecidable even if $\mathcal{B}$ uses uniquely $\downdownarrows$ constraints, or uniquely $\circlearrowleft$ constraints. Recall that the problem is decidable if $\mathcal{B}$ is simple, i.e. uses neither $\downdownarrows$ nor $\circlearrowleft$.

Proposition 7.2. The problem (ABC$_\otimes$) is undecidable if P is required to be deterministic but R may be nondeterministic and moreover a specification for R may use constraints $\downdownarrows$.

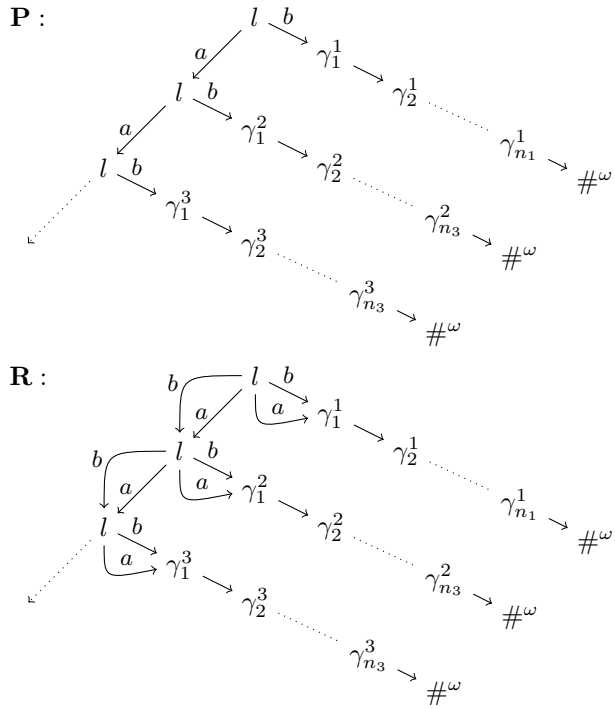

FIGURE 2. $\Downarrow\Downarrow$ constraints. Intended shapes of P and R.

The reduction is very similar to the previous one. We just need to replace nondeterminism with appropriate use of $\Downarrow\Downarrow$. This time our processes will be over the alphabet of two actions $\{a, b\}$. The intended shapes of processes P and R are shown in the Figure 2.

The shape of P is almost the same as in the previous construction, but as P needs to be deterministic, some a transitions have to be changed to b transitions. Process R has almost the same structure as P but it is nondeterministic, and each a transition has a b transition in parallel.

Looking at $P \otimes R$ we get almost exactly the same structure as in the case of nondeterministic processes. The fact that process P is deterministic and that the two transitions from an l-node of P have different actions is compensated by the fact that a and b transitions have the same targets in R.

The formalization of the constraints and the proof of the proposition is almost the same as in case of Proposition 7.1.

The following proposition treats the remaining case of $\circlearrowleft$ constraints.

Proposition 7.3. The problem ($\mathrm{ABC}_{\otimes}$) is undecidable when P is required to be deterministic but R may be nondeterministic and moreover a specification for R may use looping constraints $\circlearrowleft$.

Proof. Consider an instance of the Post correspondence problem:

$$\{(u_1, v_1), \ldots, (u_k, v_k)\};$$

where all u_i, v_i are words over an alphabet Σ. Let $D = \{1, \ldots, k\}$ stand for the alphabet of indices. As an alphabet of actions we take $A = \Sigma \cup D \cup \{\alpha_1, \alpha_2, \beta, \#\}$, with an assumption that the last four actions do not appear in $\Sigma \cup D$.

The specification $\mathcal{A}$ for P will require that

A1 Every state, except the root, has only one successor. The root has successors on α_1 and α_2.

A2 There is a maximal path of the form $\alpha_1 \beta i_1 u_{i_1} \ldots i_m u_{i_m} \#$ for some $i_1, \ldots, i_m \in D$.

A3 There is a maximal path of the form $\alpha_2 \beta j_1 v_{j_1} \ldots j_m v_{j_m} \#$ for some $j_1, \ldots, j_m \in D$.

Observe that together with requirement that P is deterministic, the first condition implies that P has exactly two maximal paths. The shape of P is presented in Figure 3.

The specification $\mathcal{B}$ for R will require that:

B1 The root has loops on actions α_1 and α_2 and some transitions on β.

B2 There is a path from the root of the form $\beta\Sigma^*\#$. Every node on this path except the root has loops on all actions from D and has a successor on at most one action from $\Sigma \cup \{\#\}$.

B3 There is a path from the root of the form $\beta D^*\#$. This time every node except the root has loops on actions from Σ and a successor on at most one action from $D \cup \{\#\}$.

The intended shape of a process satisfying $\mathcal{B}$ is presented in Figure 3. Observe that we cannot force this process to be deterministic.

The specification $\mathcal{C}$ for $P \times R$ will require that all the paths are finite and that the last action on every path is $\#$.

We claim that with this choice of $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$, the problem (ABC) has a solution iff the instance of the Post correspondence problem has a solution.

For the right-to-left direction, take a solution $i_1, \ldots, i_m$ to the correspondence problem. We construct P that has two paths: $\alpha_1 \beta i_1 u_{i_1} \ldots i_m u_{i_m} \#$

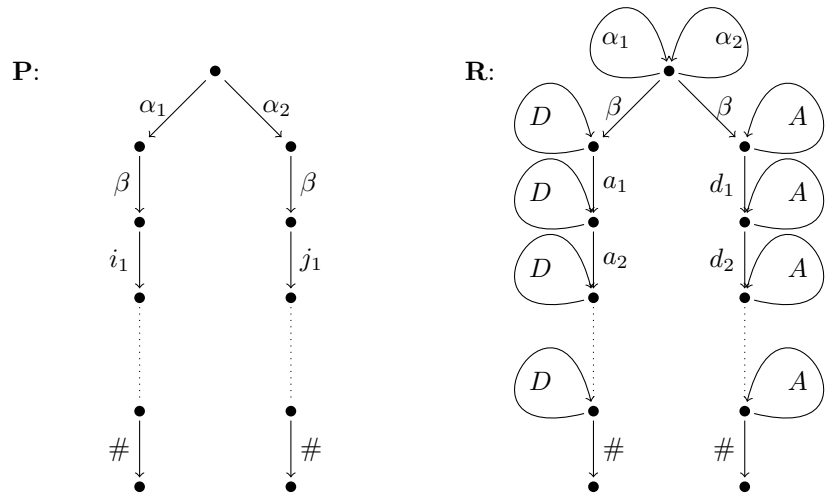

FIGURE 3. ⟳-constraints. Intended shapes of P and R.

and $\alpha_2\beta i_1 v_{i_1} \dots i_m v_{i_m}\#$. As R we take a process as depicted in Figure 3 where the path satisfying condition B2 has the form $\beta u_{i_1} \dots u_{i_m}\#$, and the path satisfying B3 is $\beta i_1 \dots i_m\#$. It is easy to see that $P \times R$ satisfies $\mathcal{A}$.

For the direction from left to right suppose that P and R are a solution to the problem. Consider a path of R labelled $\beta\Sigma^*\#$ satisfying B2 and the path $\alpha_1\beta i_1 u_{i_1} \dots i_m u_{i_m}\#$ of P as required by the condition A2. Recall that there are loops on α_1 and α_2 in the root of R. This means that the two paths synchronize, at least at the beginning. The only way that the synchronization can continue until # is that $u_{i_1} \dots u_{i_m}$ is exactly the labelling of the path in R. We can use the same argument for the path $\alpha_2\beta j_1 v_{j_1} \dots j_m v_{j_n}\#$ and in consequence we get $u_{i_1} \dots u_{i_m} = v_{j_1} \dots v_{j_n}$. If we now repeat this argument once again but with a path of R labelled with $\beta D^*\#$ as required by condition B3 then we will get that $i_1 \dots i_m = j_1 \dots j_n$. This finishes the proof. Q.E.D.

We can now summarize how the three propositions of this subsection can be used to show left to right implication of Theorem 5.2. If two of the processes R_i are allowed to be nondeterministic then the undecidability follows from Proposition 7.1. The case when there are two automata that are not simple but all processes are deterministic was proven in [1] for ⟳ constraints and in [2] for ⇊ constraints, and a mix of ⟳ and ⇊ constraints. If a spec-

ification can use either $\circlearrowleft$ or $\downdownarrows$ constraints and the corresponding process can be nondeterministic then undecidability follows from Propositions 7.2 and 7.3, respectively.

8 Conclusions

In this paper we have studied the controller synthesis problem for nondeterministic plants and controllers. We have seen that going from deterministic to nondeterministic plants does not change the complexity of the problem. Allowing nondeterministic controllers is more delicate. It can be done in centralized case, but in the decentralized case at most one controller can be nondeterministic, moreover it should be able to observe all actions of the plant.

Let us briefly comment on the complexity of the constructions presented here. The operation of division by a process gives an exponential blow-up. It is unavoidable for the same reason as in a translation from alternating to nondeterministic automaton. The complexity of the construction for division by automaton is also exponential.

Given the results above one can ask whether they also apply to the setting of architectures of Pnueli and Rosner [12]. It is quite simple to encode this latter setting into our setting using unobservable actions. Thus all decidability results in our setting transfer to architecture setting. As for undecidability results, one can show by methods very similar as those used in this paper that even two element pipeline becomes undecidable when specifications for controllers are allowed to be nondeterministic.

References

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

[2] Xavier Briand. *Contrôle avec événements indiscernables et inobservables*. PhD thesis, Bordeaux University, 2006.

[3] C. G. Cassandras and S. Lafortune. *Introduction to discrete event systems*. Kluwer Academic Pub., 1999.

[4] E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy. In *Proc. FOCS'91*, pages 368–377, 1991.

[5] David Janin and Igor Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order

logic. In *CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 263–277, 1996.

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

[7] D.A. Martin. Borel determinacy. *Ann. Math.*, 102:363–371, 1975.

[8] Robert McNaughton. Infinite games played on finite graphs. *Ann. Pure and Applied Logic*, 65:149–184, 1993.

[9] Andrzej W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *Fifth Symposium on Computation Theory*, volume 208 of *LNCS*, pages 157–168, 1984.

[10] Andrzej W. Mostowski. Games with forbidden positions. Technical Report 78, University of Gdansk, 1991.

[11] Damian Niwiński. Fixed points vs. infinite generation. In *LICS '88*, pages 402–409, 1988.

[12] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. ACM POPL*, pages 179–190, 1989.

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

[14] Igor Walukiewicz. Monadic second order logic on tree-like structures. *Theoretical Computer Science*, 257(1–2):311–346, 2002.