Controlling Loosely Cooperating Processes

Anca Muscholl[1] and Sven Schewe[2]

[1] LaBRI, Université Bordeaux, Talence, France
[2] Department of Computer Science, University of Liverpool, Liverpool, United Kingdom

Abstract

In this article, we consider the problem of controlling loosely cooperating processes. We show that this distributed control problem is EXPTIME-complete if we restrict the number of processes to two, and undecidable for three or more processes.

Keywords: Control and games, Zielonka automata, Loosely cooperating processes

1. Introduction

Asynchronous processes that synchronise on shared actions [Maz77, Kel73, DR95] are a popular model for distributed systems. In this article, we discuss the control problem for such processes [RW89]. The control problem is to check whether or not each process has a local controller that restricts the set of runs by disabling some controllable actions in such a way that all joint behaviours satisfy a given local property on each process.

When studying the control problem, a major design decision is the knowledge the processes have about each other. We study *loosely cooperating* processes [Zie87]. Loosely cooperating processes, and hence their local controllers, obtain no additional knowledge by performing shared actions with other processes. All they know about other processes is what they can infer from their local history, in particular the history of their cooperation with other processes. Synthesis of loosely cooperating processes (or *synchronised products of transition systems*) from regular specifications has a simple solution for local acceptance conditions [Zie87, Muk02] and is an open problem if the acceptance is global (see [BBL05] for partial results).

We show that the control problem for loosely cooperating processes is undecidable even for local reachability[1] properties. The proof is related to the undecidability of distributed control in the Pnueli and Rosner framework [PR90] with local specifications [MT01] and the synthesis problem for asynchronous distributed systems [SF06].

While these undecidability proofs [PR90, MT01, SF06] require only two processes, our undecidability proof uses three processes. This raises the question whether the third process is necessary, and we show that it is: we establish that local control is decidable

[1] The only other natural weak class of properties is safety, but optimal control for safety specifications can be obtained by simply blocking all actions that can be blocked.

(and EXPTIME-complete) in the two process case. The third process is therefore not a particularity of our proof, but a prerequisite for undecidability.

2. Preliminaries

2.1. Zielonka automata

Zielonka automata are simple distributed devices. Such an automaton is a parallel composition of several finite automata, called *processes*, that synchronise on shared actions. There is no global clock, and two processes can therefore perform a different number of actions between two shared 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 synchronise in order to perform this action. Actions from $\Sigma_p = \{a \in \Sigma \mid p \in dom(a)\}$ are called *p-actions*, they involve process p. If $|dom(a)| = 1$ then a is a *local action*, otherwise it is a *synchronisation action*.

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

- for every process p a finite set S_p of (local) states,
- the initial state $s_{in} \in \prod_{p\in\mathbb{P}} S_p$, and
- for every action $a \in \Sigma$ a partial transition function $\delta_a : \prod_{p\in dom(a)} S_p \stackrel{\cdot}{\to} \prod_{p\in dom(a)} S_p$ 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 refer to S_p as the set of *p-states* and to $\prod_{p\in\mathbb{P}} S_p$ as *global states*.

A *loosely cooperating automaton* (abbreviated as LCA) is a Zielonka automaton where each transition function δ_a is a product of local transition functions $\delta_p : S_p \times \Sigma_p \stackrel{\cdot}{\to} S_p$. Formally, it is a tuple of finite-state automata $\mathcal{A} = (\mathcal{A}_p)_{p\in\mathbb{P}}$, $\mathcal{A}_p = \langle S_p, \Sigma_p, \delta_p, (s_{in})_p\rangle$. It corresponds to a Zielonka automaton by setting

$$\delta_a(s_{dom(a)}) = s'_{dom(a)} \quad \text{iff} \quad \delta_p(s_p, a) = s'_p\,, \text{ for every } p \in dom(a)\,.$$

Thus, the difference between LCA and usual Zielonka automata is that, in an LCA, processes executing a shared action do not obtain any information about each other, except for the execution itself.

A Zielonka automaton can be viewed as an ordinary finite-state automaton with states $S = \prod_{p\in\mathbb{P}} S_p$, initial state s_{in}, and transitions $\Delta = \{s_{\mathbb{P}} \xrightarrow{a} s'_{\mathbb{P}} \mid (s_{dom(a)}, s'_{dom(a)}) \in \delta_a$, and $s_{\mathbb{P}\setminus dom(a)} = s'_{\mathbb{P}\setminus dom(a)}\}$. A run is a sequence $s^{(1)}, a_1, s^{(2)}, a_2, \ldots, s^{(n)}, \ldots$, with $s^{(i)} \in S$ and $a_i \in \Sigma$, which satisfies $s^{(i)} \xrightarrow{a_i} s^{(i+1)}$ for all i. An action a is enabled in a state $s \in S$ if $\Delta(s, a)$ is defined.

The finitary language $L(\mathcal{A})$ of this sequential automaton consists of the labellings of runs that start in the initial state and end in a final state, i.e., a state from some designated set $F \subseteq S$.

The location mapping dom defines an independence relation I in a natural way. Two actions $a, b \in \Sigma$ are *independent*, denoted $(a, b) \in I$, if they involve different processes, that is, if $dom(a) \cap dom(b) = \emptyset$. Note that the order of execution of two independent actions $(a, b) \in I$ in a Zielonka automaton is irrelevant: they can be executed as a, b, or b, a – or even concurrently. More generally, we can consider the congruence $\sim_I$ on Σ^* generated by I, and observe that, whenever $u \sim_I v$, the same global state is reached from the initial state on u and v. Hence, $u \in L(\mathcal{A})$ if, and only if, $v \in L(\mathcal{A})$.

The idea to describe concurrency by an independence relation was introduced by Mazurkiewicz [Maz77] and Keller [Kel73] in the late seventies. (See also [DR95].) An equivalence class $[w]_I$ induced by $\sim_I$ is called a Mazurkiewicz *trace*. It can also be viewed as a labelled partially ordered multiset (pomset) of a special kind. As we have observed, $L(\mathcal{A})$ is a sum of such equivalence classes. In other words it is *trace-closed* (w.r.t. (Σ, dom)).

Zielonka's theorem below says that every finite-state automaton whose language is trace-closed, can be turned into an equivalent Zielonka automaton. Zielonka automata are therefore a suitable model for the simple view of concurrency captured by Mazurkiewicz traces.

Theorem 2.1. *[Zie87, GGMW10] Let $dom : \Sigma \to (2^{\mathcal{P}} \setminus \{\emptyset\})$ be a distribution of actions. If a language $L \subseteq \Sigma^*$ is regular and trace-closed w.r.t. (Σ, dom) then there is a deterministic Zielonka automaton accepting L of size exponential in the number of processes and polynomial in the size of the minimal automaton for L.*

Remark 2.2. *Given a finite-automaton $\mathcal{A}$ with $L(\mathcal{A})$ trace-closed, it is not always possible to construct a loosely cooperating equivalent automaton. If the acceptance is defined by global final states, the question whether such an equivalent automaton exists, is open (see [BBL05] for some partial results). But if the acceptance is given by sets $F_p \subseteq S_p$ of* local *final states (thus $F = \prod_{p \in \mathbb{P}} F_p$) then it suffices to test whether $\mathcal{A}$ is equivalent to the LCA $\mathcal{B} = (\mathcal{B}_p)_{p \in \mathbb{P}}$, where $\mathcal{B}_p$ accepts the projection of $L(\mathcal{A})$ on Σ_p [Zie87, Muk02].*

2.2. Controlling loosely cooperating processes

We formulate our control problem as in [MTY05], or equivalently, as a variant of the Ramadge and Wonham supervisory control formulation [RW89]. We will then provide an equivalent description of the problem in terms of games.

Recall that, in the Ramadge and Wonham's control problem we are given an alphabet Σ of actions partitioned into system and environment actions: $\Sigma^{sys} \dot\cup \Sigma^{env} = \Sigma$. Given a finite automaton P (a *plant*) one looks for another finite automaton C (a *controller*) such that the synchronous product $P \times C$ satisfies a given specification. Additionally, the controller is required not to block environment actions. This can, for example, be obtained by requiring that the controller has a transition on every action from Σ^{env} from every state. In our setting, both plant and controller will be LCA.

Let $\mathsf{act}_p = \{a \in \Sigma^{sys} \mid dom(a) = \{p\}\}$ denote the *local controllable* actions of process p, and let $\mathsf{sync}_p = \Sigma_p \setminus \{a \mid dom(a) = \{p\}\}$ denote the *synchronisation* (shared) actions of p. In this article, we impose the restriction that all uncontrollable

actions are local, i.e., each process has its own environment. With some minor modifications, both results of the article also hold without this assumption.

Recall that controllers are LCA. We can represent the controller of process p as a *control strategy* $\sigma_p : \Sigma_p^* \to 2^{\mathsf{act}_p \cup \mathsf{sync}_p}$. Thus, a control strategy for p maps *local* histories to sets of controllable actions of p (local or shared).

A control strategy restricts the set of possible runs. The *p-history* $h_p(\rho)$ of a run $\rho = s^{(1)}, a_1, s^{(2)}, a_2, \ldots$ is the subsequence $a_{i_1}, a_{i_2}, \ldots$, where $i_1 < i_2 < \ldots$ are exactly the indices, where the action a_{i_j} belongs to Σ_p.

The set $\mathcal{C}(\mathcal{A}, \sigma)$ of *controlled runs* for a set of local controllers $\sigma = (\sigma_p)_{p \in \mathbb{P}}$ is the smallest set of runs of $\mathcal{A}$ that start in the initial state s_{in}, and satisfy the following conditions:

- $\epsilon \in \mathcal{C}(\mathcal{A}, \sigma)$,
- if $\rho = s^{(1)}, a_1, s^{(2)}, a_2, \ldots, s^{(n)}$ is in $\mathcal{C}(\mathcal{A}, \sigma)$ and $a \in \Sigma^{env}$ is enabled in state $s^{(n)}$ then ρ, a, s is in $\mathcal{C}(\mathcal{A}, \sigma)$, where $s^{(n)} \xrightarrow{a} s$,
- if $\rho = s^{(1)}, a_1, s^{(2)}, a_2, \ldots, s^{(n)}$ is in $\mathcal{C}(\mathcal{A}, \sigma)$, $a \in \Sigma^{sys}$, and $a \in \sigma_p(h_p(\rho))$ for each $p \in dom(a)$, then ρ, a, s is in $\mathcal{C}(\mathcal{A}, \sigma)$, where $s^{(n)} \xrightarrow{a} s$.

A controlled run $s^{(1)}, a_1, s^{(2)}, a_2, \ldots$ is *maximal*, if it cannot be extended by any action (controllable or uncontrollable). This means that either the run $s^{(1)}, a_1, s^{(2)}, a_2, \ldots, s^{(n)}$ is finite and it cannot be extended according to the rules above; or it is infinite, and then it is required that there is no action a and index n such that a is enabled in $s^{(n)}$ and, for every $m \geq n$ and $p \in dom(a)$, $p \notin dom(a_m)$.

In this article we consider *local reachability (LRP)* winning conditions. For this, every process has a set of final (target) states $F_p \subseteq S_p$, and the control objective is that all processes are eventually in a final state. We assume w.l.o.g. that all successors of final states are final.

Definition 2.3 (Control Problem). *The* control problem *is the question whether or not a set of processes of an LCA can be controlled in such a way that all maximal controlled runs satisfy a given LRP.*

Observation 2.4. *If the processes in an LCA can be controlled, then they can be controlled by a strategy $\sigma = (\sigma_p)_{p \in \mathbb{P}}$ such that σ_p maps each p-history either to a single local action or to a set of synchronisation actions, $\sigma_p : \Sigma_p^* \to \mathsf{act}_p \cup 2^{\mathsf{sync}_p}$.*

The above fact holds because when we turn a controller into a more restrictive controller that disables actions (without blocking), then the maximal controlled runs are a subset of the original maximal controlled runs. If in a given state there is some uncontrollable (and therefore local) enabled p-action then the control strategy of p can safely offer an empty set of controllable actions; if $\sigma_p(x)$ contains some local (controllable) action a, then we can safely set $\sigma_p(x) = \{a\}$. In both cases the resulting control strategy satisfies the LRP iff the former control strategy did.

3. Undecidability of the Control Problem for Loosely Cooperating Processes

In this section, we show that the control problem for loosely cooperating processes with local reachability objectives is undecidable. For this we reduce solving a Post's Correspondence Problem (PCP) to the control problem. Alternatively, we could give a reduction from distributed games with local specifications [MT01] to the control problem. The direct reduction shown below, however, is much simpler.

Let $(u_i, v_i)_{1 \leq i \leq n}$ be some instance of PCP over some alphabet $A = \{a_1, \ldots, a_m\}$. Let $K = \{1, \ldots, n\}$. We denote below by $\overline{A}$ and $\overline{K}$ a copy of A and K, respectively. Figure 1 shows two processes, U and V, that are to emit a solution to PCP, and a process T, whose task is to check that they do not cheat. The only uncontrollable actions are the two initial actions of process T, namely cw (check words) and ci (check indices).

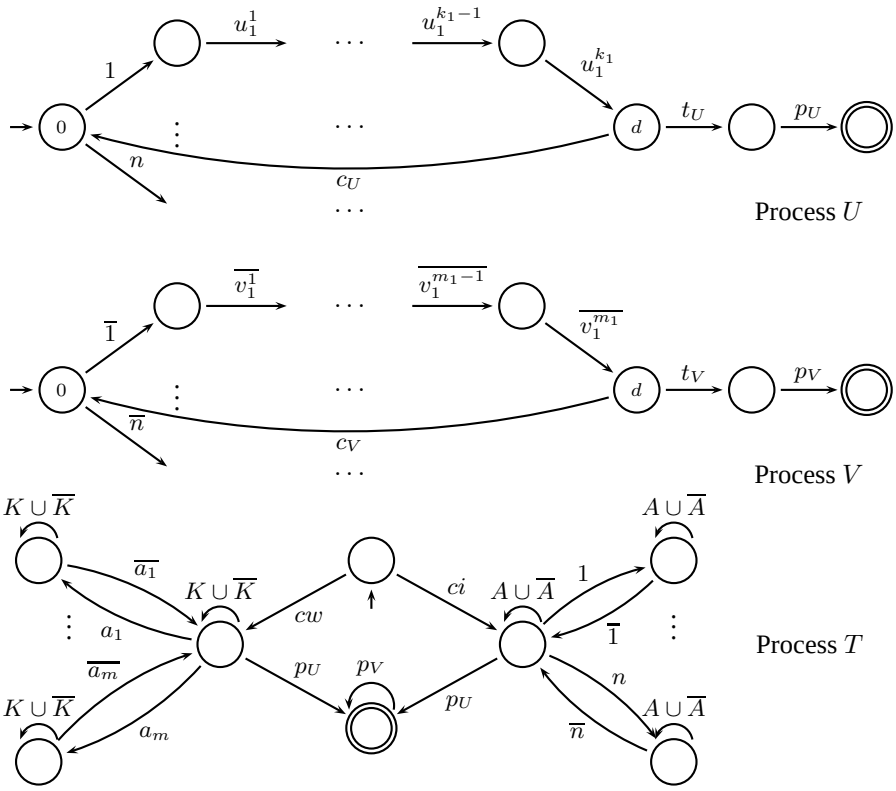

Figure 1: The processes U and V are structurally similar. They first emit a number, marking the index i, followed by the sequence $u_i^1 \cdots u_i^{k_i} = u_i$ and $v_i^1 \cdots v_i^{m_i} = v_i$ of actions from A and $\overline{A}$, respectively, defining u_i and v_i. Actions from $A \cup K$ are synchronisation actions between U and T, whereas actions from $\overline{A} \cup \overline{K}$ are synchronisation actions between V and T. At the end of this sequence, each process has in state d the choice between two local actions, c (continue), upon which it returns to its initial state, or t (test), upon which it continues to a test state. In the test state, it can only progress with p to its final state if the respective other process is in its test state, too. The test process T starts with an uncontrollable action, which can either be an ci for 'check indices', or a cw for 'check words'. In each case, T tests by synchronising alternatively with U and V, if the sequences emitted by the two processes are equivalent.

Lemma 3.1. *The three processes from Figure 1 can be controlled to satisfy the LRP if, and only if, the corresponding PCP has a solution.*

Proof. For the "if" direction, assume that there is a sequence $\alpha_1, \alpha_2, \ldots, \alpha_k$ of indices that induces a solution $u_{\alpha_1} u_{\alpha_2} \cdots u_{\alpha_k} = v_{\alpha_1} v_{\alpha_2} \cdots v_{\alpha_k}$ of PCP.

Then U uses the following control strategy (similarly for V):

- enable α_i (and only α_i) when being for the i^{th} time in state 0,
- enable the local action c_U (and only c_U) the first $k-1$ times it reaches state d,
- enable the local action t_U (and only t_U) the k^{th} time it reaches state d.

The third process, T, is constructed such that the initial uncontrollable choice decides if the equivalence of the sequence of indices or of the sequence of letters is tested (skipping over actions from the sequence that is not under test). In both cases, the test is passed, and the LRP satisfied.

Now assume that the processes can be controlled by some strategies σ_U and σ_V to satisfy the LRP. We can assume that σ_U and σ_V are as stated in Observation 2.4. Since only states 0 and d allow to choose (exactly one) controllable action, this means that each of the processes U and V has exactly one maximal run that complies with σ_U and σ_V, respectively. Since the LRP is satisfied, both these runs are finite. Let $\alpha \in K^*$ be the projection of U's run on K, and similarly $\beta \in K^*$ for V. Note that if $\alpha \neq \beta$, then T will eventually block when it starts by action ci. Therefore we have $\alpha = \beta$. For the same reasons, the projection of U's run on A is equal to the projection of V's run on A. Thus, the PCP has a solution. □

Corollary 3.2. *The control problem for LCA is undecidable for LRPs. This even holds if we restrict the number of processes to three, allow for only one process with a single state that is not fully controllable, and only allow communication between pairs of processes.*

4. Two Processes

In this section, we show that the control problem for LCA is decidable for two processes. In a nutshell, the argument for the two process case is that we can replace the local behaviour of each process by a *summary*. This summary abstracts the local strategy of a process between two synchronising actions. While such a summary will not encode the same actions we started with, it will preserve local reachability, provided the strategy of the other process remains unaltered.

Using summaries we can represent the strategies of both processes in a single tree, which branches by synchronising actions. This essentially reduces the problem to the classical case of single process synthesis.

Consider a strategy $\sigma_p : \Sigma_p^* \to \mathsf{act}_p \cup 2^{\mathsf{sync}_p}$ for process p (recall Observation 2.4). A *local run* for process p is one where every action is local on p. A local run that is consistent with σ_p (w.r.t some local history) is called *maximal* if it is either (i) infinite, or (ii) finite and there is neither an uncontrollable action nor a local controllable action allowed by σ_p in the last state of the run.

Definition 4.1 (Summary, consistent summary). *A* summary *for process p is a partial function $\varphi : S_p \dot{\to} 2^{(S_p \times \mathsf{sync}_p) \cup \{\bot\}}$ such that $\delta_p(t_p, a)$ is defined for every $(t_p, a) \in \varphi(s_p)$ and every s_p in the domain of φ.*

A summary φ is called consistent *if there exists a strategy $\sigma_p : \Sigma_p^* \to (\mathsf{act}_p \cup 2^{\mathsf{sync}_p})$ such that for some local history $h \in \Sigma_p^*$ both conditions below hold for every state s_p in the domain of φ and for every maximal local run ρ from s_p that is consistent with σ_p w.r.t h:*

- *if ρ is finite, $\rho = s_p, a_1, s_p^1, \ldots, a_n, s_p^n$ it holds that*
  - *if $\sigma_p(ha_1 \cdots a_n) \neq \emptyset$, then $(s_p^n, a) \in \varphi(s_p)$ for every $a \in \sigma_p(ha_1 \cdots a_n)$,*
  - *if $\sigma_p(ha_1 \cdots a_n) = \emptyset$, then $\bot \in \varphi(s_p)$ and $s_p^n \in F_p$,*
- *if ρ is infinite then it ultimately reaches F_p. Moreover, $\bot \in \varphi(s_p)$.*

The $\bot$ symbol in a summary indicates that process p may terminate its cooperation with other processes, either by continuing alone infinitely, or by reaching a state with no transition (neither controllable nor uncontrollable). Note that action a in the above definition is a synchronisation, since ρ is a maximal local run.

Observation 4.2. *Given a summary φ with domain S'_p for process p, we can determine whether it is consistent by solving 2-player reachability games on the the subgraph induced by the local transitions of p. For every state $s \in S'_p$ in the domain of φ, a game is played between a verifier, and a spoiler. The verifier selects the controllable local actions, and the spoiler selects the uncontrollable local actions to be executed. Verifier wins the game if she reaches*

- *a state s_p such that, for some action a, (s_p, a) is in the range of $\varphi(s)$, or*
- *in case that $\bot \in \varphi(s)$, a state s_p in the set F_p (recall that all successors of states in F_p are in F_p).*

A summary is consistent iff the verifier wins the respective game for all $s \in S'_p$ in the domain of φ.

Note that the verifier has a memoryless winning strategy in this game if she wins. We use an arbitrary (but fixed) memoryless winning strategy of a consistent summary to define a *witnessing strategy*.

In the remainder of this section, we fix two processes, P and Q, and let $\Gamma = \mathsf{sync}_P = \mathsf{sync}_Q$ be the set of synchronisation actions. A Γ-tree is a tree with edges labelled by Γ, such that each node has, for every $a \in \Gamma$, at most one a-child.

Definition 4.3 (Local summary tree). *A* local summary tree $\mathcal{T}_R$ *for a process R is a labelled Γ-tree, with each node labelled by some* consistent summary *for R. The root is labelled by a summary whose domain is singleton and contains the initial state. A node labelled by φ has an a-child with node label φ' iff a occurs in the range of φ. In this case, the domain of φ' is the set $\{\delta_R(t_R, a) \mid (t_R, a) \in \varphi(s_R)$ for some $s_R\}$.*

Definition 4.4 (Summary strategy tree). *A* summary strategy tree *for the processes P and Q is the product of two local summary trees $\mathcal{T}_P$ and $\mathcal{T}_Q$, for P and Q, respectively, synchronised on shared actions: a node with label (φ, φ') has an a-child with label (γ, γ') if γ is the a-child of φ in $\mathcal{T}_P$ and γ' is the a-child of φ' in $\mathcal{T}_Q$.*

A node in the summary strategy tree with label (φ, φ') is called *rejecting* if, for some states s_P, s_Q in the domain of φ and φ', respectively, one of the following holds:

- There is some $s'_P \notin F_P$ in the range of $\varphi(s_P)$ and either $\bot \in \varphi'(s_Q)$ or, for some s'_Q in the range of $\varphi'(s_Q)$: $\{a \mid (s'_P, a) \in \varphi(s_P) \text{ and } (s'_Q, a) \in \varphi'(s_Q)\} = \emptyset$.
- Symmetrically for Q.

Note that a rejecting node can never occur in a summary strategy tree associated with a control strategy satisfying the LRP: from such a node, one of the processes can be stuck (in non-final state) since the other process is either non-cooperating or not proposing a suitable shared action.

Definition 4.5. *A summary strategy tree is good, if it contains no rejecting node and every path eventually reaches a node with label (φ, φ') where*

- *every state in the range of φ is in F_P, and*
- *every state in the range of φ' is in F_Q.*

Proposition 4.6. *The processes P and Q can be controlled to satisfy the LRP iff there is some good summary strategy tree.*

Proof. For the "if" direction, it is sufficient to show that the strategies obtained by replacing each consistent summary by a witnessing strategy, are winning. Every play is of the form $\rho = x_0 y_0 a_0 x_1 y_1 a_1 \cdots x_k y_k a_k \cdots$ with x_i (resp. y_i) consisting of local actions of P (resp. Q), and a_i in Γ, for all i. Assume for contradiction that process P does not reach F_p in ρ. Recall that P cannot get stuck in a non-final state, since there is no rejecting node in the summary tree. If P is ultimately non-cooperating in ρ then, by definition of consistent summaries, it eventually reaches F_p. Otherwise, $a_0 a_1 \cdots$ is a path in the summary strategy tree that never reaches a node with label (φ, φ') such that every state in the range of φ is in in F_P, contradicting the fact that the summary strategy tree is good.

For the "only if" direction we define a summary strategy tree $\mathcal{T}$ from some fixed control strategies for P and Q that satisfy the LRP. As noted above, there is no rejecting node in the summary tree. Assume for contradiction the existence of some maximal path $a_0 a_1 \cdots$ in $\mathcal{T}$ witnessing that the summary tree is not good. If the path is finite and ends in a node labelled (φ, φ') then this node must be rejecting, a contradiction. If the path is infinite then one of the processes never reaches a final state, which is again a contradiction. □

Theorem 4.7. *The LRP control problem for two loosely cooperating processes is* EXPTIME*-complete.*

Proof. The upper bound follows from Proposition 4.6, by checking emptiness of an exponential-size tree automaton with reachability acceptance. To establish hardness, one can reduce the acceptance of an alternating Turing machine (ATM) with linear space bound to the control problem for a 2-process LCA. E.g., process P can emit sequences of configurations of this ATM, synchronising with process Q on each produced symbol/position pair. The update (transition) is chosen between the output of two configurations; it is chosen by the environment for universal and by the controller for existential configurations.

Process Q checks that each sequence of configurations is an accepting run of the ATM. This is done by letting the environment of Q test that two successive configurations are compatible with the transition between them, by choosing some position and recording the relevant symbols at that position for the two configurations. Note that P cannot cheat, since it is not informed about a possible test of the environment on Q.

If the respective test is passed, the second process goes into a final sink state, otherwise it goes into a rejecting sink. If no test is run, the second process will go into a final sink state when a halting configuration is output. □

Acknowledgment. We thank the reviewers for their careful reading and constructive comments.

References

[BBL05] Jean Berstel, Luc Boasson and Michel Latteux. Mixed languages. *Theor. Comput. Sci.*, 332(1-3):179-198, 2005.

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

[FS05] Bernd Finkbeiner and Sven Schewe. Uniform distributed synthesis. In *Proc. of LICS*, pages 321–330. IEEE Computer Society Press, 2005.

[GGMW10] Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. Optimal Zielonka-type construction of deterministic asynchronous automata. In *Proc. of ICALP*, LNCS 6199, 2010.

[Kel73] Robert M. Keller. Parallel program schemata and maximal parallelism I. Fundamental results. *Journal of the Association of Computing Machinery*, 20(3):514–537, 1973.

[Maz77] Antoni Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.

[MT01] P. Madhusudan and P. S. Thiagarajan. Distributed controller synthesis for local specifications. In *Proc. of ICALP*, LNCS 2076, pages 396–407, 2001.

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

[Muk02] Madhavan Mukund. From global specification to local implementations. *Synthesis and Control of Discrete Event Systems* (B. Caillaud et al., eds.), pp. 19-34, Kluwer 2002.

[PR90] Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *Proc. of FOCS*, pages 746–757. IEEE Computer Society Press, 1990.

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

[SF06] Sven Schewe and Bernd Finkbeiner. Synthesis of asynchronous systems. In *Proc. of LOPSTR*, LNCS 4407, pages 127–142, 2006.

[Zie87] Wieslaw Zielonka. Notes on finite asynchronous automata. *R.A.I.R.O. — Informatique Théorique et Applications*, 21:99–135, 1987.