Distributed synthesis for acyclic architectures

Anca Muscholl and Igor Walukiewicz

University of Bordeaux and CNRS

February 17, 2014

Abstract

Synthesising distributed systems from specifications is an attractive objective, since distributed systems are notoriously difficult to get right. Unfortunately, there are very few known decidable frameworks for distributed synthesis. We present one such framework that is based on communication by rendez-vous and causal memory. This means that the specification can talk about when a communication takes place, but it cannot limit information that is transmitted during communication. This choice is both realistic and avoids some pathological reasons for undecidability. We show decidability of the synthesis problem under the restriction that the communication graph of the system is acyclic. Our result covers all ω-regular local specifications and uncontrollable rendez-vous actions. The former can be used to have e.g. fairness constraints, the latter allows to encode shared variable communication primitives.

1 Introduction

Synthesising distributed systems from specifications is an attractive objective, since distributed systems are notoriously difficult to get right. Unfortunately, there are very few known decidable frameworks for distributed synthesis. We present one such framework that is based on communication by rendez-vous and causal memory. This means that the specification can talk about when a communication takes place, but it cannot limit information that is transmitted during communication. This choice is both realistic and avoids some pathological reasons for undecidability. We show decidability of the synthesis problem under the restriction that the communication graph of the system is acyclic. Our result covers all ω-regular local specifications and uncontrollable rendez-vous actions. The former can be used to have e.g. fairness constraints, the latter allows to encode shared variable communication primitives.

Instead of synthesis we prefer to work in the more general framework of distributed control. Our setting is a direct adaptation of the supervisory control framework of Ramadge and Wonham [17]. Given a plant (a finite automaton) and a specification, the goal is to construct a controller (another finite automaton) such that its product with the plant satisfies the specification. So control means restricting the behavior of the plant. In our case the formulation is exactly the same but we consider Zielonka automata, instead of finite automata, as plants and controllers.

Zielonka automata [19, 14] are a by now a well-established model of distributed computation. Such a device is an asynchronous product of finite-state processes synchronising on shared actions. Asynchronicity means that processes can progress at different speed. The formulation of the control problem we consider means that the controller is not allowed to initiate its own shared actions, but it is allowed to transfer control information during synchronisation actions of the plant.

We show decidability of the control problem for Zielonka automata where the communication graph is acyclic: a process can communicate (synchronise) with its parent and its children. We allow specifications that are conjunctions of ω-regular specifications for each of the component processes. We also allow uncontrollable shared actions. This model can be used to encode some known communication primitives like test-and-set, or compare-and-swap, that enable implementations to employ efficient machine-level atomic instructions that are available on modern multi-core processors.

Most research on distributed synthesis and control has been done in the setting proposed by Pnueli and Rosner [16]. This setting is also based on shared-variable communication, however it does not allow to pass additional information between processes. So their model leads to partial information games, and decidability of synthesis holds only for very restricted architectures [3]. Actually, using the interplay between specifications and architecture in a suitable way, one can get undecidability results for most architectures rather easily. While specifications leading to undecidability are very artificial, no elegant solution to eliminate them exists at present.

We do not know whether the control problem for Zielonka automata is decidable for all architectures. The reason is that in our model, controllers can exchange all their causal memory and there is no artificially hidden information. Our result extends [6] where we showed that control is decidable with non-elementary complexity for a restricted form of reachability objectives (blocking final states). Incorporating all ω-regular objectives allows to express fairness constraints but at the same time introduces important technical obstacles. Indeed, for our construction to work it is essential to allow uncontrollable synchronisation actions. Such actions make a separation into controllable and uncontrollable states impossible: now there can be both controllable and uncontrollable actions outgoing from the same state. These extensions required to abandon the game metaphor, to invent new arguments, and to design a new proof structure.

Related work. The paper [9] gives an automata-theoretic approach to solving pipeline architectures and at the same time extends the decidability results to CTL specifications and variations of the pipeline architecture, like one-way ring architectures. The synthesis setting is investigated in [10] for local specifications, meaning that each process has its own, linear-time specification. For such specifications, it is shown that an architecture has a decidable synthesis problem if and only if it is a sub-architecture of a pipeline with inputs at both endpoints. The paper [3] proposes information forks as an uniform notion explaining the (un)decidability results in distributed synthesis. In [7] the authors consider distributed control by adding communication in order to combine local knowledge. The paper [5] studies external specifications on strongly connected architectures where processes communicate via signals.

Apart from [6], two closely related decidability results for synthesis with causal memory are known, both of different flavor than ours. The first one [4] restricts the alphabet of actions: control with reachability condition is decidable for co-graph alphabets. This restriction excludes among others client-server architectures, which are captured by our setting. The second result [11] shows decidability by restricting the plant: roughly speaking, the restriction says that every process can have only bounded missing knowledge about the other processes, unless they diverge (see also [15] that shows a doubly exponential upper bound). The proof of [11] goes beyond the controller synthesis problem, by coding it into monadic second-order theory of event structures and showing that this theory is decidable when the criterion on the plant holds. Unfortunately, very simple plants have a decidable control problem but undecidable MSO-theory of the associated event structure. Melliès [13] relates game semantics and asynchronous games, played on event structures. More recent work [8] considers games on event structures and shows a Borel determinacy result for such games under some restrictions.

Overview. In Section 2 we state our control problem, and in Section 3 we give the main lines of the proof. Finally, Section 5 shows an application of our main result to the control of multi-threaded programs.

2 Control for Zielonka automata

We state our control problem as a variant of the Ramadge and Wonham formulation [17]. So we are given an alphabet Σ of actions partitioned into system and environment actions: $\Sigma^{sys} \cup \Sigma^{env} = \Sigma$. Given a plant $\mathcal{A}$ we are asked to find a controller $\mathcal{C}$ such that the product $\mathcal{A} \times \mathcal{C}$ satisfies a given specification. Here both the plant and the controller are finite deterministic automata over Σ. Additionally, the controller is required not to block environment actions, which in technical terms means that from every state of the controller there should be a transition on every action from Σ^{env}.

The definition of our problem will be the same with the difference that we will consider a distributed automaton model, Zielonka automata, instead of standard finite automata.

2.1 Zielonka automata

In this section we define our communicating automata model. We will make precise what is a maximal run of such an automaton. We also introduce correctness conditions for maximal runs. Finally, we present a notion of a product to two automata. As we will see later, our goal will be to find a controller such that when composed with a given automaton ensures correctness of all maximal runs.

Zielonka automata are simple distributed finite-state devices. Such an automaton is a parallel composition of several finite automata, called *processes*, synchronizing on shared actions. There is no global clock, so between two synchronizations, two processes can do a different number of actions. Because of this Zielonka automata are also called asynchronous automata.

A *distributed action alphabet* on a finite set $\mathbb{P}$ of processes is a pair (Σ, dom), where Σ is a finite set of *actions* and $dom : \Sigma \to (2^{\mathbb{P}} \setminus \emptyset)$ is a *location function*. The location $dom(a)$ of action $a \in \Sigma$ comprises all processes that need to synchronize in order to perform this action. Actions from $\Sigma_p = \{a \in \Sigma \mid p \in dom(a)\}$ are called *p-actions*. We write $\Sigma_p^{loc} = \{a \mid dom(a) = \{p\}\}$ for the set of *local* actions of p.

A (deterministic) *Zielonka automaton* $\mathcal{A} = \langle \{S_p\}_{p\in\mathbb{P}}, s_{in}, \{\delta_a\}_{a\in\Sigma} \rangle$ is given by:

- for every process p a finite set S_p of (local) states,
- the initial state $s_{in} \in \prod_{p\in\mathbb{P}} S_p$,
- for every action $a \in \Sigma$ a partial transition function $\delta_a : \prod_{p\in dom(a)} S_p \dot{\to} \prod_{p\in dom(a)} S_p$ on tuples of states of processes in $dom(a)$.

For convenience, we abbreviate a tuple $(s_p)_{p\in P}$ of local states by s_P, where $P \subseteq \mathbb{P}$. We also talk about S_p as the set of *p-states* and of $\prod_{p\in\mathbb{P}} S_p$ as *global states*.

A Zielonka 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 $(s_{dom(a)}, s'_{dom(a)}) \in \delta_a$, and $s_{\mathbb{P}\setminus dom(a)} = s'_{\mathbb{P}\setminus dom(a)}$. So the states of the sequential automaton are the tuples of states of the processes of the Zielonka automaton. For a process p we will talk about the *p-component* of the state. Notice that the automaton $\mathcal{A}$ satisfies the following properties, for every $s, s' \in S$ and $a, b \in \Sigma$ such that $dom(a) \cap dom(b) = \emptyset$:

- (diamond) $s \xrightarrow{ab} s'$ iff $s \xrightarrow{ba} s'$,

- (forward diamond) if $a, b \in \Sigma$ are both enabled in s and $s \xrightarrow{a} s'$, then b is enabled in s'.

The idea of describing concurrency by a fixed independence relation on actions goes back to the late seventies, to Mazurkiewicz [12] (see also [2]). By $dom(u)$ we denote below the union of $dom(a)$, for all $a \in \Sigma$ occurring in u.

Definition 2.1 (Maximal run) *For a word $w \in \Sigma^\infty$ such that $run(w)$ is defined, we say that $run(w)$ is* maximal *if there is no $a \in \Sigma$ such that we can write $w = uv$ for some $v \in \Sigma^\infty$ with $dom(v) \cap dom(a) = \emptyset$, and $run(uav)$ is defined.*

The above definition says that a run is maximal if processes that have only finitely many actions in the run cannot perform any additional action. The definition conforms to the fact that a run can be extended on different processes, since our automata are distributed devices.

Automata can be equipped with a *correctness condition*. We prefer to talk about correctness condition rather than acceptance condition since we will be interested in the set of runs of an automaton rather than in the set of words it accepts. We will consider local correctness conditions: every process has its own correctness condition $Corr_p$. A run of $\mathcal{A}$ is *correct* if for every process p, the projection of the run on the transitions of $\mathcal{A}_p$ is in $Corr_p$. A particular example of $Corr_p$ we will work with consists of *regular* conditions, specified as a set $T_p \subseteq S_p$ of terminal states and an ω-regular set $\Omega_p \subseteq (S_p \times \Sigma_p \times S_p)^\omega$. A sequence $(s_p^0, a_0, s_p^1)(s_p^1, a_1, s_p^2)\ldots$ satisfies $Corr_p$ if either

- it is finite and ends with a state from T_p, or
- it is infinite and belongs to Ω_p.

At this stage the set of terminal states T_p may look unnecessary, but it will simplify our constructions later.

Finally, we will need the notion of synchronized product $\mathcal{A} \times \mathcal{C}$ of two Zielonka automata. For $\mathcal{A} = \langle \{S_p\}_{p\in\mathbb{P}}, s_{in}, \{\delta_a^A\}_{a\in\Sigma}\rangle$ and $\mathcal{C} = \langle \{C_p\}_{p\in\mathbb{P}}, c_{in}, \{\delta_a^C\}_{a\in\Sigma}\rangle$ let $\mathcal{A} \times \mathcal{C} = \langle \{S_p \times C_p\}_{p\in\mathbb{P}}, (s_{in}, c_{in}), \{\delta_a^\times)_{a\in\Sigma}\}\rangle$ where there is a transition from $(s_{dom(a)}, c_{dom(a)})$ to $(s'_{dom(a)}, c'_{dom(a)})$ in $\delta_a^\times$ iff $(s_{dom(a)}, s'_{dom(a)}) \in \delta_a^A$ and $(c_{dom(a)}, c'_{dom(a)}) \in \delta_a^C$.

2.2 Control for Zielonka automata

Consider a distributed alphabet $\langle \mathbb{P}, dom : \Sigma \to (2^{\mathbb{P}} \setminus \emptyset)\rangle$. We partition Σ into the set of system actions Σ^{sys} and environment actions Σ^{env}. Below we will introduce the notion of controller, and require that it does not block environment actions. For this reason will also talk about controllable/uncontrollable actions when referring to system/environment actions. We impose three simplifying assumptions:

1. All actions are at most binary: $|dom(a)| \leq 2$, for every $a \in \Sigma$.
2. Every process has a controllable action: $\Sigma_p \cap \Sigma^{sys}$ is non-empty for every p.
3. All controllable actions are local: $|dom(a)| = 1$, for every $a \in \Sigma^{sys}$.

The first condition is indeed a restriction of our setting. The second condition is easy to satisfy by extending the alphabet with dummy control actions. The third condition is also just for simplicity of presentation. We will see later how controllable communication can be simulated by a local controllable choice, followed by non-controllable local or shared actions.

Definition 2.2 (Controller, Correct Controller) *A* controller *is a Zielonka automaton that cannot block environment (uncontrollable) actions. In other words, from every state every environment action is possible: for every $b \in \Sigma^{env}$, δ_b is a total function. We say that a controller $\mathcal{C}$* is correct for *$\mathcal{A}$ if all maximal runs of $\mathcal{A} \times \mathcal{C}$ satisfy the correctness condition of $\mathcal{A}$.*

The correctness of $\mathcal{C}$ means that all the runs of $\mathcal{A}$ that are allowed by the controller are correct. In particular, $\mathcal{C}$ does not have a correctness condition by itself. Considering only maximal runs of $\mathcal{A} \times \mathcal{C}$ imposes some minimal fairness conditions: for example it implies that if a process can do a local action almost always, then it will eventually do some action.

Our control problem can be formulated as follows:

Definition 2.3 (Control problem) *Given a distributed alphabet $\langle \mathbb{P}, dom : \Sigma \to (2^{\mathbb{P}} \setminus \emptyset)\rangle$ together with a partition of actions $(\Sigma^{sys}, \Sigma^{env})$, and given a Zielonka automaton $\mathcal{A}$ over this alphabet, does there exist a controller $\mathcal{C}$ over the same alphabet such that $\mathcal{C}$ is correct for $\mathcal{A}$.*

The important point in our definition is that the controller should have the same distributed structure as the environment. The product of the two automata means that plant and controller are totally synchronized, in particular communications between processes happen at the same time. Hence concurrency in the controlled system is the same as in the plant. The major difference between the controlled system and the plant is that the states of the controller carry the additional information computed by the controller. Zielonka automata use rendez-vous model with complete sharing of information between processes participating in a rendez-vous. So our controllers can use causal memory.

3 Decidability for acyclic architectures

The goal is to show decidability of the control problem for Zielonka automata for acyclic architectures. A communication architecture is a graph where nodes are processes and edges link processes that have common actions. An acyclic architecture is one whose communication graph is acyclic.

Theorem 3.1 *Let $\langle \mathbb{P}, dom : \Sigma \to (2^{\mathbb{P}} \setminus \emptyset)\rangle$ be a distributed alphabet with acyclic architecture. The control problem for Zielonka automata over this alphabet is decidable. If a controller exists, then it can be effectively constructed.*

Our constructive procedure to solve the control problem will work by induction on the number of processes in the automaton. A Zielonka automaton over a single process is just a finite automaton, and the control problem is then just the standard control problem as considered by Ramadge and Wonham, but extended to all ω-regular conditions [1]. If there are several processes that do not communicate, then we can solve the problem for each process separately.

Otherwise we choose a leaf process r and its parent q (cf. Figure 1). We will construct a new automaton $\mathcal{A}^{\nabla}$ over $\mathbb{P}\setminus\{r\}$, where r is "glued together" with q. The control problem for $\mathcal{A}$ will have a solution iff the one for $\mathcal{A}^{\nabla}$ will have. Moreover, for every solution for $\mathcal{A}^{\nabla}$ we will be able to construct a solution for $\mathcal{A}$.

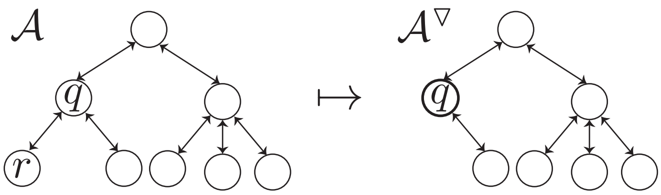

Figure 1: Eliminating process r: r is glued with q.

For the rest of this section let us fix (1) the leaf process r and its parent q, and (2) a Zielonka automaton with a correctness condition

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

We will first simplify the problem by requiring some additional properties from $\mathcal{A}$, and show in Section 4 how to satisfy these requirements.

Definition 3.2 (r-short, r-local strategy) *Automaton $\mathcal{A}$ is* r-short *if there is a bound on the number of actions that r can perform without doing a communication with q. An* r-local strategy *is a partial function $f : (S_r)^* \to \Sigma_r^{sys}$ mapping sequences of r-states to controllable r-actions, such that if $f(v) = a$ then a is possible from the last state in the sequence v.*

Observe that if the automaton is r-short then the domain of f is finite.

Before we start we also introduce the notion of covering controller. This notion will simplify the presentation because it will allow us to focus on the runs of the controller instead of building the product of the given automaton and the controller.

Definition 3.3 (Covering controller) *Let $\mathcal{C}$ be a Zielonka automaton over the same alphabet as $\mathcal{A}$; let C_p be the set of states of process p in $\mathcal{C}$. Automaton $\mathcal{C}$ is a* covering controller *for $\mathcal{A}$ if there is a function $\pi : \{C_p\}_{p\in\mathbb{P}} \to \{S_p\}_{p\in\mathbb{P}}$, mapping each C_p to S_p and satisfying two conditions: (i) if $c_{dom(b)} \xrightarrow{b} c'_{dom(b)}$ then $\pi(c_{dom(b)}) \xrightarrow{b} \pi(c'_{dom(b)})$; (ii) for every uncontrollable action a: if a is enabled from $\pi(c_{dom(a)})$ then it is also enabled from $c_{dom(a)}$.*

Remark 3.4 *The first condition on the covering controller ensures that all its runs are relevant, the second condition says that it cannot block relevant uncontrollable actions. Strictly speaking, a covering controller $\mathcal{C}$ may not be a controller since we do not require that every uncontrollable action is enabled in every state, but only those actions that are enabled in $\mathcal{A}$. We define $\hat{\mathcal{C}}$ by adding to every state of $\mathcal{C}$ a self-loop on every uncontrollable action missing from that state.*

Notice that thanks to the projection π, a covering controller can inherit the correctness condition of $\mathcal{A}$. Moreover, the sequences labeling the maximal runs of $\mathcal{C}$, $\mathcal{A} \times \mathcal{C}$ and $\mathcal{A} \times \hat{\mathcal{C}}$ are the same.

Lemma 3.5 *There is a correct controller for $\mathcal{A}$ if and only if there is a covering controller $\mathcal{C}$ for $\mathcal{A}$ such that all the maximal runs of $\mathcal{C}$ satisfy the inherited correctness condition.*

Proof. If $\mathcal{C}$ is a covering controller for $\mathcal{A}$ such that all its maximal runs satisfy the inherited correctness condition then $\hat{\mathcal{C}}$ is a correct controller for $\mathcal{A}$.

Conversely, if $\mathcal{C}$ is a correct controller for $\mathcal{A}$ then $\mathcal{A} \times \mathcal{C}$ is a covering controller where all maximal runs satisfy the inherited correctness condition. □

Even though strictly speaking a covering controller is not a controller, we will refer to a covering controller with the property that all its maximal runs satisfy the inherited correctness condition, as *correct covering controller*.

We end the section by showing the assumption that controllable actions are local, is not a restriction.

Proposition 3.6 *The control problem for Zielonka automata where communication actions may be controllable, reduces to the setting where controllable actions are all local.*

Proof. We start with an automaton $\mathcal{A}$ over a distributed alphabet $\langle \Sigma, dom \rangle$ and a correct covering controller $\mathcal{C}$. We define first a new automaton $\mathcal{A}'$ over an extended distributed alphabet $\langle \Sigma', dom' \rangle$ with $\Sigma' = \Sigma \cup \{ch(A) \mid A \subseteq \Sigma_p^{sys} \text{ for some } p \in \mathbb{P}\}$. All new actions are local: $dom'(ch(A)) = \{p\}$ if $A \subseteq \Sigma_p^{sys}$; the domain of other actions do not change. What changes is that all old actions become uncontrollable, and the only controllable actions in Σ' are those of the form $ch(A)$.

- The set of p-states of $\mathcal{A}'$ is the set of p-states of $\mathcal{A}$, plus some new states of the form $\langle s_p, A \rangle$ where s_p is a p-state of $\mathcal{A}$ and $A \subseteq \Sigma_p^{sys}$.
- For every old p-state s_p we delete all outgoing controllable transitions and add
$$s_p \xrightarrow{ch(A)} \langle s_p, A \rangle ,$$
for every set A of controllable actions enabled in s_p. From $\langle s_p, A \rangle$ we put in $\mathcal{A}'$ transitions as follows. If $a \in A$ is local then we have $\langle s_p, A \rangle \xrightarrow{a} s'_p$ whenever $s_p \xrightarrow{a} s'_p$ in $\mathcal{A}$. If $a \in A \cap B$ and $dom(a) = \{p, p'\}$ then we have $(\langle s_p, A \rangle, \langle s_{p'}, B \rangle) \xrightarrow{a} (s'_p, s'_{p'})$ whenever $(s_p, s_{p'}) \xrightarrow{a} (s'_p, s'_{p'})$ in $\mathcal{A}$.
- The correctness condition of $\mathcal{A}'$ is a straightforward modification of the one of $\mathcal{A}$.

Assume first that $\mathcal{C}'$ is a correct covering controller for $\mathcal{A}'$. From $\mathcal{C}'$ we define the automaton $\mathcal{C}$ over the same sets of states, by modifying slightly the transitions as follows. Suppose that $c \xrightarrow{ch(A)} d$ is a (local) transition in $\mathcal{C}'_p$. Since $\mathcal{C}'$ is covering we have a transition of the form $s_p = \pi'(c) \xrightarrow{ch(A)} \pi'(d) = \langle s_p, A \rangle$ in $\mathcal{A}'$. Let $a \in A$ be local. Since a is uncontrollable in $\mathcal{A}'$ and $\langle s_p, A \rangle \xrightarrow{a} s'_p$ (for some s'_p) we must also have $d \xrightarrow{a} e$ for some state e of $\mathcal{C}'_p$, since $\mathcal{C}'$ is covering. We delete $c \xrightarrow{ch(A)} d$ from $\mathcal{C}'$ and replace $d \xrightarrow{a} e$ by $c \xrightarrow{a} e$. If a is shared by p, p', let us consider some transition $c' \xrightarrow{ch(B)} d'$ with $a \in B$ in $\mathcal{C}'_{p'}$. Since a is uncontrollable in $\mathcal{A}'$ we find again some transition $(d, d') \xrightarrow{a} (e, e')$ in $\mathcal{C}'$. We replace then $(d, d') \xrightarrow{a} (e, e')$ by $(c, c') \xrightarrow{a} (e, e')$ in $\mathcal{C}$. Of course, this is done in parallel for all transitions labelled by some $ch(A)$. It is immediate that $\mathcal{C}$ is covering $\mathcal{A}$, by taking $\pi = \pi'$. Maximal runs of $\mathcal{C}$ map to maximal runs of $\mathcal{C}'$ and thus satisfy the correctness condition for $\mathcal{A}$.

Conversely, given a correct covering controller $\mathcal{C}$ for $\mathcal{A}$ we define $\mathcal{C}'$ for $\mathcal{A}'$. Local p-states of $\mathcal{C}'$ are those of $\mathcal{C}$, plus additional states of the form c_A, where c is a p-state of $\mathcal{C}$ and $A \subseteq \Sigma_p^{sys}$. Consider any p-state c of $\mathcal{C}$, and let A be the set of controllable actions enabled in c (a communication action a with $dom(a) = \{p, p'\}$ is enabled in c if there exists some p'-state c' and an a-transition from (c, c')). We replace all controllable transitions from c by one (local) controllable transition $c \xrightarrow{ch(A)} c_A$, plus some uncontrollable transitions. If $a \in A$ is local, then we add the uncontrollable transitions $c_A \xrightarrow{a} d$ whenever $c \xrightarrow{a} d$ in $\mathcal{C}$. If $dom(a) = \{p, p'\}$, $(c, c') \xrightarrow{a} (d, d')$ in $\mathcal{C}$, and B is the set of controllable actions enabled in the p'-state c', then we replace $(c, c') \xrightarrow{a} (d, d')$ by $(c_A, c'_B) \xrightarrow{a} (d, d')$. Extending π by $\pi'(c_A) = \langle \pi(c), A \rangle$ shows that $\mathcal{C}'$ is a covering controller for $\mathcal{A}'$. Maximal runs of $\mathcal{C}'$ satisfy the acceptance condition as for $\mathcal{C}$. $\square$

3.1 The new plant $\mathcal{A}^{\triangledown}$

We suppose in the following that $\mathcal{A} = \langle \{S_p\}_{p \in \mathbb{P}^{\triangledown}}, s_{in}, \{\delta_a\}_{a \in \Sigma} \rangle$ is r-short and we define the reduced automaton $\mathcal{A}^{\triangledown}$ that is the result of eliminating process r (cf. Figure 1). Let $\mathbb{P}^{\triangledown} = \mathbb{P} \setminus \{r\}$. We construct $\mathcal{A}^{\triangledown} = \langle \{S_p^{\triangledown}\}_{p \in \mathbb{P}^{\triangledown}}, s_{in}^{\triangledown}, \{\delta_a^{\triangledown}\}_{a \in \Sigma^{\triangledown}} \rangle$ where the components are defined below.

All the processes $p \neq q$ of $\mathcal{A}^{\triangledown}$ will be the same as in $\mathcal{A}$. This means: $S_p^{\triangledown} = S_p$, and $\Sigma_p^{\triangledown} = \Sigma_p$. Moreover, all transitions δ_a with $dom(a) \cap \{q, r\} = \emptyset$ are as in $\mathcal{A}$. Finally, in $\mathcal{A}^{\triangledown}$ the correctness condition of $p \neq q$ is the same as in $\mathcal{A}$.

The states of process q in $\mathcal{A}^{\triangledown}$ are of one of the following types:

$$\langle s_q, s_r \rangle\,, \quad \langle s_q, s_r, f \rangle\,, \quad \langle s_q, a, s_r, f \rangle\,,$$

where $s_q \in S_q$, $s_r \in S_r$, f is an r-local strategy (see Definition 3.2), $a \in \Sigma_q^{loc}$. The new initial state for q is $\langle (s_{in})_q, (s_{in})_r \rangle$. Recall that that since $\mathcal{A}$ is r-short, any r-local strategy in $\mathcal{A}_r$ is necessarily finite, so $S_q^{\triangledown}$ is a finite set.

The set $\Sigma_q^{\triangledown}$ of actions of q is introduced below, together with the transitions. An example of a simulation of $\mathcal{A}_q$ and $\mathcal{A}_r$ by $\mathcal{A}_q^{\triangledown}$ is presented in Figure 2.

1. Choose a local strategy for r:

$$\langle s_q, s_r \rangle \xrightarrow{ch(f)} \langle s_q, s_r, f \rangle\,, \qquad \text{if } f \text{ is } r\text{-local strategy.}$$

2. Declare a (local) controllable action:

$$\langle s_q, s_r, f \rangle \xrightarrow{ch(a)} \langle s_q, a, s_r, f \rangle \qquad \text{if } a \in \Sigma_q \text{ controllable and enabled in } s_q.$$

3. Execute a:

$$\langle s_q, a, s_r, f \rangle \xrightarrow{a} \langle s'_q, s_r, f \rangle \qquad \text{if } s_q \xrightarrow{a} s'_q.$$

4. Local uncontrollable action of q:

$$\langle s_q, a, s_r, f \rangle \xrightarrow{b} \langle s'_q, s_r, f \rangle, \quad \text{if } s_q \xrightarrow{b} s'_q \text{ with } b \in \Sigma_q \text{ uncontrollable.}$$

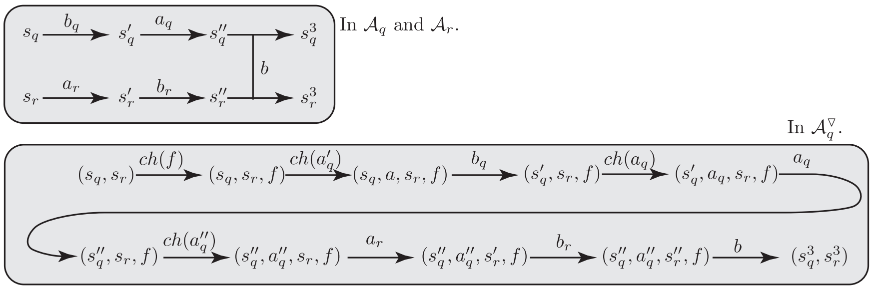

Figure 2: Simulation of $\mathcal{A}_q$ and $\mathcal{A}_r$ by $\mathcal{A}^{\nabla}_q$.

5. Synchronization between q and $p \neq r$

$$(s_p, \langle s_q, a, s_r, f\rangle) \xrightarrow{b} (s'_p, \langle s'_q, s_r, f\rangle) \quad \text{if } (s_p, s_q) \xrightarrow{b} (s'_p, s'_q).$$

   Here $b \in \Sigma_p \cap \Sigma_q$ is uncontrollable (since all communication actions are uncontrollable).

6. Local action of r:

$$\langle s_q, a, s_r, f\rangle \xrightarrow{b} \langle s_q, a, s'_r, f\rangle, \qquad \text{if } s_r \xrightarrow{b} s'_r \text{ and allowed by } f.$$

7. Synchronization between q and r:

$$\langle s_q, a, s_r, f\rangle \xrightarrow{b} \langle s'_q, s'_r\rangle \qquad \text{if } (s_q, s_r) \xrightarrow{b} (s'_q, s'_r).$$

   Here $b \in \Sigma_q \cap \Sigma_r$, it is uncontrollable.

To summarize, in Σ^{∇}_q we have all actions of Σ_r and Σ_q, but they become uncontrollable. The new actions of process q in plant $\mathcal{A}^{\nabla}$ are:

- controllable action $ch(f) \in \Sigma^{sys}$, for every local r-strategy f,
- controllable action $ch(a)$, for every controllable $a \in \Sigma_q$.

The correctness condition for process q in $\mathcal{A}^{\nabla}$ is:

1. The correct infinite runs of q in $\mathcal{A}^{\nabla}$ are those that have the projection on transitions of $\mathcal{A}_q$ correct with respect to $Corr_q$, and either: (i) the projection on transitions of $\mathcal{A}_r$ is infinite and correct with respect to $Corr_r$; or (ii) the projection on transitions of $\mathcal{A}_r$ is finite and for f, s_r appearing in almost all states of q of the run we have that from s_r all sequences respecting strategy f end in a state from T_r.
2. T^{∇}_q contains states $\langle s_q, s_r, f\rangle$ such that $s_q \in T_q$, and $s_r \in T_r$.

Item $1(ii)$ in the definition of correct runs is to cater for the case where q progresses alone till infinity and blocks r, even though r could reach a terminal state in a couple of moves. Clearly, item 1 can be expressed as an ω-regular condition.

Observe also the need to schedule actions of process q, using $ch(a)$ actions, before the actions of process r. The reason is the following. First, we need to make all r-actions uncontrollable, so that the environment could choose any play respecting the chosen r-local strategy. If we allowed controllable q-actions at the same time as r actions then the strategy for automaton $\mathcal{A}^\triangledown$ would be to propose nothing and force the environment to play r-actions. This would allow the controller for $\mathcal{A}^\triangledown$ to get information that is impossible to obtain in $\mathcal{A}$.

The correctness of this construction is stated in the theorem below, whose proof will be given in the next two subsections.

Theorem 3.7 *For every r-short Zielonka automaton $\mathcal{A}$ and every local, ω-regular correctness conditions: there is a correct controller for $\mathcal{A}$ iff there is a correct controller for $\mathcal{A}^\triangledown$.*

3.2 Correctness of the reduction

3.2.1 From $\mathcal{C}$ to $\mathcal{C}^\triangledown$

We want to construct from a correct covering controller for $\mathcal{A}$ a correct covering controller for $\mathcal{A}^\triangledown$. We start with some notation.

Definition 3.8 **(*run*, *state*, *run*$^\triangledown$, *state*$^\triangledown$)** *Given two Zielonka automata $\mathcal{C}$ over (Σ, dom) and $\mathcal{C}^\triangledown$ over $(\Sigma^\triangledown, loc)$. For $w \in \Sigma^\infty$ we denote by $run(w)$ the sequence of transitions of $\mathcal{C}$ when reading w; observe that $run(w)$ may not be defined since $\mathcal{C}$ may not be complete. For finite w let $state(w)$ be the last state in $run(w)$. By $run_p(w)$ we denote the projection of $run(w)$ on transitions of process p. Similarly we define $run^\triangledown(w')$, $state^\triangledown(w')$ and $run_p^\triangledown(w')$ for $\mathcal{C}^\triangledown$ and $w' \in (\Sigma^\triangledown)^\infty$.*

By Lemma 3.5 we can assume that we have a *correct covering controller* $\mathcal{C}$ for $\mathcal{A}$. We show how to construct a correct controller $\mathcal{C}^\triangledown$ for $\mathcal{A}^\triangledown$. This will give the left to right implication of Theorem 3.7.

Remark 3.9 *Some simple observations about $\mathcal{C}$.*

1. *We may assume that from every state of C there is at most one transition on a local controllable action. If there were more than one, we could arbitrary remove one of them. This will reduce the number of maximal runs so the resulting controller with stay correct.*

2. $\mathcal{C}$ determines for every state c of $\mathcal{C}_r$ a local r-strategy f from $\pi(c)$: if $c = c_0 \xrightarrow{a_1} c_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} c_k$, $\pi(c_i) = s_i$ and $a_i \in \Sigma_r^{loc}$ for all i, then $f(c_0 \cdots c_k) = a$ where $a \in \Sigma_r^{sys}$ is a (unique) controllable action possible from c_k. This strategy may have memory, but all the (local) plays respecting f are of bounded length, assuming that $\mathcal{A}$ is r-short.

The components $\mathcal{C}_p^\triangledown$ for $p \neq q$ are just $\mathcal{C}_p$, and the initial state is the same. The component $\mathcal{C}_q^\triangledown$ is described below. Its states are of the form (c_q, c_r), (c_q, c_r, f) and (c_q, a, c_r, f) with $c_q \in C_q$, $c_r \in C_r$, $a \in \Sigma_q^{sys}$, and local r-strategy f. Its initial state is (c_q^0, c_r^0), with c_q^0, c_r^0 initial states of $\mathcal{C}_q, \mathcal{C}_r$.

The transitions of $\mathcal{C}_q^\triangledown$ ensure the right choice of a local strategy and of a local action:

- Choice of r-strategy:

  $$(c_q, c_r) \xrightarrow{ch(f)} (c_q, c_r, f)$$

  where f is the local r-strategy from $\pi(c_r)$ determined by $\mathcal{C}$ in state c_r.

- Choice of a (local) controllable q-action:

  $$(c_q, c_r, f) \xrightarrow{ch(a)} (c_q, a, c_r, f)$$

  For $a \in \Sigma_q^{sys}$ unique such that $c_q \xrightarrow{a} c'_q$, for some c'_q. If there is no such transition then we put some arbitrary fixed action $a_0 \in \Sigma_q^{sys}$.

The other transitions of $\mathcal{C}_q^\triangledown$ are on uncontrollable actions, they just reflect the structure of $\mathcal{A}^\triangledown$:

- Execution of the chosen controllable q-action:

  $$(c_q, a, c_r, f) \xrightarrow{a} (c'_q, c_r, f) \qquad \text{if } c_q \xrightarrow{a} c'_q \text{ in } \mathcal{C}_q.$$

- Execution of an uncontrollable local q-action:

  $$(c_q, a, c_r, f) \xrightarrow{b} (c'_q, c_r, f) \qquad \text{if } c_q \xrightarrow{b} c'_q \text{ in } \mathcal{C}_q, \text{ where } b \in \Sigma_q^{env} \cap \Sigma_q^{loc}.$$

- Communication between q and $p \neq r$:

  $$(c_p, (c_q, a, c_r, f)) \xrightarrow{b} (c'_p, (c'_q, c_r, f)) \quad \text{if } (c_p, c_q) \xrightarrow{b} (c'_p, c'_q) \text{ in } \mathcal{C}.$$

- Local move of r:

  $$(c_q, a, c_r, f) \xrightarrow{b} (c_q, a, c'_r, f) \quad \text{if } c_r \xrightarrow{b} c'_r \text{ in } \mathcal{C}_r, \text{ where } b \in \Sigma_r^{loc}.$$

- Communication between q and r

$$(c_q, a, c_r, f) \xrightarrow{b} (c'_q, c'_r) \quad \text{if } (c_q, c_r) \xrightarrow{b} (c'_q, c'_r) \text{ in } \mathcal{C}.$$

Lemma 3.10 *If C is a covering controller for $\mathcal{A}$ then $\mathcal{C}^\triangledown$ is a covering controller for $\mathcal{A}^\triangledown$. The covering function is*

$$\begin{aligned}
\pi^\triangledown(c_q, c_r) =&(\pi(c_q), \pi(c_r))\\
\pi^\triangledown(c_q, c_r, f) =&(\pi(c_q), \pi(c_r), f)\\
\pi^\triangledown(c_q, a, c_r, f) =&(\pi(c_q), a, \pi(c_r), f)\ .
\end{aligned}$$

For the correctness proof we will need one more definition:

Definition 3.11 (*hide*) *For $w \in (\Sigma^\triangledown)^\infty$ we let $hide(w) \in \Sigma^\infty$ be the sequence obtained by removing actions from $\Sigma^\triangledown \setminus \Sigma$.*

Observe that by construction of $\mathcal{C}^\triangledown$ if $run^\triangledown(w)$ is defined then in w there can be at most two consecutive q-actions from $\Sigma^\triangledown \setminus \Sigma$.

Lemma 3.12 *Let $w \in (\Sigma^\triangledown)^*$. If $run^\triangledown(w)$ is defined then so is $run(hide(w))$. Moreover, letting $c^\triangledown = state^\triangledown(w)$ and $c = state(hide(w))$, we have that (i) $c^\triangledown_p = c_p$ for all $p \neq q, r$, and (ii) $c^\triangledown_q$ is either (c_q, c_r), or (c_q, c_r, f), or (c_q, a, c_r, f); where a and f are determined by c_q and c_r as follows:*

- *a is the unique controllable q-action from c_q in $\mathcal{C}$ (or a_0 if there is none).*
- *f is the local r-strategy determined by $\mathcal{C}$ in c_r.*

Proof. The proof is by induction on the length of w. It follows by direct examination of the rules. □

Lemma 3.13 *Assume that $w \in (\Sigma^\triangledown)^\infty$. For every process $p \neq q$ we have $run^\triangledown_p(w) = run_p(hide(w))$. Concerning $run^\triangledown_q(w)$: if we project it on transitions of $\mathcal{C}_q$ we obtain $run_q(hide(w))$; if we project it on transitions of $\mathcal{C}_r$ we obtain $run_r(hide(w))$.*

Proof. Directly from the previous lemma. □

Lemma 3.14 *If $\mathcal{C}$ is a correct covering controller for $\mathcal{A}$ then $\mathcal{C}^\triangledown$ is a correct covering controller for $\mathcal{A}^\triangledown$.*

Proof. Since $\mathcal{C}$ is a correct covering controller we have that all maximal runs of $\mathcal{C}$ are correct w.r.t $\mathcal{A}$. By Lemma 3.10 we know that $\mathcal{C}^\triangledown$ is a covering controller, so it is enough to show that all maximal runs of $\mathcal{C}^\triangledown$ are correct w.r.t. $\mathcal{A}^\triangledown$.

Take a maximal run in $\mathcal{C}^\triangledown$, say on $w \in (\Sigma^\triangledown)^\infty$. The first obstacle is that $run(hide(w))$ may be not maximal in $\mathcal{C}$. This can only happen when there are infinitely many q-actions in w, but only finitely many r-actions. Then we have $w = v_1 v_2$ and there are no r-actions in v_2. Let $state^\triangledown_q(v_1) = (c_q, a, c_r, f)$. We have that c_r and f appear in all $state^\triangledown_q(v_1 v')$, for every prefix v' of v_2. The run $run(hide(w))$ is not maximal when there is at least some local action of $\mathcal{C}_r$ enabled in c_r. Let x be a maximal sequence of local r-actions that is possible in $\mathcal{C}_r$ from state c_r. Since $\mathcal{A}$ is r-short, every such sequence is finite. Moreover we choose x in such a way that it brings $\mathcal{C}_r$ into a state not in T_r (if it is possible). We get that $u = v_1 x v_2$ also defines a maximal run of $\mathcal{C}^\triangledown$, but now the run on $hide(u)$ is maximal in $\mathcal{C}$. Notice that $run^\triangledown(u)$ satisfies $Corr^\triangledown$ iff $run^\triangledown(w)$ does: the difference is the sequence x, and we have chosen, if possible, a losing sequence.

We need to show that the run of $\mathcal{A}^\triangledown$ on u satisfies $Corr^\triangledown$ using the fact that the run on $hide(u)$ satisfies $Corr$. For $p \neq q$, Lemma 3.13 tells us that $run^\triangledown_p(u)$ is the same as $run_p(hide(u))$. Since $Corr^\triangledown_p$ and $Corr_p$ are the same, we are done.

It remains to consider $run^\triangledown_q(u)$. If there are finitely many q-actions in $u \in (\Sigma^\triangledown)^\infty$ then $u = u_1 u_2$ with no q-action in u_2. Consider $state^\triangledown_q(u_1) = (c_q, a, c_r, f)$. We have that $state_q(hide(u_1)) = c_q$ and $state_r(hide(u_1)) = c_r$. As there are no q-actions in u_2, and $run_q(u)$ satisfies $Corr_q$, we must have $\pi(c_q) \in T_q$ and $\pi(c_r) \in T_r$. This shows that $run^\triangledown_q(u)$ satisfies $Corr^\triangledown_q$.

If there are infinitely many q-actions in $u \in (\Sigma^\triangledown)^\infty$, we still have two cases. The first is when there are infinitely many actions from Σ_r as well. Then $run^\triangledown_q(u)$ satisfies $Corr^\triangledown_q$ if the corresponding runs $run_q(hide(u))$ and $run_r(hide(u))$ satisfy $Corr_q$ and $Corr_r$, respectively. This is guaranteed by our assumption that $run(hide(u))$ satisfies $Corr$.

The last case is when in $u \in (\Sigma^\triangledown)^\infty$ we have infinitely many q-actions and only finitely many actions from Σ_r. Then $u = u_1 u_2$ with no actions from Σ_r in u_2. We get $state^\triangledown_q(u_1) = (c_q, a, c_r, f)$ with both c_r, f appearing in all the further states of the run. Since $run_r(hide(u))$ satisfies $Corr_r$, we have that $\pi(c_r) \in T_r$. But then, by the construction of u, there is no Σ_r-transition possible from c_r (and neither from $\pi(c_r)$ in $\mathcal{A}^\triangledown_r$, since $\mathcal{C}^\triangledown$ is covering). This means that $run^\triangledown_q(u)$ satisfies $Corr^\triangledown_q$.

□

3.3 From $\mathcal{D}^\triangledown$ to $\mathcal{D}$

Given a correct covering controller $\mathcal{D}^\triangledown$ for $\mathcal{A}^\triangledown$ we will construct a correct controller $\mathcal{D}$ for $\mathcal{A}$. This property is stated in Lemma 3.21 below. The lemma gives the right-to-left implication of Theorem 3.7.

The components $\mathcal{D}_p$ for $p \neq q, r$ will be the same as in $\mathcal{D}^\triangledown$. So it remains to define $\mathcal{D}_q$ and $\mathcal{D}_r$.

The states of $\mathcal{D}_q$ and $\mathcal{D}_r$ will be constructed from states of $\mathcal{D}^\triangledown_q$. We will need only certain states of $\mathcal{D}_q^\triangledown$, namely those d_q whose projection $\pi^\triangledown(d_q)$ has four components, we call them *true states* of $\mathcal{D}_q^\triangledown$:

$$ts(\mathcal{D}_q^\triangledown) = \{d_q \in \mathcal{D}_q^\triangledown \mid \pi^\triangledown(d_q) \text{ is of the form } (s_q, a, s_r, f)\}.$$

The set of states of $\mathcal{D}_q$ is just $ts(\mathcal{D}_q^\triangledown)$, while the states of $\mathcal{D}_r$ are pairs (d_q, x) where d_q is a state from $ts(\mathcal{D}_q^\triangledown)$ and $x \in (\Sigma_r^{loc})^*$ is a sequence of local r-actions that is possible from d_q in $\mathcal{D}^\triangledown$, in symbols $d_q \xrightarrow{x}$. We will show later that $\mathcal{D}_r$ is finite. The initial state of $\mathcal{D}_q$ is the state d_q^1 reached from the initial state of $\mathcal{D}_q^\triangledown$ by the (unique) transitions of the form $ch(f_0), ch(a_0)$. The initial state of $\mathcal{D}_r$ is (d_q^1, ε).

The local transitions for $\mathcal{D}_r$ are very easy to describe

$$(d_q, x) \xrightarrow{b} (d_q, xb) \qquad \text{if } b \in \Sigma_r^{loc} \text{ and } d_q \xrightarrow{xb}.$$

Before defining the transitions of $\mathcal{D}_q$ let us observe that if $d_q \in \mathcal{D}_q^\triangledown$ is not in $ts(\mathcal{D}_q^\triangledown)$ then only one controllable transition is possible from it. Indeed, as $\mathcal{D}^\triangledown$ is a covering controller, if $\pi^\triangledown(d_q)$ is of the form (s_q, s_r) then there can be only an outgoing transition on a letter of the form $ch(f)$. Similarly, if $\pi^\triangledown(d_q)$ is of the form (s_q, s_r, f) then only a $ch(a)$ transition is possible. Since both $ch(f)$ and $ch(a)$ are controllable, we can assume that in $\mathcal{D}_q^\triangledown$ there is no state with two outgoing transitions on a letter of this form. For a state $d_q \in \mathcal{D}_q^\triangledown$ not in $ts(\mathcal{D}_q^\triangledown)$ we will denote by $ts(d_q)$ the unique state of $ts(\mathcal{D}_q^\triangledown)$ reachable from d_q by one or two transitions of the kind $\xrightarrow{ch(f)}$ or $\xrightarrow{ch(a)}$, depending on the cases discussed above.

We now describe the q-actions possible in $\mathcal{D}$.

- Local action of q:

$$d_q \xrightarrow{b} ts(d'_q) \qquad \text{if } d_q \xrightarrow{b} d'_q \text{ in } \mathcal{D}_q^\triangledown,\ b \in \Sigma_q^{loc}.$$

- Communication between q and $p \neq r$:

$$(d_p, d_q) \xrightarrow{b} (d'_p, ts(d'_q)) \qquad \text{if } (d_p, d_q) \xrightarrow{b} (d'_p, d'_q) \text{ in } \mathcal{D}^\triangledown.$$

- Communication between q and r:

$$(d_q^1, (d_q^2, x)) \xrightarrow{b} (ts(d''_q), (ts(d''_q), \varepsilon))$$

  if $d_q^1 \xrightarrow{x} d'_q \xrightarrow{b} d''_q$ in $\mathcal{D}_q^\triangledown$, $b \in \Sigma_q \cap \Sigma_r$; observe that $\xrightarrow{x}$ is a sequence of transitions.

In the last item the transition does not depend on d_q^2. We will show later that with respect to x, that is a sequence of local r-actions, both d_q^1 and

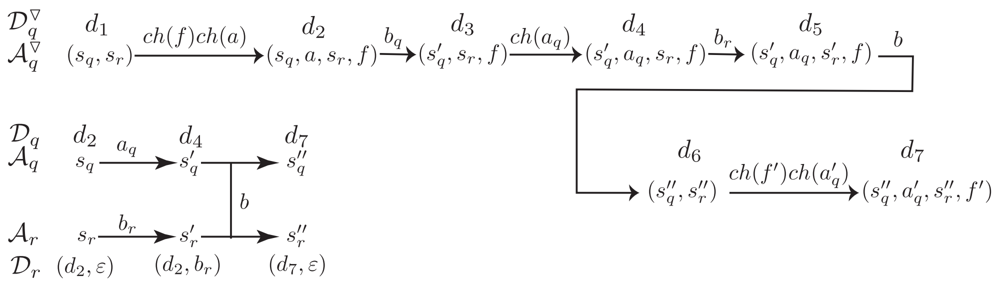

Figure 3: Decomposing controller $\mathcal{D}_q^\triangledown$ into $\mathcal{D}_q$ and $\mathcal{D}_r$.

d_q^2 carry the same information. Informally, d_q^1 has been reached from d_q^2 by a sequence of actions of q that are either local or shared with $p \neq r$. The condition $d_q^1 \xrightarrow{x} d'_q \xrightarrow{b} d''_q$ simulates the order of actions where all local r-actions come after the other actions of q, then we add a communication between q and r.

The next lemma says that $\mathcal{D}$ is a covering controller for $\mathcal{A}$. Since $\mathcal{A}$ is assumed to be r-short, the lemma also gives a bound on the length of sequences in the states of $\mathcal{D}_r$.

Lemma 3.15 *If $\mathcal{D}^\triangledown$ is a covering controller for $\mathcal{A}^\triangledown$ then $\mathcal{D}$ is a covering controller for $\mathcal{A}$.*

Proof. We need to define the projection function π using the projection function $\pi^\triangledown$. For $p \neq q, r$ set $\pi = \pi^\triangledown$. For $\mathcal{D}_q$ we define $\pi(d_q) = s_q$ where s_q is the state of $\mathcal{A}_q$ in $\pi^\triangledown(d_q)$. For $\mathcal{D}_r$ and its state (d_q, x) we define $\pi(d_q, x) = s'_r$ where s'_r is the state of $\mathcal{A}_r$ in $\pi^\triangledown(d'_q)$, and $d_q \xrightarrow{x} d'_q$.

We need to check that the transitions defined above preserve this projection function; namely for every process p: if $d_p \xrightarrow{b} d'_p$ in $\mathcal{D}_p$ then $\pi(d_p) \xrightarrow{b} \pi(d'_p)$ in $\mathcal{A}_p$; and similarly for communication actions. The statement is obvious if the move is in components other than q or r. We are left with four cases:

- Local move of q, namely $d_q \xrightarrow{b} d'_q$. We have $d_q \xrightarrow{b} d''_1 \xrightarrow{ch(a')} d'_q$ in $\mathcal{D}^\triangledown$ for some a', since $d'_q = ts(d''_q)$. By the fact that $\mathcal{D}^\triangledown$ covers $\mathcal{A}^\triangledown$ and the definition of moves of the latter automaton we have in $\mathcal{A}^\triangledown$:

$$\pi^\triangledown(d_q) = \langle s_q, a, s_r, f\rangle \xrightarrow{b} \langle s'_q, s_r, f\rangle \xrightarrow{ch(a')} \langle s'_q, a', s_r, f\rangle = \pi^\triangledown(d'_q)\,,$$

  and by definition of $\mathcal{A}^\triangledown$ we know that $s_q \xrightarrow{b} s'_q$ is in $\mathcal{A}$.

- Communication between q and $p \neq r$ is similar.

- Local move of r: $(d_q, x) \xrightarrow{b} (d_q, xb)$. By definition we know that from d_q it is possible to do in $\mathcal{D}^\triangledown$ the sequence of actions xb, that is $d_q \xrightarrow{x} d_q^1 \xrightarrow{b} d_q^2$. We have $\pi^\triangledown(d_q) = (s_q, a, s_r, f)$, $\pi^\triangledown(d_q^1) = (s_q, a, s_r^1, f)$ and $\pi^\triangledown(d_q^2) = (s_q, a, s_r^2, f)$; since xb is a sequence of local r-actions the other components do not change. We have $s_r^1 \xrightarrow{b} s_r^2$ by definition of $\mathcal{A}_q^\triangledown$, and $\pi^\triangledown(d_q, x) = s_r^1$, $\pi^\triangledown(d_q, xb) = s_r^2$, as required.

- Communication between q and r: $(d_q^1, (d_q^2, x)) \xrightarrow{b} (ts(d_q''), (ts(d_q''), \varepsilon))$. By definition this is possible only when $d_q^1 \xrightarrow{x} d_q' \xrightarrow{b} d_q''$ in $\mathcal{D}_q^\triangledown$. Since $\mathcal{D}^\triangledown$ is covering we get the following sequence of transitions in $\mathcal{A}^\triangledown$:

$$\pi^\triangledown(d_q^1) = \langle s_q, a, s_r, f\rangle \xrightarrow{x} \langle s_q, a, s_r^1, f\rangle \xrightarrow{b} \langle s_q', s_r'\rangle \xrightarrow{ch(f')}$$
$$\langle s_q', s_r', f'\rangle \xrightarrow{ch(a')} \langle s_q', a', s_r', f'\rangle = \pi^\triangledown(ts(d_q''))$$

So we have $(s_q, s_r^1) \xrightarrow{b} (s_q', s_r')$ in $\mathcal{A}$ and $\pi(d_q^1) = s_q$, $\pi(ts(d_q'')) = s_q'$, $\pi(ts(d_q''), \varepsilon) = s_r'$. We claim that $\pi(d_q^2, x) = s_r^1$, and for this we need to observe a property of the runs of $\mathcal{D}$ (proved by induction on the length of the run).

> **Property (*)** If from the initial state $\mathcal{D}$ can reach a global state of with d_q and (d_q', x) at the coordinates corresponding to q and r, respectively, then the s_r and f components of $\pi^\triangledown$ projections of d_q and d_q' are the same: $\pi^\triangledown(d_q) = (s_q, a, s_r, f)$ and $\pi^\triangledown(d_q') = (s_q', a', s_r, f)$, for some s_q, s_q', a, a', s_r, f.

From Property (*) it follows that $\pi(d_q^2, \varepsilon) = s_r$, hence $\pi(d_q^2, x) = s_r^1$ since $s_r \xrightarrow{x} s_r^1$.

It remains to check the controllability condition for $\mathcal{D}$. For components other than q and r this is obvious. We have four cases to examine.

First, let us take a state (d_q, x) of $\mathcal{D}_r$. Suppose that $\pi(d_q, x) \xrightarrow{b} s_r'$ is a local, uncontrollable transition in $\mathcal{A}_r$. We need to show that $(d_q, x) \xrightarrow{b} (d_q, xb)$ is possible in $\mathcal{D}_r$. Since (d_q, x) is a state of $\mathcal{D}_r$ we have $d_q \xrightarrow{x} d_q'$ in $\mathcal{D}_q^\triangledown$. Moreover, $\pi^\triangledown(d_q')$ is of the form (s_q, a, s_r, f) and $\pi(d_q, x) = s_r$. We get that $(s_q, a, s_r, f) \xrightarrow{b} (s_q, s_r', f)$ exists in $\mathcal{A}_q^\triangledown$. Since $\mathcal{D}^\triangledown$ satisfies the controllability condition, in $\mathcal{D}_q^\triangledown$ there must be a transition $d_q' \xrightarrow{b} d_q''$ for some d_q''. Hence, by definition, $(d_q, x) \xrightarrow{b} (d_q, xb)$ exists in $\mathcal{D}_r$.

For the next case we take a state d_q of $\mathcal{D}_q$ and suppose that $\pi(d_q) \xrightarrow{b} s_q'$ is a local, uncontrollable transition in $\mathcal{A}_q$. We need to show that a b-transition is possible from d_q in $\mathcal{D}_q$. We get $\pi^\triangledown(d_q)$ is of the form (s_q, a, s_r, f), and $\pi(d_q) = s_q$. This means that the transition $(s_q, a, s_r, f) \xrightarrow{b} (s_q', s_r, f)$

is in $\mathcal{A}_q^\triangledown$. Since $\mathcal{D}^\triangledown$ is covering, we get $d_q \xrightarrow{b} d'_q$ for some d'_q in $\mathcal{D}_q^\triangledown$. But then $d_q \xrightarrow{b} ts(d'_q)$ in $\mathcal{D}_q$ by definition.

The case of communication of q with $p \neq r$ is similar to the above.

The last case is a communication between q and r. So take $(d_q^1, (d_q^2, x))$ and suppose $(\pi(d_q^1), \pi(d_q^2, x)) \xrightarrow{b} (s'_q, s'_r)$ in $\mathcal{A}$. We have that $\pi^\triangledown(d_q^1)$ is of the form (s_q^1, a_1, s_r, f) and $\pi^\triangledown(d_q^2)$ is of the form (s_q^2, a_2, s_r, f); the s_r- and f-components are the same by Property (*). Moreover, by definition $\pi(d_q^1) = s_q^1$ holds. Let $s_r^1 = \pi(d_q^2, x)$, thus $s_r \xrightarrow{x} s_r^1$. These observations allow us to obtain the following sequence of transitions in $\mathcal{A}^\triangledown$:

$$(s_q^1, a_1, s_r, f) \xrightarrow{x} (s_q^1, a_1, s_r^1, f) \xrightarrow{b} (s'_q, s'_r)$$

Since $\mathcal{D}^\triangledown$ satisfies the controllability condition we must have transitions $d_q^1 \xrightarrow{x} d'_q \xrightarrow{b} d''_q$ in $\mathcal{D}^\triangledown$, with $\pi^\triangledown(d''_q) = (s'_q, s'_r)$. This means $(d_q^1, (d_q^2, x)) \xrightarrow{b} (ts(d''_q), (ts(d''_q), \varepsilon))$ in $\mathcal{D}$ and $\pi(ts(d''_q)) = s'_q$, $\pi(ts(d''_q), \varepsilon) = s'_r$. $\square$

As in the previous subsection we introduce the notation for runs and states.

Definition 3.16 *For $w \in \Sigma^\infty$ let $run(w)$ be the sequence of transitions of $\mathcal{D}$ when reading w. Observe that $run(w)$ may not be always defined. Let $state(w)$ be the last state in $run(w)$. By $run_p(w)$ we denote the projection of $run(w)$ on transitions of process p. Similarly for $w \in (\Sigma^\triangledown)^\infty$ we define $run^\triangledown(w)$, $state^\triangledown(w)$ and $run_p^\triangledown(w)$.*

For the correctness lemma we will need one more definition, that will allow to relate runs of $\mathcal{D}$ to runs of $\mathcal{D}^\triangledown$.

Definition 3.17 (*slow*) *We define $slow_r(\mathcal{D})$ as the set of all sequences labelling runs of $\mathcal{D}$ of the form*

$$y_0 x_0 a_1 \cdots a_k y_k x_k a_{k+1} \ldots \quad or \quad y_0 x_0 a_1 \cdots y_{k-1} x_{k-1} a_k x_k y_\omega\,,$$

where $a_i \in \Sigma_q \cap \Sigma_r$, $x_i \in (\Sigma_r^{loc})^*$, $y_i \in (\Sigma \setminus \Sigma_r)^*$, and $y_\omega \in (\Sigma \setminus \Sigma_r)^\omega$

Lemma 3.18 *A covering controller $\mathcal{D}$ is correct for $\mathcal{A}$ iff for all $w \in slow_r(\mathcal{D})$, $run(w)$ satisfies the correctness condition inherited from $\mathcal{A}$.*

Proof. Observe first $\mathcal{D}$ is r-short, since $\mathcal{A}$ is r-short and $\mathcal{D}$ is covering. Thus every sequence labelling some run of $\mathcal{D}$ either has finitely many r-actions or infinitely many communications of r with q.

Secondly, note that every sequence w labelling some run of $\mathcal{D}$ can be rewritten into a sequence w' from $slow_r(\mathcal{D})$ by repeatedly replacing factors ab by ba, if $dom(a) \cap dom(b) = \emptyset$. We have that $run(w')$ is also defined and

$run_p(w) = run_p(w')$ for every process p. Therefore for correctness it will be enough to reason on sequences from $slow_r(\mathcal{D})$. $\square$

For every sequence $w \in slow_r(\mathcal{D})$ as in Definition 3.17 we define the sequence $\chi(w) \in (\Sigma^\nabla)^\infty$ by induction on the length of w:

$\chi(\varepsilon) = ch(f_0)\, ch(a_0)$ $\qquad$ f_0 and a_0 determined by the initial q-state of $\mathcal{D}^\nabla$,

$$\chi(wb) = \begin{cases} \chi(w)b & \text{if } b \notin \Sigma_q \\ \chi(w)b\, ch(a) & \text{if } b \in \Sigma_q \setminus \Sigma_r \\ \chi(w)b\, ch(f)\, ch(a) & \text{if } b \in \Sigma_q \cap \Sigma_r. \end{cases}$$

where a and f are determined by $state_q^\nabla(\chi(w)b)$. The next lemma says, among other, that this definition makes sense, that is, $state_q^\nabla(\chi(w)b)$ is defined when needed.

Lemma 3.19 *For every sequence $w \in slow_r(\mathcal{D})$ we have that $run^\nabla(\chi(w))$ is defined. If w is finite then the states reached on w and $\chi(w)$ satisfy the following:*

1. *$state_p(w) = state_p^\nabla(\chi(w))$ for every $p \neq q, r$.*

2. *Let $w = y_0 x_0 a_1 \cdots a_k y_k x_k$, where $a_i \in \Sigma_q \cap \Sigma_r$, $x_i \in (\Sigma_r^{loc})^*$, and $y_i \in (\Sigma \setminus \Sigma_r)^*$. Then $state_r(w) = (d_q, x_k)$ and $state_q(w) = d'_q$, where $d_q = state_q^\nabla(\chi(y_0 x_0 a_1 \cdots a_k))$ and $d'_q = state_q^\nabla(\chi(y_0 x_0 a_1 \cdots a_k y_k))$.*

Proof. Induction on the length of $w = y_0 x_0 a_1 \cdots a_k y_k x_k$. If $w = \varepsilon$ then $state_q(\varepsilon) = d_q^1$ and $state_r(\varepsilon) = (d_q^1, \varepsilon)$ where $d_q^0 \xrightarrow{ch(f_0)\, ch(a)} d_q^1$ in $\mathcal{D}_q$, which shows the claim. Let $w = w'b$. If $b \notin (\Sigma_q \cup \Sigma_r)$, then $x_k = \varepsilon$, $y_k = y'b$, $\chi(w'b) = \chi(w')b$, $state_q(w'b) = state_q(w') \stackrel{ind.}{=} state_q^\nabla(\chi(y_0 x_0 a_1 \cdots a_k y')) = state_q^\nabla(\chi(y_0 x_0 a_1 \cdots a_k y')b)$. Moreover, $state_r(w'b) = state_r(w') \stackrel{ind.}{=} (d_q, \varepsilon)$, where $d_q = state_q^\nabla(\chi(y_0 x_0 a_1 \cdots a_k))$. Finally, assuming that $run^\nabla(\chi(w'))$ defined, observe that this run can be extended by a b-transition since it can be in w and the concerned states are the same.

We consider the remaining cases:

1. Let $b \in \Sigma_r^{loc}$, then $\chi(w'b) = \chi(w')b$ and $x_k = x'b$. We have $state_q(w'b) = state_q(w') \stackrel{ind.}{=} state_q^\nabla(\chi(y_0 x_0 a_1 \cdots y_k)) =: d'_q$. Moreover, $state_r(w') = (d_q, x')$, where $d_q = state_q^\nabla(\chi(y_0 x_0 a_1 \cdots a_k))$. In $\mathcal{D}_r$ there is a transition $(d_q, x') \xrightarrow{b} (d_q, x'b)$, which shows the claim about states. Finally we justify that the run on $\chi(w')$ in $\mathcal{D}^\nabla$ can be extended by a b. We know that $d_q \xrightarrow{x'b}$ and $d'_q \xrightarrow{x'}$ in $\mathcal{D}^\nabla$, and want to show that $d'_q \xrightarrow{x'b}$. This holds since $\mathcal{D}^\nabla$ is covering and since Property (*) guarantees that the s_r and f components of $\pi^\nabla(d_q)$ and $\pi^\nabla(d'_q)$ are the same.

2. Let $b \in \Sigma_q \setminus \Sigma_r$, so b is either local on q or a communication with $p \neq q, r$. We have $x_k = \varepsilon$ and $y_k = y'b$. Assume that b is local on q. We have $\chi(w) = \chi(w')b\,ch(a)$, where $a \in \Sigma_q^{loc}$ and d_q are such that $d_q = state_q^{\triangledown}(\chi(w'))$ and $d_q \xrightarrow{b} d_q^1 \xrightarrow{ch(a)} d_q^2$ in $\mathcal{D}^{\triangledown}$. By induction, $state_q(w') = state_q^{\triangledown}(\chi(y_0x_0a_1\cdots a_ky')) = d_q$, and by definition of $\mathcal{D}_q$, $d_q \xrightarrow{b} d_q^2 = ts(d_q^1)$. Thus $state_q(w) = d_q^2 = state_q^{\triangledown}(\chi(w))$ and the claim about states is shown. The run on $\chi(w)$ in $\mathcal{D}^{\triangledown}$ exists by the definition of $\chi(w)$ from $\chi(w')$.

   The case of a communication with $p \neq r$ is similar to the above.

3. Let $b \in \Sigma_q \cap \Sigma_r$ be a communication between q and r, thus $a_k = b$ and $x_k = y_k = \varepsilon$. We have $\chi(w) = \chi(w')b\,ch(f)\,ch(a)$, where a, f are such that $state_q^{\triangledown}(\chi(w')) = d_q \xrightarrow{b} d_q^1 \xrightarrow{ch(f)\,ch(a)} d_q^2$. Let $d_q' = state_q^{\triangledown}(\chi(y_0x_0a_1\cdots a_{k-1}))$ and $d_q'' = state_q^{\triangledown}(\chi(y_0x_0a_1\cdots a_{k-1}y_{k-1}))$. By induction, $state_q(w') = d_q''$ and $state_r(w') = (d_q', x_{k-1})$. In $\mathcal{D}$ we have a transition $(d_q'', (d_q', x_{k-1})) \xrightarrow{b} (d_q^2, (d_q^2, \varepsilon))$ since $d_q'' \xrightarrow{x_{k-1}} d_q \xrightarrow{b} d_q^1 \xrightarrow{ch(f)\,ch(a)} d_q^2$ in $\mathcal{D}^{\triangledown}$. Thus, $state_q(w) = d_q^2 = state_q^{\triangledown}(\chi(w))$ and $state_r(w) = (d_q^2, \varepsilon) = (state_q^{\triangledown}(\chi(w)), \varepsilon)$, which shows the claim about states. The run on $\chi(w)$ in $\mathcal{D}^{\triangledown}$ exists by the definition of $\chi(w)$ from $\chi(w')$.

□

Lemma 3.20 *If $w \in slow_r(\mathcal{D})$ and $run(w)$ is maximal in $\mathcal{D}$, then $run(\chi(w))$ is maximal in $\mathcal{D}^{\triangledown}$.*

Proof. Recall first that $run(\chi(w))$ is not maximal only if for some finite prefix x of $\chi(w)$, $run(x)$ can be extended by some action a (and the processes in $dom(a)$ do not appear anymore in the remaining suffix of $\chi(w)$). From the definition of $\chi(w)$ it follows that it suffices to consider prefixes of $\chi(w)$ of the form $\chi(u)$, where $w = uv$ with u finite. By Lemma 3.19 we note first that such an a cannot be on processes other than q or r, since $state_p(u) = state_p^{\triangledown}(\chi(u))$ for all $p \neq q, r$.

We consider the remaining cases, and assume $u = y_0x_0a_1\cdots a_ky_kx_k$:

1. Assume that $\chi(u)$ can be extended by some $b \in \Sigma_r^{loc}$ in $\mathcal{D}^{\triangledown}$, and let $d_q = state_q^{\triangledown}(\chi(y_0x_0a_1\cdots a_k))$, $d_q' = state_q^{\triangledown}(\chi(y_0x_0a_1\cdots a_ky_k))$, so $d_q' \xrightarrow{x_kb}$ in $\mathcal{D}_q^{\triangledown}$. By Lemma 3.19 we have $state_r(u) = (d_q, x_k)$ and by Property (*), the s_r- and f-components of $\pi^{\triangledown}(d_q)$ and $\pi^{\triangledown}(d_q')$ are the same. Since $\mathcal{D}^{\triangledown}$ is covering, this means that $d_q \xrightarrow{x_kb}$, hence there is a run on ubv in $\mathcal{D}$ so w was not maximal.

2. Assume that $\chi(u)$ can be extended by some $b \in \Sigma_q \setminus \Sigma_r$ and recall from Lemma 3.19 that $state_q(u) = d_q$, where $d_q = state_q^\nabla(\chi(y_0x_0a_1\cdots a_ky_k))$. Consider $u_1 = y_0x_0a_1\cdots a_ky_kb$ and assume that b is q-local (the case of a communication with $p \neq r$ is similar). We have $d_q \xrightarrow{x_k} d'_q \xrightarrow{b}$ in $\mathcal{D}^\nabla$ from some d'_q, and we want to show that $d_q \xrightarrow{b} d''_q$ for some d''_q. But this holds since $\mathcal{D}^\nabla$ is covering and the s_q components of $\pi^\nabla(d_q)$ and $\pi^\nabla(d'_q)$ are the same. So the run of w in $\mathcal{D}_q$ was not maximal, since there is a run on ubv in $\mathcal{D}$.

3. Assume that $\chi(u)$ can be extended by some $b \in \Sigma_q \cap \Sigma_r$. Recall from Lemma 3.19 that $state_q(u) = d'_q$ and $state_r(u) = (d_q, x_k)$, where $d_q = state_q^\nabla(\chi(y_0x_0a_1\cdots a_k))$ and $d'_q = state_q^\nabla(\chi(y_0x_0a_1\cdots a_ky_k))$. We have that $state_q^\nabla(\chi(u)) = d_q^1$ where $d'_q \xrightarrow{x_k} d_q^1$, and $d_q^1 \xrightarrow{b} d_q^2$. According to the definition of $\mathcal{D}$, there is a transition $(d'_q, (d_q, x_k)) \xrightarrow{b} (ts(d_q^2), (ts(d_q^2), \varepsilon))$ in $\mathcal{D}$, so that the run on w was not maximal.

Note that a run on $\chi(u)$ cannot be extended by actions of the form $ch(a)$ or $ch(f)$, since $\mathcal{D}^\nabla$ is covering. So the above four cases exhaust all the possibilities. □

Lemma 3.21 *If $\mathcal{D}^\nabla$ is a correct covering controller for $\mathcal{A}^\nabla$, then $\mathcal{D}$ is a correct covering controller for $\mathcal{A}$.*

Proof. By Lemma 3.18 it is enough to show that for all $w \in slow_r(\mathcal{D})$, $run(w)$ satisfies $Corr$. By Lemmas 3.19 and 3.20 the run on $\chi(w)$ exists and is maximal. Since $\mathcal{D}^\nabla$ is correct this run satisfies $Corr^\nabla$.

Consider a maximal run in $\mathcal{D}$, labelled by some $w \in slow_r(\mathcal{D})$. It is of one of the forms

$$y_0x_0a_1\cdots a_ky_kx_ka_{k+1}\ldots \quad \text{or} \quad y_0x_0a_1\cdots a_kx_ky_\omega$$

where $a_i \in \Sigma_q \cap \Sigma_r$, $x_i \in (\Sigma_r^{loc})^*$, $y_i \in (\Sigma \setminus \Sigma_r)^*$, and $y_\omega \in (\Sigma \setminus \Sigma_r)^\omega$

By Lemma 3.19 $run_p(w)$ and $run_p^\nabla(\chi(w))$ are the same for $p \neq q, r$. Since for such p also the correctness conditions of $\mathcal{A}$ and $\mathcal{A}^\nabla$ are the same, and since $run_p^\nabla(\chi(w))$ satisfies $Corr_p^\nabla$, so does $run_p(w)$.

Considering $run_q(w)$, Lemma 3.19 gives us $state_q(y_0x_0a_1\cdots a_ky_k) = state_q(y_0x_0a_1\cdots a_ky_kx_k) = state_q^\nabla(\chi(y_0x_0a_1\cdots a_ky_k))$ for every k. Moreover, the $\mathcal{A}_q$-component does not change when going from $\pi^\nabla(\chi(y_0x_0a_1\cdots a_ky_k))$ to $\pi^\nabla(\chi(y_0x_0a_1\cdots a_ky_kx_k))$. Thus, $\pi(run_q(w))$ is equal to the projection on $\mathcal{A}_q$ of $\pi^\nabla(run_q(\chi(w)))$, so $run_q(w)$ satisfies $Corr_q$.

It remains to consider $run_r(w)$. By Lemma 3.19 we have $state_r(y_0x_0a_1\cdots a_ky_kx_k) = (d_q, x_k)$ with $d_q = state_q^\nabla(\chi(y_0x_0a_1\cdots a_k))$, for every k. Recall that $\pi(d_q, x)$ was defined as the $\mathcal{A}_r$-component of $\pi^\nabla(d'_q)$, where $d_q \xrightarrow{x} d'_q$ in $\mathcal{D}^\nabla$. Assume first that w is of the form $y_0x_0a_1\cdots a_ky_kx_ka_{k+1}\ldots$. Observe that

$\pi(run_r(w))$ is equal to the projection on $\mathcal{A}_r$ of $\pi^\triangledown(run_q^\triangledown(\chi(w)))$, thus $run_r(w)$ satisfies $Corr_r$ because $run_r^\triangledown(\chi(w))$ satisfies $Corr_r$. Let now w be of the form $y_0x_0a_1\cdots a_kx_ky_\omega$. Since $run(w)$ is maximal we have that $state_r(w)\in T_r$, again because $run_r^\triangledown(\chi(w))$ satisfies $Corr_r$.

□

4 Short automata

In this section we justify our restriction to r-short automata. Recall that we have assumed that all controllable actions are local.We consider a tree architecture with a leaf process r and its parent q. In this section we want to simplify the r-component of $\mathcal{A}$. We will show that we can assume that the r-component is short (cf. Definition 3.2).

The first simplification step is to assume that the correctness condition on r is a parity condition. That is, it is given by a rank function $\Omega_r : S_r \to \mathbb{N}$ and the set of terminal states T_r. We can assume this since every regular language of infinite sequences can be recognized by a deterministic parity automaton. So if the correctness condition of $\mathcal{A}_r$ is not parity, it suffices to take the product of $\mathcal{A}_r$ and a deterministic parity automaton recognizing the correctness condition of $\mathcal{A}_r$.

The second simplification is to assume that the automaton $\mathcal{A}$ is *r-aware* with respect to the parity condition on r. This means that the state of r determines the biggest rank that has been seen since the last communication of r with q. It is easy to transform an automaton to an r-aware one.

Recall that an automaton $\mathcal{A}$ is *r-short* if there is a bound on the number of r-actions it can do without doing a communication with q. The goal of this section is essentially to show that we can reduce the control problem to the problem for r-short automata.

Recall that if $\mathcal{C}$ is a covering controller for $\mathcal{A}$ (cf. Definition 3.3)then there is a function $\pi : \{C_p\}_{p\in\mathbb{P}} \to \{S_p\}_{p\in\mathbb{P}}$, mapping each C_p to S_p and respecting the transition relation: if $c_{dom(b)} \xrightarrow{b} c'_{dom(b)}$ then $\pi(c_{dom(b)}) \xrightarrow{b} \pi(c'_{dom(b)})$.

Definition 4.1 (r-memoryless controller) *A covering controller $\mathcal{C}$ for $\mathcal{A}$ is* r-memoryless *if for every pair of states $c_r \neq c'_r$ of $\mathcal{C}_r$:*

> *if there is a path on local r-actions from c_r to c'_r then $\pi(c_r) \neq \pi(c'_r)$.*

Intuitively, a controller can be seen as a strategy, and r-memoryless means that it does not allow the controlled automaton to go twice through the same r-state between two consecutive communication actions of r and q.

Lemma 4.2 *Fix an r-aware automaton $\mathcal{A}$ with a parity correctness condition for process r. If there is a correct controller for $\mathcal{A}$ then there is also one that is covering and r-memoryless.*

Before proving the lemma we will state some preparatory definitions and facts.

By Lemma 3.5 we can assume that we have a covering controller for $\mathcal{A}$. We will see in the following how to convert this covering controller to an r-memoryless one. Let us fix an arbitrary linear order on the set C_r of states of the automaton $\mathcal{C}_r$. Let $\mathcal{C}_r^{\downarrow loc}$ denote the graph obtained from $\mathcal{C}_r$ by taking C_r as set of vertices and the transitions on local r-actions as edges. Since $\mathcal{C}$ is a covering controller, every sequence of actions in $\mathcal{C}$ can be performed in the controlled plant. Since $\mathcal{C}$ is correct for $\mathcal{A}$, every infinite sequence of local r-actions in the controlled plant satisfies the parity condition. We can lift this parity condition directly to $\mathcal{C}$ thanks to the fact that $\mathcal{C}$ is covering. We obtain that every infinite path in $\mathcal{C}_r^{\downarrow loc}$ satisfies the parity condition.

Before proceeding it will be convenient to recall some facts about parity games, in particular the notion of signature (or progress measure) [18]. We consider $\mathcal{C}_r^{\downarrow loc}$ as a parity game. Suppose that it uses priorities from $\{1, \ldots, d\}$. A signature is a d-tuple of natural numbers, that is, an element of $\mathbb{N}^d$. We will be interested in assignments of signatures to states of $\mathcal{C}_r^{\downarrow loc}$, that is in functions $sig : C_r \to \mathbb{N}^d$. Signatures are ordered lexicographically. We write $sig(c) \geq sig(c')$ if the signature assigned to c is lexicographically bigger or equal to that of c'. For $i \in \{1, \ldots, d\}$ we write $sig(c) \geq_i sig(c')$ if the signature of c truncated to the first i positions is lexicographically bigger or equal to the signature of c' truncated to the first i positions. For a fixed assignment of signatures sig and two states c, c' of $\mathcal{C}_r$ we write $c \rhd_{sig} c'$ if

$$sig(c) \geq_{\Omega(c)} sig(c') \quad \text{and the inequality is strict if } \Omega(c) \text{ is odd.}$$

We say that an assignment of signatures $sig : C_r \to \mathbb{N}^d$ is *consistent* if for every edge (c, c') of $\mathcal{C}_r^{\downarrow loc}$ we have $c \rhd_{sig} c'$. We now recall a fact that holds for every finite parity game, but we specialize them to $\mathcal{C}_r^{\downarrow loc}$.

Fact. Every path of $\mathcal{C}_r^{\downarrow loc}$ satisfies the parity condition iff there is a consistent assignment of signatures to states of $\mathcal{C}_r^{\downarrow loc}$.

After these preparations we can define for every state c_r of C_r its representative state in C_r, denoted $rep(c_r)$, as the unique state c'_r satisfying the following conditions:

1. $\pi(c_r) = \pi(c'_r)$ and c'_r is reachable from c_r;
2. for every c''_r with $\pi(c_r) = \pi(c''_r)$: if c''_r is reachable from c'_r then it belongs to the same SCC as c'_r;
3. among all states satisfying points (1) and (2) above we choose the smallest one in our fixed arbitrary ordering among those with smallest

possible signature. So we order first on the signature and then on the fixed arbitrary ordering.

Remark 4.3 *For every c'_r reachable in $\mathcal{C}_r^{\downarrow loc}$ from $rep(c_r)$: if $\pi(c'_r) = \pi(c_r)$ then $rep(c'_r) = rep(c_r)$. Indeed, by conditions (1) and (2) above $rep(c'_r)$ and $rep(c_r)$ must be in the same SCC. But then, the representative is uniquely determined by signature and ordering.*

We define now $\mathcal{C}_r^m$ from $\mathcal{C}_r$ by redirecting every transition on a local r-action to representatives: if the transition goes to a state c_r we make it go to $rep(c_r)$. Of course, $\mathcal{C}^m$ is still covering and Remark 4.3 implies that it is r-memoryless.

Remark 4.4 *If we have a transition $c_r \xrightarrow{b} c'_r$ in $\mathcal{C}_r^m$ then there is a sequence $z \in \Sigma_r^{loc}$ of local r-actions and some state c''_r such that $c_r \xrightarrow{b} c''_r \xrightarrow{z} c'_r$ in $\mathcal{C}_r$.*

Remark 4.4 allows to map paths in $\mathcal{C}_r^m$ into paths in $\mathcal{C}_r$. Consider a state c_1 of $\mathcal{C}_r^m$ and a finite sequence $x \in (\Sigma_r^{loc})^*$ such that $x = b_1 \cdots b_k$ labels some path from c_1 in $\mathcal{C}_r^m$, say $u = c_1 \xrightarrow{b_1} c_2 \xrightarrow{b_2} c_3 \cdots \xrightarrow{b_k} c_{k+1}$. Remark 4.4 gives us a sequence $rep^{-1}(c_1, x) = b_1 z_1 b_2 z_2 \dots b_k z_k$, and a corresponding path in $\mathcal{C}_r$: $c_1 \xrightarrow{b_1 z_1} c_2 \xrightarrow{b_2 z_2} c_3 \dots \xrightarrow{b_k z_k} c_{k+1}$, for some $z_i \in (\Sigma_r^{loc})^*$. In particular the two paths end in the same state. Of course $rep^{-1}(c_1, x)$ is defined similarly for infinite sequences x.

Proof of Lemma 4.2. We are ready to show that $\mathcal{C}^m$ obtained from $\mathcal{C}$ by replacing $\mathcal{C}_r$ with $\mathcal{C}_r^m$ satisfies the parity condition. For this take a maximal run and suppose towards a contradiction that it does not satisfy the parity condition.

If on this run there are infinitely many communications between q and r then there is an equivalent run whose labelling has the form:

$$u = y_0 x_0 a_1 y_1 x_1 a_2 \dots \tag{1}$$

where $a_i \in \Sigma_q \cap \Sigma_r$, $x_i \in (\Sigma_r^{loc})^*$, and $y_i \in (\Sigma \setminus \Sigma_r)^*$. Here two runs are equivalent means that the projections of the two runs on every process are identical. In particular, if two runs are equivalent and one of them satisfies the correctness condition then so does the other.

Let $c_r^i = state_r^{\mathcal{C}^m}(y_0 x_0 a_1 \cdots a_i)$ be the state of $\mathcal{C}_r^m$ reached on the prefix of u up to a_i. Let $x'_i = rep^{-1}(c_r^i, x_i)$. We get that the sequence

$$u' = y_0 x'_0 a_1 y_1 x'_1 a_2 \dots \tag{2}$$

is a labelling of a maximal run in $\mathcal{C}$. The projections on processes other than r are the same for u and u'. It remains to see if the parity condition on r is satisfied. We have $c_r^i \xrightarrow{x_i} c_r^{i+1}$ in $\mathcal{C}_r^m$ and $c_r^i \xrightarrow{x_i'} c_r^{i+1}$ in $\mathcal{C}_r$. Since we lifted priorities to $\mathcal{C}$ and $\mathcal{C}^m$ (being both covering), the r-awareness of $\mathcal{A}$ lifts to $\mathcal{C}$ and $\mathcal{C}^m$, so the same maximal rank is seen when reading x_i and x_i'. This shows that the parity condition on r is satisfied on the run of $\mathcal{C}^m$ on u, since it is satisfied by the run of $\mathcal{C}$ on u'.

Consider now a maximal run with finitely many communications between q and r. There is an equivalent one labeled by a sequence of the form:

$$u = y_0 x_0 a_1 y_1 x_1 a_2 \cdots a_k y_k x_k \tag{3}$$

where y_k and x_k are potentially infinite. Since we have only modified the r-component of the controller, it must be x_k that does not satisfy the parity condition on r.

Suppose first that $x_k = b_1 b_2 \cdots$ is infinite. Take the run $c_1 \xrightarrow{b_1} c_2 \xrightarrow{b_2} c_3 \ldots$ in $\mathcal{C}_r^m$, where $c_1 = \mathit{state}_r^{\mathcal{C}^m}(y_0 x_0 a_1 \cdots a_k)$. We have a run $c_1 \xrightarrow{b_1} c_2' \xrightarrow{x_2} c_2 \xrightarrow{b_2} c_3' \xrightarrow{x_3} c_3 \cdots$ in $\mathcal{C}_r$, where $\mathit{rep}(c_i') = c_i$ and $x_i \in (\Sigma_r^{loc})^*$ is the accessibility path, as given by Remark 4.4. We have $c_i \rhd_{\mathit{sig}} c_{i+1}'$ for all $i = 1, 2, \ldots$ because there is an edge from c_i to c_{i+1}' in $\mathcal{C}_r^{\downarrow loc}$. Recall that $c_i = \mathit{rep}(c_i')$. The definition of representatives implies that either $\mathit{sig}(c_i') \geq \mathit{sig}(c_i)$ or c_i is in a strictly lower SCC than c_i'. Since lowering a component can happen only finitely many times we have $\mathit{sig}(c_i') \geq \mathit{sig}(c_i)$ for all i bigger than some n. We get $c_i \rhd_{\mathit{sig}} c_{i+1}$ for $i > n$ which implies that x_k satisfies the parity condition. A contradiction.

If x_k is finite then we define the sequence $u' = y_0 x_0' a_1 y_1 x_1' a_2 \cdots a_k y_k x_k'$ in $L(\mathcal{C})$, as in the first case. Since u was maximal in $\mathcal{C}^m$, we have that u' is maximal in $\mathcal{C}$ (if r can do an action in u', the same can be done in u, since u and u' end in the same state). Thus the r-state reached in $\mathcal{A}$ by u belongs to T_r, since this holds already for u'. We get again a contradiction.

We will use Lemma 4.2 to reduce the control problem to that for r-short automata.

Given $\mathcal{A}$ we define a r-short automaton $\mathcal{A}^{Ⓢ}$. All its components will be the same but for the component r. The states $S_r^{Ⓢ}$ of r will be sequences $w \in S_r^+$ of states of $\mathcal{A}_r$ without repetitions, plus two new states $\top, \bot$. For a local transition $s_r' \xrightarrow{b} s_r''$ in $\mathcal{A}_r$ we have in $\mathcal{A}_r^{Ⓢ}$ transitions:

$$\begin{aligned} &ws_r' \xrightarrow{b} ws_r's_r'' && \text{if } ws_r's_r'' \text{ a sequence without repetitions}\\ &ws_r' \xrightarrow{b} \top && \text{if } s_r'' \text{ appears in } w \text{ and the resulting loop is even}\\ &ws_r' \xrightarrow{b} \bot && \text{if } s_r'' \text{ appears in } w \text{ and the resulting loop is odd}\end{aligned}$$

There are also communication transitions between q and r:

$$(s_q, ws_r') \xrightarrow{b} (s_q', s_r'') \qquad \text{if } (s_q, s_r') \xrightarrow{b} (s_q', s_r'') \text{ in } \mathcal{A}$$

Notice that w disappears in communication transitions. The parity condition for $\mathcal{A}^{Ⓢ}$ is also rather straightforward: it is the same for the components other than r, and for $\mathcal{A}_r$ it is

- $\Omega^{Ⓢ}(ws_r) = \Omega(s_r)$,
- $T_r^{Ⓢ} = \{\top\} \cup \{ws_r : s_r \in T_r\}$.

Lemma 4.5 *The length of every sequence of local actions of process r in $\mathcal{A}^{Ⓢ}$ is bounded by the number of states of $\mathcal{A}_r$.*

Theorem 4.6 *There is a correct covering controller for $\mathcal{A}$ iff there is one for $\mathcal{A}^{Ⓢ}$.*

Proof. Consider the implication from left to right. Let $\mathcal{C}$ be a correct covering controller for $\mathcal{A}$. By Lemma 4.2 we can assume that it is r-memoryless. We show that $\mathcal{C}$ is also a covering correct controller for $\mathcal{A}^{Ⓢ}$. We will concentrate on correctness, since the covering part follows by examination of the definitions.

Let us take some maximal run $run^{Ⓢ}(u)$ of $\mathcal{A}^{Ⓢ} \times \mathcal{C}$, and suppose by contradiction that it does not satisfy the parity condition of $\mathcal{A}^{Ⓢ}$. By definition $run(u)$ is a run of $\mathcal{A} \times \mathcal{C}$, but it may not be maximal. We have by construction of $\mathcal{A}^{Ⓢ}$ that $state_p^{Ⓢ}(u) = state_p(u)$ for $p \neq r$ and that $state_r(u)$ is the last element of $state_r^{Ⓢ}(u)$. (Recall that $state_p^{Ⓢ}(u), state_p(u)$ denote the state reached on u by $\mathcal{A}^{Ⓢ} \times \mathcal{C}$ and $\mathcal{A} \times \mathcal{C}$, resp.)

Suppose that $run(u)$ is not a maximal run of $\mathcal{A} \times \mathcal{C}$. We will extend it to a maximal run $run(\overline{u})$. If $run(u)$ ended in $\perp$ in the r-component of $\mathcal{A}^{Ⓢ}$ then we could extend $run(u)$ to a run of $\mathcal{A} \times \mathcal{C}$ not satisfying the parity condition (here we use that $\mathcal{C}$ is memoryless, so the odd loop in $\mathcal{A}$ exists also into one in $\mathcal{A} \times \mathcal{C}$). So the only other possibility is that $run(u)$ ends in $\top$. In this case it is possible to extend $run(u)$ to a complete run of $\mathcal{A} \times \mathcal{C}$ by adding the even loop in the r-component. This makes r satisfy the parity condition. Let $run(\overline{u})$ be the resulting run.

Now observe that if a parity condition for some process $p \neq r$ is violated on u then on $\overline{u}$ the same condition is violated. If it is violated on r then the only remaining possibility is that there are finitely many r-actions in u, and the state reached on u is ws_r with $s_r \notin T_r$. But then u is a maximal run of $\mathcal{A} \times \mathcal{C}$ and is not well terminated on r either, a contradiction.

For implication from right to left we take a covering controller $\mathcal{C}^{Ⓢ}$ for $\mathcal{A}^{Ⓢ}$ and construct a controller $\mathcal{C}$ for $\mathcal{A}$. The controller $\mathcal{C}$ will be obtained by modifying the r-component of $\mathcal{C}^{Ⓢ}$. The states of $\mathcal{C}_r$ will be sequences of states of $\mathcal{C}_r^{Ⓢ}$. They will be of bounded length. We will have that if $c_1^{Ⓢ} \cdots c_k^{Ⓢ}$ is a state of $\mathcal{C}_r$ then $\pi^{Ⓢ}(c_k^{Ⓢ})$ is the state $s_1 \cdots s_k$ of $\mathcal{A}_r^{Ⓢ}$, where $\pi^{Ⓢ}(c_j^{Ⓢ}) = s_1 \cdots s_j$, for $j = 1, \ldots, k$. Moreover, we define $\pi(c_1^{Ⓢ} \cdots c_k^{Ⓢ}) = s_k$. The transitions of $\mathcal{C}_r$ are

- $wc^{Ⓢ} \xrightarrow{b} wc^{Ⓢ}d^{Ⓢ}$ if $c^{Ⓢ} \xrightarrow{b} d^{Ⓢ}$ in $\mathcal{C}_r^{Ⓢ}$ and $\pi^{Ⓢ}(d^{Ⓢ}) \neq \top$.
- $c_1^{Ⓢ} \cdots c_k^{Ⓢ} \xrightarrow{b} c_1^{Ⓢ} \cdots c_j^{Ⓢ}$ if $c_k^{Ⓢ} \xrightarrow{b} c^{Ⓢ}$ in $\mathcal{C}_r^{Ⓢ}$, $\pi^{Ⓢ}(c^{Ⓢ}) = \top$ and j is such that $\pi^{Ⓢ}(c_k^{Ⓢ})$ is $s_1 \cdots s_k$ with $s_k \xrightarrow{b} s_j$ in $\mathcal{A}$.

Notice that since $\mathcal{C}^{Ⓢ}$ satisfies the parity condition $\perp$ cannot be reached.

Observation 1: The construction of $\mathcal{C}$ guarantees that every sequence of local r-actions x of $\mathcal{C}$ has a corresponding (possibly shorter) sequence x' of $\mathcal{C}^{Ⓢ}$. If the sequence in $\mathcal{C}^{Ⓢ}$ starts in s_1 and finishes in s_2 then the sequence in $\mathcal{C}$ starts also in s_1, but now considered as a sequence of length 1, and finishes in a sequence ending in s_2. Since $\mathcal{A}$ is r-aware and $\mathcal{C}$ and $\mathcal{C}^{Ⓢ}$ are both covering, this means that the maximal rank seen on both sequences is the same.

We need to show that all maximal runs of $\mathcal{A} \times \mathcal{C}$ satisfy the parity condition. For contradiction suppose that $run(u)$ does not.

If there are infinitely many communications between q and r on u then we write it as

$$u = y_0 x_0 a_1 y_1 x_1 a_2 \dots \tag{4}$$

where $a_i \in \Sigma_q \cap \Sigma_r$, $x_i \in (\Sigma_r^{loc})^*$, and $y_i \in (\Sigma \setminus \Sigma_r)^*$. Now for every x_i, Observation 1 gives x_i' so that the maximal ranks on x_i and x_i' are the same, so for

$$u' = y_0 x_0' a_1 y_1 x_1' a_2 \dots \tag{5}$$

$run(u')$ is a maximal run of $\mathcal{C}^{Ⓢ}$. This gives a run violating the parity condition of $\mathcal{C}^{Ⓢ}$.

If there are finitely many communications between q and r on u then we write it as

$$u = y_0 x_0 a_1 y_1 x_1 a_2 \dots a_k y_k x_k \tag{6}$$

where y_k and x_k are potentially infinite. The only complicated case is when x_k is infinite. We need to show that the run of $\mathcal{C}_r$ on x_k satisfies the parity condition. Recall that the states of $\mathcal{C}_r$ are sequences of states of $\mathcal{C}_r^{Ⓢ}$. Moreover the length of this sequences is bounded. Take the shortest sequence appearing infinitely often in x_k. The biggest rank seen between consecutive appearances of this sequence is even, since the path can be decomposed into (several) even loops of $\mathcal{A}$. $\square$

The following corollary may be interesting for the analysis of the complexity.

Corollary 4.7 *In the r-short plant $\mathcal{A}^{Ⓢ}$ the r-controller may be chosen memoryless since there are no infinite local r-plays.*

Remark 4.8 *We claim that the complexity of the reduction from $\mathcal{A}$ to $\mathcal{A}^{\triangledown}$ is polynomial in the size of $\mathcal{A}_q$ and simply exponential in the size of $\mathcal{A}_q$. The reason is as follows. States of $\mathcal{A}_r^{Ⓢ}$ are simple paths (i.e., without repetition* *of states) of $\mathcal{A}_r$. When going from $\mathcal{A}$ to $\mathcal{A}^{\triangledown}$, the states of $\mathcal{A}^{\triangledown}_q$ contain r-local strategies $f : (S_r)^* \to \Sigma^{sys}_r$. Putting things together, in f we deal with paths of $\mathcal{A}^{\circledS}_r$ (mapping them to Σ^{sys}_r). But the latter can be written more succinctly as paths of $\mathcal{A}_r$.*

5 Controlling distributed programs

We sketch in this section an application of our main result to the control of multi-threaded Boolean programs with atomic read-write-modify instructions. The schema is the following. We introduce a process for every variable. An access to a shared variable can be modelled by a synchronisation. This way from a program we obtain a Zielonka automaton. We then find a controller for the automaton, and implement rendez-vous from the controller using the read-write-modify instructions we are given.

We consider as an example multi-threaded programs using the *compare-and-swap* (CAS) instruction. This instruction has 3 parameters: $\mathsf{CAS}(x$: variable; *old*, *new*: int). Its effect is to return the value of x and at the same time set the value of x to *new* but only if the value of x was *old*. This operation is more general than the *compare-and-set* instruction, that returns a single bit telling whether the value has changed. It is easier to deal with the more powerful instruction since usually there is no difficulty to encode the semantics of an instruction as a rendez-vous of a Zielonka automaton, but it is more challenging to encode a rendez-vous by read-write instructions.

Let us see how to model a multi-threaded boolean program with CAS instructions by a Zielonka automaton. Each thread t is represented by a process, say P_t. Each shared variable x is modelled by a process P_x, the state of which represents the value of x. The access of thread t to variable x via a CAS operation corresponds to a shared transition of P_t and P_x. Shared transitions are labelled $\mathrm{CAS}_{t,x}(i,k)$, taking as parameters the thread name t, the variable name x and two values i, k:

$$\langle i, s\rangle \xrightarrow{\mathsf{CAS}_{t,x}(i,k)} \langle k, s'\rangle \quad \text{and} \quad \langle i, s\rangle \xrightarrow{\mathsf{CAS}_{t,x}(j,k)} \langle i, s'\rangle\,, \ \text{ if } j \neq i\,.$$

where we use the notation $\langle i, s\rangle$ for the pair of states of P_x and P_t.

Let $\mathcal{A}$ be the Zielonka automaton corresponding to a given program using CAS instructions. We can apply Theorem 3.1 to $\mathcal{A}$ if (i) every variable is either local or shared by two threads, and (ii) the communication architecture given by processes P_t, P_x, is acyclic. If $\mathcal{A}$ is controllable then Theorem 3.1 gives us a correct covering controller $\mathcal{C}$. It is a Zielonka automaton and we need to translate it back into a multi-threaded program. In the new program, each shared variable takes as values the states of the P_x-component of $\mathcal{C}$. There is also an additional local variable c_t for each thread t, containing the state of the P_t-component of $\mathcal{C}$. Observe also that in our setting, all actions involving processes associated with variables, are uncontrollable. Thus, the P_x-component of the controller $\mathcal{C}$ forbids no action.

The figure below shows how to simulate an action a shared between P_t and P_x, by a subprogram with CAS-instructions. Note first that given a state q of P_x, if there is some action a enabled in q that is shared by P_t and P_x, then all transitions enabled in q are a-transitions. This is due to the fact that $\mathcal{C}$ is covering and $\mathcal{A}$ is obtained from a deterministic program. Let $(v_1, q) \xrightarrow{a} (v'_1, q'_1), \ldots, (v_k, q) \xrightarrow{a} (v'_k, q'_k)$ be the enabled transitions in q. We use below value 0 as a value that does not occur in the range of x in $\mathcal{A}$.

if $(c_t = q)$ then {
(1) $v_{old} := \mathsf{CAS}_{t,x}(0, 0)$;
(2) while $(v_{old} \neq \mathsf{CAS}(v_1, v'_1))$ and $\cdots$ and $v_{old} \neq \mathsf{CAS}(v_k, v'_k))$ do
(3) $v_{old} := \mathsf{CAS}_{t,x}(0, 0)$
(4) end-while
(5) $c_t := q'_i$ where i such that $v_{old} = v_i$}.

Line (1) above reads the current value of x into v_{old}. Line (2) tries to set the value of x to a new value as required by the transition function on a. This can fail, since another process might have changed x in the meantime. If some test in line (2) succeeds, say for $v_{old} = v_i$ then line (5) updates variable c_t accordingly and the result is a simulation of a transition $(v_i, q) \xrightarrow{a} (v'_i, q'_i)$ of the Zielonka automaton.

Note that we need for the above simulation of $\mathcal{C}$ to be faithful some fairness condition preventing P_t to stay forever in the while-loop since another process overwrites infinitely often x. This is a common situation with shared variable programs.

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

[3] B. Finkbeiner and S. Schewe. Uniform distributed synthesis. In *Proceedings of LICS'05*, pages 321–330. IEEE, 2005.

[4] P. Gastin, B. Lerman, and M. Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In *FSTTCS*, volume 3328 of *LNCS*, pages 275–286. Springer, 2004.

[5] P. Gastin and N. Sznajder. Fair synthesis for asynchronous distributed systems. *ACM Transactions on Computational Logic*, 2013.

[6] B. Genest, H. Gimbert, A. Muscholl, and I. Walukiewicz. Asynchronous games over tree architectures. In *Proceedings of ICALP'13*, 2013.

[7] S. Graf, D. Peled, and S. Quinton. Achieving distributed control through model checking. *Formal Methods in System Design*, 40(2):263–281, 2012.

[8] J. Gutierrez and G. Winskel. Borel determinacy of concurrent games. In *Proceedings of CONCUR'13*, 2013.

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

[10] P. Madhusudan and P. Thiagarajan. Distributed control and synthesis for local specifications. In *ICALP'01*, volume 2076 of *LNCS*, pages 396–407. Springer, 2001.

[11] P. Madhusudan, P. S. Thiagarajan, and S. Yang. The MSO theory of connectedly communicating processes. In *Proceedings of FSTTCS'05*, volume 3821 of *LNCS*, pages 201–212. Springer, 2005.

[12] A. Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.

[13] P.-A. Melliès. Asynchronous games 2: The true concurrency of innocence. *TCS*, 358(2-3):200–228, 2006.

[14] M. Mukund and M. A. Sohoni. Keeping Track of the Latest Gossip in a Distributed System. *Distributed Computing*, 10(3):137–148, 1997.

[15] A. Muscholl and S. Schewe. Unlimited decidability of distributed synthesis with limited missing knowledge. In *Proceedings of MFCS'13*, 2013.

[16] 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.

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

[18] I. Walukiewicz. Pushdown processes: Games and model checking. *Inf. Comput.*, 164(2):234–263, 2001.

[19] W. Zielonka. Notes on finite asynchronous automata. *RAIRO–Theoretical Informatics and Applications*, 21:99–135, 1987.