Theoretical Computer Science 611 (2016) 136–141

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Note

Controlling loosely cooperating processes

Anca Muscholl [a], Sven Schewe [b,*]

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

ARTICLE INFO

Article history:
Received 23 May 2013
Received in revised form 12 February 2015
Accepted 25 July 2015
Available online 29 July 2015
Communicated by P.-A. Mellies

Keywords:
Control and games
Zielonka automata
Loosely cooperating processes

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.

1. Introduction

Asynchronous processes that synchronise on shared actions [6,5,2] are a popular model for distributed systems. In this article, we discuss the control problem for such processes [11]. 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 [13]. 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 [13,9] and is an open problem if the acceptance is global (see [1] 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 [10,3] with local specifications [7] and the synthesis problem for asynchronous distributed systems [12].

While these undecidability proofs [10,3,7,12] 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 (and EXPTIME-complete) in the two process case. The third process is therefore not a particularity of our proof, but a prerequisite for undecidability.

* Corresponding author.
E-mail address: sven.schewe@liverpool.ac.uk (S. Schewe).

[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.

http://dx.doi.org/10.1016/j.tcs.2015.07.044
0304-3975/

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 $\delta_a : \prod_{p\in dom(a)} S_p \to \prod_{p\in dom(a)} S_p$ 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 $\delta_p : S_p \times \Sigma_p \to S_p$. 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 $\Delta = \{s_{\mathbb{P}} \xrightarrow{a} s'_{\mathbb{P}} \mid (s_{dom(a)}, s'_{dom(a)}) \in \delta_a, \text{ and } s_{\mathbb{P}\setminus dom(a)} = s'_{\mathbb{P}\setminus dom(a)}\}$. 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 [6] and Keller [5] in the late seventies. (See also [2].) 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. *(See [13,4].) 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 [1] 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 [13,9].