Complexity Results for Checking Distributed Implementability

Keijo Heljanko
Laboratory for Theoretical Computer Science
Helsinki University of Technology
P.O. Box 5400, FI-02015 TKK, Finland
Keijo.Heljanko@tkk.fi

Alin Ştefănescu
Institute for Formal Methods
in Computer Science, University of Stuttgart
Universitätsstr. 38, Stuttgart, Germany
Alin.Stefanescu@fmi.uni-stuttgart.de

Abstract

We consider the distributed implementability problem: Given a labeled transition system TS together with a distribution Δ of its actions over a set of processes, does there exist a distributed system over Δ such that its global transition system is 'equivalent' to TS? We work with the distributed system models of synchronous products of transition systems [1] and asynchronous automata [18]. In this paper we provide complexity bounds for the above problem with three interpretations of 'equivalent': as transition system isomorphism, as language equivalence, and as bisimilarity. In particular, we solve problems left open in [4, 10].

1. Introduction

In this paper we study the computational complexity of the distributed synthesis problem. The problem has different versions, which share the following abstract formulation: Given a labeled transition system together with a *distribution* (as a relation telling which actions can be executed by which local agents), does there exist a *distributed system* over the given distribution that is behaviourally equivalent to the transition system?

The distributed synthesis problem has been studied for a number of abstract models of distributed systems (elementary net systems, place/transition Petri nets, synchronous products of transition systems [1], and Zielonka's asynchronous automata [18]) using various behavioural equivalences between the implementation and the specification (isomorphism, language equivalence, and bisimilarity). For nearly all these variants, axiomatic or language theoretic characterizations of the transition systems that can be distributed have been provided [5, 12, 9, 4, 17, 11]. Moreover, the computational complexity of the variants concerning elementary net systems and place/transition Petri nets is well understood [2, 3]. However, the complexity of many variants concerning synchronous products and asynchronous automata were still open. In this paper we fill many of these gaps, and in particular solve some problems left open in [4, 10].

Mukund [11] surveys (structural, behavioural) characterizations for synchronous products and asynchronous automata. In this paper we provide (the missing) lower and upper bounds for all the implementability tests presented in [11]. Tables 1,2 present a summary of the known and the new results. Note that, due to slightly different existing characterizations, the two models consider one, respectively multiple initial states. Also, we consider special cases in which the input transition system is assumed to be deterministic or acyclic (column 1).

In [9], Morin proved that the distributed implementability modulo isomorphism (column 2) can be solved in polynomial time when the input transition system is deterministic (the result holds for both synchronous products and asynchronous automata). In the nondeterministic case, results from [4, 10] show that the problem is in NP, but precise lower bounds were explicitly left open[1]. We show that the problem is NP-complete, even for acyclic specifications.

In [11], Mukund characterized the transition systems that can be implemented as a synchronous product modulo language equivalence. It is not difficult to see that this characterization leads to a PSPACE algorithm. We show that the problem is PSPACE-complete, even if the input transition system is deterministic, and coNP-complete if it is acyclic (Table 1, column 3). From the above we easily obtain the same results for the implementability problem modulo bisimulation *when the implementation is required to be deterministic*[2] (Table 1, column 4).

In [19], Zielonka characterized the transition systems that can be implemented as asynchronous automata modulo language equivalence. Combining this result with several others from the literature, we show that the implementabil-

[1] In [4, Sect. 5], the authors conjecture that "the synthesis problem for deterministic systems is much less expensive computationally than the synthesis problem in the general case".

[2] This is a natural constraint in many areas of hardware design.

Table 1. Implementability of synchronous products with one initial state

Specification (TS)	Isomorphism	Language Equivalence	Bisim. (determ. impl.)
Nondeterministic Deterministic	NP-complete P [9]	PSPACE-complete	PSPACE-complete
Acyclic & Nondet. Acyclic & Determ.	NP-complete P [9]	coNP-complete	coNP-complete

Table 2. Implementability of asynchronous automata with multiple initial states

Specification (TS)	Isomorphism	Language Equivalence	Bisim. (determ. impl.)
Nondeterministic Deterministic	NP-complete P [9]	PSPACE-complete P	P
Acyclic & Nondet. Acyclic & Determ.	NP-complete P [9]	coNP-complete P	P

ity problem has the same complexity as for synchronous products in the nondeterministic case, but can be solved in polynomial time in the deterministic case (Table 2, column 3). Maybe surprisingly, a simple trick allows us to extend this result to the implementability problem modulo bisimulation, again when the implementation is required to be deterministic (Table 2, column 4).

The paper is organized as follows. We start defining the distributed systems and their associated synthesis problem (Section 2). Sections 3,4,5 present the complexity bounds for the implementability problem, while last section is reserved for conclusions. Some of the easy proofs and technical details can be found in the full version of this paper [7] (available online).

2. The implementability problem for distributed systems

We begin with the general notion of a transition system. A *(labeled) transition system* is a tuple $TS = (Q, \Sigma, \rightarrow, I)$, where Q denotes the set of states, Σ the nonempty, finite alphabet of *actions*, $\rightarrow \subseteq Q \times \Sigma \times Q$ the transition relation, and $I \subseteq Q$ the nonempty set of initial states. We write $q \xrightarrow{a} q'$ to denote $(q, a, q') \in \rightarrow$. A transition system is called: *deterministic* if $|I| = 1$ and if $q \xrightarrow{a} q'$ and $q \xrightarrow{a} q''$ implies $q' = q''$; *reachable* if $\forall q \in Q\, \exists q^{in} \in I,\, w \in \Sigma^* : q^{in} \xrightarrow{w} q$; and *finite* if Q is finite. We assume that all the transition systems in this paper are finite and reachable.

To model synchronization, we need as ingredient the notion of *distributed alphabet* or (shorter) *distribution*: A *distribution* is a tuple $(\Sigma, Proc, \Delta)$, where Σ is a nonempty, finite set of *actions*, $Proc$ is a nonempty, finite set of *process labels*, and $\Delta \subseteq \Sigma \times Proc$ is a relation between actions and processes such that each action is in relation with at least one process and vice versa. Δ provides for each action the (nonempty) set of processes that are able to execute that action through the function $dom : \Sigma \rightarrow 2^{Proc}$ defined as $dom(a) := \{p \in Proc \mid (a, p) \in \Delta\}$. Dually, Δ provides for each process the (nonempty) set of actions that may be executed by that process through the function $\Sigma_{loc} : Proc \rightarrow 2^{\Sigma}$ defined as $\Sigma_{loc}(p) := \{a \in \Sigma \mid (a, p) \in \Delta\}$. We will sometimes use Δ to denote $(\Sigma, Proc, \Delta)$.

The two models of *distributed transition systems* considered in this paper are based on *synchronization on common actions* for a family of (local) transition systems: We study the (well known) *synchronous products of transition systems* [1] and a generalization of them, *asynchronous automata* [18].

Let $(\Sigma, Proc, \Delta)$ be a distribution. In the first synchronization model, we associate a local transition system with *each process* in $Proc$. A synchronization on a common action $a \in \Sigma$ occurs only when all the local states of the processes in $dom(a)$ enable a and execute it (i.e., update their local states). In the second synchronization model, we associate a transition relation with *each action* $a \in \Sigma$. A synchronization on a occurs only when the tuple of the local states of the processes in $dom(a)$ enable a in the 'hand-shake' relation associated with a (the local states are then updated according to this 'hand-shake'). In both cases, the execution of a only changes the local states of the processes in $dom(a)$.

Definition 2.1 A *synchronous product of transition systems* $\mathcal{SP}$ over a distribution $(\Sigma, Proc, \Delta)$ is a transition system $(Q, \Sigma, \rightarrow, I)$ for which there exist a family of local state sets $(Q_p)_{p \in Proc}$ and a family of local transition relations $(\rightarrow_p)_{p \in Proc}$ with $\rightarrow_p \subseteq Q_p \times \Sigma_{loc}(p) \times Q_p$ such that:

$Q \subseteq \prod_{p \in Proc} Q_p$ and Q consists of all the states reachable from I by

$$(q_p)_{p\in Proc} \xrightarrow{a} (q'_p)_{p\in Proc} \quad \Leftrightarrow$$
$$\begin{cases} q_p \xrightarrow{a}_p q'_p & \text{for all } p \in dom(a) \text{ and} \\ q_p = q'_p & \text{for all } p \notin dom(a). \end{cases}$$

Definition 2.2 An *asynchronous automaton* $\mathcal{AA}$ over a distribution $(\Sigma, Proc, \Delta)$ is a transition system $(Q, \Sigma, \rightarrow, I)$ for which there exist a family of local state sets $(Q_p)_{p\in Proc}$ and a transition relation $\rightarrow_a \subseteq \prod_{p\in dom(a)} Q_p \times \prod_{p\in dom(a)} Q_p$ for each $a \in \Sigma$, such that:

$Q \subseteq \prod_{p\in Proc} Q_p$ and Q consists of all the states reachable from I by

$$(q_p)_{p\in Proc} \xrightarrow{a} (q'_p)_{p\in Proc} \quad \Leftrightarrow$$
$$\begin{cases} (q_p)_{p\in dom(a)} \rightarrow_a (q'_p)_{p\in dom(a)} \text{ and} \\ q_p = q'_p \quad \text{for all } p \notin dom(a). \end{cases}$$

The asynchronous automata are more powerful than the synchronous products of transition systems: Any synchronous product can be described as an asynchronous automaton, but there exist asynchronous automata that cannot be simulated by synchronous products [18, 11]).

The problem whose computational complexity we will study in this paper is:

Problem 2.3 (Distributed Implementability)
Given a distribution $(\Sigma, Proc, \Delta)$ *and a finite transition system* TS*, does there exist a* distributed transition system *over* Δ equivalent *to* TS*?*

Mukund [11] surveyed solutions for the above problem where the 'distributed transition system' is one of {*synchronous product of transition systems*, *asynchronous automaton*} and 'equivalent' is one of {*isomorphic*, *language equivalent*, *bisimilar*[3]}. Mukund presents characterizations results without a computational complexity analysis viewpoint. Since we are interested to know which cases are tractable in practice, in this paper we study the complexity of the implementability problem (in many cases, solving this problem also provides an implementation). We follow the presentation of [11] and in addition we study the special cases when the input transition system is supposed to be *deterministic* and/or *acyclic*, for which the complexity results turn out to be usually more favorable. Also, we go a bit more general, allowing the (nondeterministic) distributed systems (Def. 2.1, 2.2) to have a set of initial states as opposed to only one initial state in [11].

3. Implementability modulo isomorphism

This section presents the complexity of checking whether an input transition system is isomorphic to the global state space of a distributed transition system.

[3]This case was only solved for *deterministic* distributed implementations [4].

We mention that, although in practice the initial specification is usually not isomorphic to a distributed transition system, the synthesis modulo isomorphism is still of relevance because it can be used to guide heuristics of constructing a distributed system exhibiting the same behaviour with the specification (see for instance the approach of [16] for the synthesis of asynchronous automata).

3.1. Synchronous products of transition systems

The *theory of regions* [5] proposed an approach of solving the synthesis modulo isomorphism for Petri nets. Along the same lines goes Theorem 3.1 below that characterizes the transition systems for which there exists an isomorphic synchronous product of transition systems. If such synchronous product exists, each of its local states Q_p (cf. Def. 2.1) is constructed as the quotient of the input state space under a local equivalence relation $\equiv_p$. These equivalences must be chosen such that the following hold: (SP1) an a-labeled transition does not affect the local states of the processes *not contained* in $dom(a)$; (SP2) the global state space is no more than the cartesian product of the Q_p's; and (SP3) for an action $a \in \Sigma$, if the local states of the processes in $dom(a)$ are able to perform an a-labeled transition, then a global synchronization must also be possible.

To simplify the notation, we use the following convention: For two given sets I and J such that $J \subseteq I$ and a given indexed family of binary relations $(\equiv_i)_{i\in I}$, the expression $(q_1 \equiv_J q_2)$ abbreviates $(\forall j \in J : q_1 \equiv_j q_2)$.

Theorem 3.1 *[4, 11] Let* $(\Sigma, Proc, \Delta)$ *be a distribution and* $TS = (Q, \Sigma, \rightarrow, I)$[4] *be a transition system. Then,* TS *is isomorphic to a synchronous product of transition systems over* Δ *if and only if for each* $p \in Proc$ *there exists an equivalence relation* $\equiv_p \subseteq Q \times Q$ *such that the following conditions hold:*

SP$_1$: *If* $q_1 \xrightarrow{a} q_2$*, then* $q_1 \equiv_{Proc\setminus dom(a)} q_2$.

SP$_2$: *If* $q_1 \equiv_{Proc} q_2$*, then* $q_1 = q_2$.

SP$_3$: *Let* $a \in \Sigma$ *and* $q \in Q$*. If for each* $p \in dom(a)$*, there exist* $q_p, q'_p \in Q$ *such that* $q_p \xrightarrow{a} q'_p$ *and* $q \equiv_p q_p$*, then for each choice of such* q_p*'s and* q'_p*'s, there exists* $q' \in Q$ *such that* $q \xrightarrow{a} q'$ *and* $q' \equiv_p q'_p$ *for each* $p \in dom(a)$.

Theorem 3.2 *The implementability problem for synchronous products modulo isomorphism is NP-complete, even for acyclic specifications.*

[4]In [4, 11], Th. 3.1 is restricted to the case when $|I| = 1$. By inspecting the proof of [4], it is easy to see that the theorem holds in fact for an arbitrary I.

Proof. First, it is easy to see that the problem is in NP: Given a distribution $(\Sigma, Proc, \Delta)$ and a transition system TS, a nondeterministic machine can 'guess' a family of equivalences $(\equiv_p)_{p \in Proc}$ and then verify in *polynomial* time (in the size of the distribution and of the transition system), whether the properties SP_1–SP_3 from Theorem 3.1 are satisfied or not.

For the NP-hardness part, we use a polynomial reduction from the classical SAT problem. Before going into details, we present an overview of the construction: Given a formula in conjunctive normal form, we associate to each variable and each clause, a group of three states and two transitions (as in Fig. 1). The nondeterminism is used to implement a choice gadget between the Boolean values *True* and *False* for each variable. We connect then the triples according to the occurrence of variables as literals in the clauses (these edges will be the wires that will transmit the information from variables to clauses). The distribution is chosen such that a clause will evaluate to *False* if and only if the condition SP_3 will be violated for the triple associated to the given clause. The application of Theorem 3.1 finishes the job.

Let ϕ be a formula in conjunctive normal form with variables $x_1, \ldots, x_n$ appearing in the clauses $c_1, \ldots, c_m$. For technical reasons and w.l.o.g., we assume that no clause contains some variable both as a positive and as a negative literal.

We will construct a distribution $(\Sigma_\phi, Proc_\phi, \Delta_\phi)$ and a (nondeterministic) transition system $TS_\phi = (Q_\phi, \Sigma_\phi, \rightarrow_\phi, I_\phi)$ such that: ϕ is satisfiable if and only if TS_ϕ is isomorphic to a synchronous product of transition systems over Δ_ϕ. To relieve a bit the notation, we will drop all ϕ indices.

First, the set of processes $Proc$ consists of two processes for each variable and one process for each clause:

$$Proc := \{p_{x_i}, p_{\overline{x_i}} \mid i \in [1..n]\} \cup \{p_{c_j} \mid j \in [1..m]\}.$$

Then, the set Σ of actions and their domains (which determine Δ) consist of:

- one action for each variable:
  $\{a_{x_i} \mid i \in [1..n]\}$ with $dom(a_{x_i}) := \{p_{x_i}, p_{\overline{x_i}}\}$.
- two actions for each positive literal from each clause:
  $\{a_{x_i c_j}, a'_{x_i c_j} \mid j \in [1..m], x_i \in c_j\}$ with $dom(a_{x_i c_j}) = dom(a'_{x_i c_j}) := Proc \setminus \{p_{x_i}\}$.
- two actions for each negative literal from each clause:
  $\{a_{\overline{x_i} c_j}, a'_{\overline{x_i} c_j} \mid j \in [1..m], \overline{x_i} \in c_j\}$ with $dom(a_{\overline{x_i} c_j}) = dom(a'_{\overline{x_i} c_j}) := Proc \setminus \{p_{\overline{x_i}}\}$.
- two actions for each clause:
  $\{a_{c_j}, a'_{c_j} \mid j \in [1..m]\}$ with $dom(a_{c_j}) := \{p_{c_j}\} \cup \{p_{x_i} \mid x_i \in c_j\} \cup \{p_{\overline{x_i}} \mid \overline{x_i} \in c_j\}$
  (the domain of a_{c_j} consists of the process associated to c_j and the processes associated to the literals of c_j) and $dom(a'_{c_j}) := Proc \setminus \{p_{c_j}\}$.

Last, we construct the transition system TS. The state space Q consists of:

- three states for each variable:
  $\{q^0_{x_i}, q_{x_i}, q'_{x_i} \mid i \in [1..n]\}$ and
- three states for each clause:
  $\{q_{c_j}, q'_{c_j}, q^0_{c_j} \mid j \in [1..m]\}$.

The transition relation $\rightarrow \subseteq Q \times \Sigma \times Q$ is defined as follows:

- for each $i \in [1..n]$: $q^0_{x_i} \xrightarrow{a_{x_i}} q_{x_i}$ and $q^0_{x_i} \xrightarrow{a_{x_i}} q'_{x_i}$ (nondeterminism is allowed).
- for each $j \in [1..m]$: $q_{c_j} \xrightarrow{a_{x_i c_j}} q_{x_i}$ for $x_i \in c_j$, $q_{c_j} \xrightarrow{a_{\overline{x_i} c_j}} q_{x_i}$ for $\overline{x_i} \in c_j$, and $q_{c_j} \xrightarrow{a_{c_j}} q^0_{c_j}$.
- for each $j \in [1..m]$: $q'_{c_j} \xrightarrow{a'_{x_i c_j}} q'_{x_i}$ for $x_i \in c_j$, $q'_{c_j} \xrightarrow{a'_{\overline{x_i} c_j}} q'_{x_i}$ for $\overline{x_i} \in c_j$, and $q'_{c_j} \xrightarrow{a'_{c_j}} q_{c_j}$.

The set of initial states I is chosen such that all states of Q are reachable from I. For instance, $I := \{q^0_{x_i} \mid i \in [1..n]\} \cup \{q'_{c_j} \mid j \in [1..m]\}$. [5]

An example is provided in Fig. 1 (the initial states are not marked).

The 'choice gadget' is provided by the three states for each variable x_i and their associated transitions. The Boolean 'value' of each choice (this will correspond to a local equivalence relation: either $q_{x_i} \equiv_{p_{x_i}} q'_{x_i}$ or $q_{x_i} \equiv_{p_{\overline{x_i}}} q'_{x_i}$) is then propagated further to the clauses using the transitions labeled $a_{x_i c_j}$ and $a_{\overline{x_i} c_j}$, respectively. More precisely, to each clause we forward only the information that a variable was set to *False* in such a way that the clause c_j is not satisfied iff q_{c_j} and q'_{c_j} are equivalent on all processes of the domain of a_{c_j}. This will imply that a clause c_j has all the literals evaluated to *False* if and only if the condition SP_3 is violated for $a := a_{c_j}$, $q := q'_{c_j}$, and $q_p := q_{c_j}$, $q'_p := q^0_{c_j}$.

The above construction is polynomial in the size of the initial formula ϕ and we claim that ϕ is satisfiable if and only if TS is isomorphic to a synchronous product of transition systems over Δ (given by dom).

First Implication. We prove the easier part: ϕ is not satisfiable implies TS is not isomorphic to a synchronous product of transition systems over Δ. If ϕ is not satisfiable, then for any assignment of the variables $x_1, \ldots, x_n$ there exists a

[5] It is easy to modify the construction such that there is only *one initial state* and the proof still works.

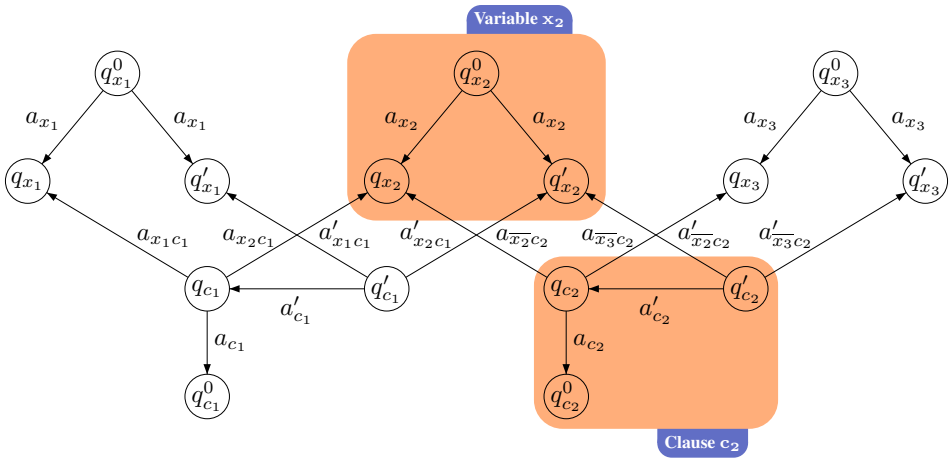

Figure 1. The transition system TS associated to $\phi = (x_1 \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3})$

clause that is evaluated to *False*. We must show that in this case, there are no $(\equiv_p)_{p \in Proc}$ satisfying all SP_1–SP_3.

By contradiction, assume that there exist $(\equiv_p)_{p \in Proc}$ satisfying all SP_1–SP_3. For each $i \in [1..n]$, we use first the condition SP_3 which we have assumed to hold. Let $a := a_{x_i}$ and $q := q^0_{x_i}$. We choose $q_p \xrightarrow{a} q'_p$ from SP_3 for each $p \in dom(a_{x_i}) = \{p_{x_i}, p_{\overline{x_i}}\}$ as follows: $q^0_{x_i} \xrightarrow{a_{x_i}} q_{x_i}$ for $p = p_{x_i}$ and $q^0_{x_i} \xrightarrow{a_{x_i}} q'_{x_i}$ for $p = p_{\overline{x_i}}$. Since $q \equiv_{p_{x_i}} q^0_{x_i}$ and $q \equiv_{p_{\overline{x_i}}} q^0_{x_i}$ (recall that $q = q^0_{x_i}$), the hypothesis of SP_3 is satisfied, so there must exist a state q' such that $q^0_{x_i} \xrightarrow{a_{x_i}} q'$, and also $q' \equiv_{p_{x_i}} q_{x_i}$ and $q' \equiv_{p_{\overline{x_i}}} q'_{x_i}$.

There are only two possible cases:

1. $q' = q_{x_i}$. In this case, we have $q_{x_i} \equiv_{p_{x_i}} q_{x_i}$ and $q_{x_i} \equiv_{p_{\overline{x_i}}} q'_{x_i}$.
2. $q' = q'_{x_i}$. In this case, we have $q'_{x_i} \equiv_{p_{x_i}} q_{x_i}$ and $q'_{x_i} \equiv_{p_{\overline{x_i}}} q'_{x_i}$.

So, we have that either $q_{x_i} \equiv_{p_{\overline{x_i}}} q'_{x_i}$ (case 1) or $q_{x_i} \equiv_{p_{x_i}} q'_{x_i}$ (case 2), but *not both at the same time* (otherwise, on one hand we have that $q_{x_i} \equiv_{dom(a_{x_i})} q'_{x_i}$ and on the other hand, by SP_1 applied to the transitions $q_{x_i} \xleftarrow{a_{x_i}} q^0_{x_i} \xrightarrow{a_{x_i}} q'_{x_i}$, we have $q_{x_i} \equiv_{Proc \setminus dom(a_{x_i})} q'_{x_i}$, so $q_{x_i} \equiv_{Proc} q'_{x_i}$ which contradicts SP_2).

Let us choose an assignment of the variables given by the above equivalences in the following way: For each $i \in [1..n]$,

x_i is evaluated to *False* if and only if $q_{x_i} \equiv_{p_{x_i}} q'_{x_i}$. (*)

Since ϕ is not satisfiable, there exists a clause, say c_k, that has all its literals evaluated to *False*. Let x_i be a positive literal in c_k (if any). Since the literal x_i is evaluated to *False*, we have that the variable x_i is *False*, so $q_{x_i} \equiv_{p_{x_i}} q'_{x_i}$. In addition, we have $q_{c_k} \xrightarrow{a_{x_i c_k}} q_{x_i}$ and $q'_{c_k} \xrightarrow{a'_{x_i c_k}} q'_{x_i}$ (see the construction of TS) and, using SP_1, we deduce that $q_{c_k} \equiv_{p_{x_i}} q_{x_i}$ and $q'_{c_k} \equiv_{p_{x_i}} q'_{x_i}$. By the transitivity of $\equiv_{p_{x_i}}$, we obtain that $q_{c_k} \equiv_{p_{x_i}} q'_{c_k}$. A similar argument for the negative literals $\overline{x_i}$ in c_k (if any) proves that $q_{c_k} \equiv_{p_{\overline{x_i}}} q'_{c_k}$ ($q_{x_i} \equiv_{p_{\overline{x_i}}} q'_{x_i}$ is used). Moreover, using SP_1 for $q'_{c_k} \xrightarrow{a'_{c_k}} q_{c_k}$, we have that $q_{c_k} \equiv_{p_{c_k}} q'_{c_k}$.

Summing up, we proved that $q'_{c_k} \equiv_p q_{c_k}$, for each $p \in dom(a_{c_k})$ (recall the definition of $dom(a_{c_k})$). But this contradicts SP_3, because $q_{c_k} \xrightarrow{a_{c_k}} q^0_{c_k}$ and there is no state q' such that $q'_{c_k} \xrightarrow{a_{c_k}} q'$.

Second Implication. The proof of this part is a bit technical and can be found in the full paper [7]. (Idea: For an assignment validating the satisfiable formula ϕ, we directly construct using (*) a set of equivalences $(\equiv_p)_{p \in Proc}$ that satisfy all the SP_1–SP_3 conditions.) □

Going into the proof details of Theorem 3.1 given in [4], we can show that if there exists a set of equivalences $(\equiv_p)_{p \in Proc}$ satisfying only conditions SP_1 and SP_3 (but not necessarily SP_2), then we can synthesize a synchronous product of transition systems accepting the same language as the initial transition system.[6] This trick widens the class of 'implementable' transition systems, while preserving the behavior. Yet, the problem is as hard as the implementability modulo isomorphism (from which we do the reduction – see the proof in [7]):

Corollary 3.3 *Let $(\Sigma, Proc, \Delta)$ be a distribution and TS a transition system. The problem of finding a set of equivalences $(\equiv_p)_{p \in Proc}$ satisfying only the conditions SP_1 and SP_3 of Theorem 3.1, is NP-complete.*

[6] In fact, the synthesized synchronous product is even bisimilar (in the sense of Milner) to the initial transition system.

Proposition 3.4 *[9] The implementability problem for synchronous products modulo isomorphism becomes decidable in* polynomial *time, if the input transition system is* deterministic.

3.2. Asynchronous automata

Complexity results similar to those for synchronous products of transition systems apply to asynchronous automata. The proofs have a similar structure and can be found in the full paper [7].

4. Implementability modulo language equivalence

This section presents the complexity of checking whether an input transition system admits the same sequences of actions as a distributed transition system.

A *run* of a transition system TS is a sequence of labels $a_1 \dots a_n \in \Sigma^*$ such that: $\exists q^{in} \in I, q \in Q : q^{in} \xrightarrow{a_1} \dots \xrightarrow{a_n} q$ (also written as $q^{in} \xrightarrow{a_1 \dots a_n} q$). The *language* of TS is the set of all its runs,

$$L(TS) := \{w \in \Sigma^* \mid \exists q^{in} \in I, q \in Q,\ q^{in} \xrightarrow{w} q\}.$$

Note that the language of a transition system is *prefix-closed*, i.e., $\forall u, w \in \Sigma^* : uw \in L \Rightarrow u \in L$. In fact:

Lemma 4.1 *A language $L \subseteq \Sigma^*$ is accepted by a finite transition system if and only if L is a prefix-closed regular language.*

The *language* of a distributed transition system is defined as the language of its underlying global transition system.

4.1. Synchronous products of transition systems

The synthesis modulo language equivalence for synchronous products is based on projections onto the local alphabets of the distribution. The solution provided in [11] works only for the class of synchronous products with just one initial state (i.e., $|I| = 1$ in Def. 2.1). In this section we discuss only the complexity of this problem and we will touch upon the general case at the end of the next section.

Problem 4.2 *Given a distribution $(\Sigma, Proc, \Delta)$ and a finite transition system TS, does there exist a synchronous product of transition systems over Δ with only* one initial state *that is language equivalent to TS?*

We present, following [11], the algorithm of deciding Problem 4.2: Let $(\Sigma, Proc, \Delta)$ be a distribution and TS a transition system.

1. W.l.o.g. we suppose that TS has only one initial state.
2. Let $TS = (Q, \Sigma, \to, \{q^{in}\})$. For each process $p \in Proc$, we construct the projection $TS_p := (Q, \Sigma_{loc}(p), \to_p, \{q^{in}\})$ obtained from a copy of TS in which the labels from $\Sigma \setminus \Sigma_{loc}(p)$ are replaced by ε and $\to_p$ is the ε-closure of $\to$ (a polynomial algorithm for ε-closure can be found in [8, Chap. 2.4]).
3. Problem 4.2 has a positive answer iff TS is language equivalent to the synchronous product over Δ of the transition systems $(TS_p)_{p \in Proc}$ with one global initial state $(q^{in}, \dots, q^{in})$.

We introduce now the reachability problem used in a subsequent reduction:

Problem 4.3 *(*Reachability in synchronous products*) Given $(\Sigma, Proc, \Delta)$ a distribution, a set of local transition systems $(TS_p)_{p \in Proc}$ with $TS_p = (Q_p, \Sigma_{loc}(p), \to_p, \{q_p^{in}\})$, and a global state $q \in \prod_{p \in Proc} Q_p$, is the state q reachable from the global initial state $(q_p^{in})_{p \in Proc}$ via the global synchronization of the $\to_p$'s as in Def. 2.1?*

Lemma 4.4 *The non-reachability problem (i.e., the complement of Problem 4.3) for synchronous products can be in polynomial time reduced to Problem 4.2.*

Proof. Given a distribution $(\Sigma, Proc, \Delta)$, we suppose $Proc := \{1, \dots, n\}$. Also, we are given a local transition system $TS_p = (Q_p, \Sigma_{loc}(p), \to_p, \{q_p^{in}\})$ for each $p \in [1..n]$ and a global state $q \in \prod_{p \in Proc} Q_p$.

We construct a distribution $(\Sigma', Proc', \Delta')$ and a transition system R such that: Problem 4.2 has a solution for Δ' and R if and only if the global state $q := (q_1, \dots, q_n)$ is not reachable from the global initial state $(q_1^{in}, \dots, q_n^{in})$.

The new distribution $(\Sigma', Proc', \Delta')$ is chosen as follows:

- $\Sigma' := \Sigma \cup \{a_p \mid p \in [1..n]\} \cup \{\surd\}$.
  (Note that $\Sigma = \bigcup_{p \in [1..n]} \Sigma_{loc}(p)$.)
- $Proc' := Proc \cup \{p_0\}$, and
- $\Delta' \subseteq \Sigma' \times Proc'$ is given by the local alphabets $\Sigma'_{loc}(p)$ as follows:
  - $\Sigma'_{loc}(p) := \Sigma_{loc}(p) \cup \{a_{p'} \mid p' \in [1..n] \wedge p' \neq p\} \cup \{\surd\}$ for every $p \in [1..n]$ and
  - $\Sigma'_{loc}(p_0) := \Sigma' \setminus \{\surd\}$.

  This gives the following domains dom' for the actions of Σ':
  - $dom'(a) = dom(a) \cup \{p_0\}$, for all $a \in \Sigma$ (where $dom(a)$ is given by Δ),
  - $dom'(a_p) = Proc' \setminus \{p\}$, for all $p \in [1..n]$, and
  - $dom'(\surd) = Proc' \setminus \{p_0\} = Proc$.

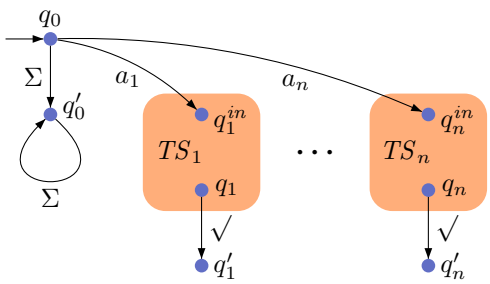

Figure 2. A schematic representation of the reduction in Lemma 4.4

The transition system $R := (Q', \Sigma', \rightarrow, \{q_0\})$ is sketched in Fig. 2 and defined as:

- $Q' := \{q_0, q_0'\} \cup \bigcup_{p\in[1..n]} Q_p \cup \{q_p' \mid p \in [1..n]\}$ (we assume $Q_p \cap Q_{p'} = \emptyset$ for all $p \neq p'$),
- $\rightarrow := \{q_0 \xrightarrow{a} q_0' \mid a \in \Sigma\} \cup \{q_0' \xrightarrow{a} q_0' \mid a \in \Sigma\} \cup \bigcup_{p\in[1..n]} \left(\{q_0 \xrightarrow{a_p} q_p^{in}\} \cup \rightarrow_p \cup \{q_p \xrightarrow{\surd} q_p'\} \right)$.

Remark 4.5 According to Step 2 in the decision algorithm for Problem 4.2, we construct the projections R_p of R onto the local alphabets $\Sigma'_{loc}(p)$ as follows: For $p \in [1..n]$, R_p is obtained from R replacing the labels from $\Sigma' \setminus \Sigma'_{loc}(p)$ by ε and applying an ε-closure. Since $a_p \notin \Sigma'_{loc}(p)$ and $\Sigma_{loc}(p) \cup \{\surd\} \subseteq \Sigma'_{loc}(p)$, we have that all the states reachable by a run v from q_p^{in} in TS_p can also be reached by *the same* run v from q_0 in R_p. For $p := p_0$, since $\Sigma'_{loc}(p_0) = \Sigma' \setminus \{\surd\}$, the projection R_{p_0} is just R without the $\surd$-labeled transitions.

First Implication. We assume that R is implementable over Δ' and we prove that the global state $q := (q_1, \ldots, q_n)$ is not reachable from $q^{in} := (q_1^{in}, \ldots, q_n^{in})$ in the synchronous product of the TS_p's over Δ.

By contradiction, suppose that there exists a run $w \in \Sigma^*$ such that q is reachable from q^{in} after executing the sequence w of actions. We show that R is not language equivalent to the synchronous product over Δ' of its projections R_p as described in the decision algorithm for Problem 4.2:

On one hand, the run $w\surd \notin L(R)$ because all the runs of R containing $\surd$ start with an a_p action and $a_p \notin \Sigma$, for any $p \in [1..n]$.

On the other hand, we can show that $w\surd$ is a run of the synchronization of the R_p's. Informally, we will simulate the synchronizations of TS_p's on $w \in \Sigma^*$ by synchronizations of R_p's and at the end we will also have a synchronization of the local transitions $q_p \xrightarrow{\surd} q_p'$:

In the synchronous product of the TS_p's, we can execute w from q^{in} and we reach q. According to Def. 2.1, the synchronization on each $a \in \Sigma$ involves only the processes of $dom(a)$. When synchronizing the projections R_p on $a \in \Sigma$, we must observe $dom'(a) = dom(a) \cup \{p_0\}$. For $p := p_0$, we can always execute $a \in \Sigma$ from $q_0 \xrightarrow{a} q_0' \xrightarrow{a} q_0'$ which is part of R_{p_0}. For $p \in dom(a)$, we can move in R_p (starting in q_0) similar to the synchronization of the TS_p (starting in q_0^{in}) according to Remark 4.5. In this way, we are able to execute w in the synchronous product of the R_p starting from the global state $(q_0, \ldots, q_0)$ and to reach the state q_p in R_p for each $p \in [1..n]$. Since $dom'(\surd) = Proc = [1..n]$ and we have $q_p \xrightarrow{\surd} q_p'$ in each $p \in [1..n]$, we can finally have a $\surd$-synchronization. Therefore, the run $w\surd$ belongs to the synchronization of the R_p's over Δ'.

Second Implication. We assume that q is not reachable from q^{in} in the synchronization of the TS_p's, and we prove that R is language equivalent to the synchronization of its projections over Δ'. Since it is easy to show that in general the language of a transition system is included in the language of the synchronization of its projections, we only have to prove the reverse inclusion.

Let $v \in \Sigma'$ be a run of the synchronization of the R_p's. We will show that $v \in L(R)$. It is easy to see that v can only have two forms:

$v \in (\Sigma' \setminus \{\surd\})^*$: From $\Sigma'_{loc}(p_0) = \Sigma' \setminus \{\surd\}$, we necessarily have that $v \in L(R_{p_0})$. Then, with the help of Remark 4.5, $L(R_{p_0}) \subseteq L(R)$, so $v \in L(R)$.

$v = w\surd$ with $w \in (\Sigma' \setminus \{\surd\})^*$: Again, from $\Sigma'_{loc}(p_0) = \Sigma' \setminus \{\surd\}$, we necessarily have that $w \in L(R_{p_0})$. Looking at R_{p_0}, w can only have two forms:

- $w \in \Sigma^*$: We show that this is not possible, given the fact that $w\surd$ is a run of the synchronization of the R_p's. The action $\surd$ will be executed only if all R_p's with $p \in dom'(\surd) = [1..n]$ will execute a $\surd$-labeled transition and this implies that no R_p with $p \in [1..n]$ will ever synchronize on a $q_0 \xrightarrow{a} q_0'$ transition for $a \in \Sigma$, because no run from q_0' can contain $\surd$. That means that the synchronization of the R_p's on $w \in \Sigma^*$ simulates a synchronization of the TS_p's on w. From the hypothesis, $q = (q_1, \ldots, q_n)$ is not reachable, so no $\surd$-synchronization will be possible.
- $w = a_i u$ with $i \in [1..n]$ and $u \in L(TS_i)$: Since the first action of w (and v) is a_i and $dom'(a_i) = Proc' \setminus \{i\}$, all R_p's *except* R_i must execute their local $q_0 \xrightarrow{a_i} q_p^{in}$ transition (this transition belongs to R_p for $p \neq i$, because in this case $a_i \in \Sigma'_{loc}(p)$). Then, the R_p's must synchronize on u such that at the end also a $\surd$-synchronization is possible. Since $u \in L(TS_i)$, we have that

$u \in (\Sigma'_{loc}(i))^*$ and also $u\surd \in (\Sigma'_{loc}(i))^*$. That means that R_i will take part in all synchronizations on $u\surd$ starting from q_0 and the only possibility for R_i to do this is by $q_0 \xrightarrow{u} q_i \xrightarrow{\surd} q'_i$. This necessarily implies a run $q_i^{in} \xrightarrow{u}_i q_i$ in TS_i (because $\Sigma_{loc}(i) \subseteq \Sigma'_{loc}(i)$), which further implies a run $q_0 \xrightarrow{a_i} q_i^{in} \xrightarrow{u} q_i \xrightarrow{\surd} q'_i$ in R, so $v \in L(R)$. □

Based on Lemma 4.4, the next results make also use of complexity results for checking non-reachability and language equivalence of synchronous products from [14]:

Theorem 4.6 *The implementability problem for synchronous products with $|I| = 1$ modulo language equivalence is PSPACE-complete.*

Proof. In Lemma 4.4 we have shown how the non-reachability problem for synchronous products can be in polynomial time reduced to the problem of deciding the implementability of a *single* transition system as a language equivalent synchronous product of transition systems. Thus by the PSPACE-hardness of the non-reachability problem for synchronous products proved in [14, Th. 3.10] we are able to deduce the PSPACE-hardness of our problem.

According to Step 3 of the decision algorithm of our problem, it is enough to check whether TS is language equivalent to the synchronization of its projections TS_p. But this test can be done in PSPACE as proved by [14, Th. 3.12], so our problem is in PSPACE. □

Proposition 4.7 *The implementability problem for synchronous products with $|I| = 1$ modulo language equivalence remains PSPACE-complete, when the input transition system TS is* deterministic. *For* acyclic *specifications the problem is coNP-complete, and it remains so even for deterministic ones.*

Proof. The PSPACE-hardness proof of [14, Th. 3.10] works in fact for *deterministic* TS_p's. The reduction of Lemma 4.4 constructs a deterministic input transition system R if the components TS_p's are all deterministic (see Fig. 2).

When TS is supposed to be acyclic, the coNP-hardness follows from the coNP-hardness of the non-reachability problem for synchronous products of *acyclic* and *deterministic* transition systems [14, Th. 3.16] and from Lemma 4.4 in which we modify the construction of R by replacing the loops $\{q'_0 \xrightarrow{a} q'_0 \mid a \in \Sigma\}$ by a set of new transitions $\bigcup_{j\in[0..M]}\{s_j \xrightarrow{a} s_{j+1} \mid a \in \Sigma\}$, where $s_0 = q'_0$ and $M = \max\{|w| \mid w \in L(TS_p),\, p \in [1..n]\}$ (this maximum exists if all the TS_p's are acyclic). In this way R is acyclic if all the TS_p's are acyclic and the reduction is still correct. The coNP-completeness follows from [14, Th. 3.17], which easily proves that checking language equivalence of synchronous products of acyclic transition systems is in coNP. □

4.2. Asynchronous automata

The 'engine' of the synthesis modulo language equivalence for asynchronous automata is a classical result by Zielonka [18] which constructs a (deterministic) asynchronous automaton accepting a regular *trace* language.

We go now a bit more into details. Each distribution $(\Sigma, Proc, \Delta)$ generates an *independence* relation between the actions of Σ: $a\|b$ iff $dom(a) \cap dom(b) = \emptyset$. Then, we say $T \subseteq \Sigma^*$ is a *trace language* if T is closed under the independence relation: $\forall w, w' \in \Sigma^*,\, a, b \in \Sigma : wabw' \in T \wedge a\|b \Rightarrow wbaw' \in T$. According to [18], for any regular trace language T there exists an asynchronous automaton equipped with a set of global accepting states recognizing T. Zielonka devoted a subsequent paper [19] to obtain the same result for the restricted class of *safe* asynchronous automata which have the property that any run from an initial state can be extended to an accepted run.[7] Global accepting states are not really suitable for a distributed setting and for this reason we defined in this paper the language of an asynchronous automaton as the set of *all* possible runs from an initial state. Using [19] we easily get the following characterization:

Proposition 4.8 *For a language $T \subseteq \Sigma^*$ and a distribution Δ, T is the language of an asynchronous automaton over Δ if and only if T is a prefix-closed regular trace language.*

For the results in this section we only give the proof idea and refer the curious reader to the full paper [7] for details.

Theorem 4.9 *The implementability problem for asynchronous automata modulo language equivalence is PSPACE-complete.*

Proof. Our implementability problem is in PSPACE, due to Proposition 4.8 and [13, Th. 8 with Cor. 10] which proved that checking whether the language of a finite automaton is a trace language can be decided in PSPACE. (Checking trace-closure is enough, for the input of the implementability problem is a transition system, whose language is always prefix-closed and regular.)

For the PSPACE-hardness part, we use a simple reduction from the totality problem '$= \Sigma^*$?' for nondeterministic finite automata, which is known to be PSPACE-hard [6]. For each nondeterministic finite automaton $\mathcal{A}$ over Σ, which we can suppose w.l.o.g. that has only one initial q^{in},

[7] However, we mention that for a regular trace language, there exists a *deterministic* asynchronous automaton accepting it [18], but not necessarily a *deterministic safe* one [19].

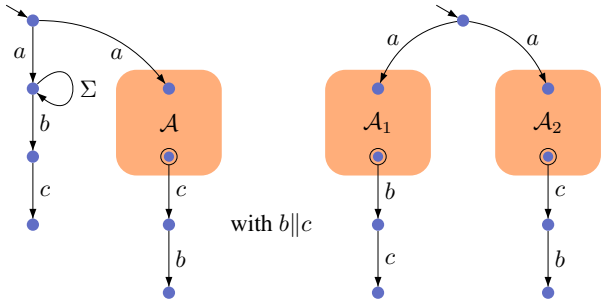

Figure 3. Schematic representations of the reductions in Th. 4.9 (left) and Prop. 4.10 (right)

respectively only one accepting state q^{acc}, we build a transition system TS over $\Sigma \cup \{a, b, c\}$ as in Fig. 3 (left) and a distribution such that $b \| c$ is the only independence. It is easy to prove that $L(\mathcal{A}, \{q^{acc}\}) = \Sigma^*$ iff $L(TS)$ is a trace language (this is enough according to Proposition 4.8). □

Proposition 4.10 *The implementability problem for asynchronous automata modulo language equivalence for* deterministic *specifications is decidable in polynomial time. For* nondeterministic acyclic *specifications, the problem is coNP-complete.*

Proof. The first part follows directly from Proposition 4.8 and [13, Th. 7] proving that it is decidable in polynomial time whether the language of a deterministic finite automaton is a trace language.

For the second part, we use a simple reduction sketched in Fig. 3 (right) from the NP-complete problem if two acyclic nondeterministic automata $\mathcal{A}_1$ and $\mathcal{A}_2$ accept different languages [6]. (The language of the transition system on the right of Fig. 3 is a trace language iff $L(\mathcal{A}_1) = L(\mathcal{A}_2)$.)□

Based on the observation that the language accepted by a synchronous product is necessarily a trace language, we can recycle the constructions in Fig. 3 to derive (lower) complexity bounds for synchronous products with *multiple initial states*:

Theorem 4.11 *The implementability problem for synchronous products of transition systems (with $|I| \geq 1$) modulo language equivalence is PSPACE-hard. For* nondeterministic acyclic *specifications, the problem is coNP-complete, whereas for the* deterministic acyclic *ones is in P.*

However, we suspect that the above general problem (i.e., the implementability problem for synchronous products with $|I| \geq 1$ modulo language equivalence) is much harder than PSPACE. Moreover, we do not know anything about its complexity when the specification is deterministic.

For the synthesis modulo language equivalence we can give the specification as a *regular expression*. It is easy to show that the results of Theorems 4.9 and 4.6 are preserved (we use the fact that checking the prefix-closure of the language of a regular expression E can be done in PSPACE[8]):

Proposition 4.12 *The implementability problem modulo language equivalence for asynchronous automata (respectively, for synchronous products with $|I| = 1$)* with regular expressions as specifications *is PSPACE-complete.*

4.2.1 Non-regular specifications

The following result suggests that there is no hope to test the implementability once we move higher in Chomsky's hierarchy:

Proposition 4.13 *Checking that a context-free language is a trace language is undecidable.*

The proof uses the fact that the set of invalid computations of a Turing machine is a context-free language [8, Lemma 8.7], together with the trick of making the first two letters of an accepting computation independent (see the proof technique of [13, Th. 11]).

5. Deterministic implementability modulo bisimulation

Based on the observation that bisimilarity and language equivalence coincide for deterministic transition systems, [11] provides characterizations for the synthesis modulo bisimulation with the restriction that *the distributed implementation is deterministic* (the specification may still be nondeterministic). More precisely, the deterministic implementability problem modulo bisimulation for a given input TS reduces to checking whether the quotient $TS/_{\sim_{TS}}$ (with $\sim_{TS}$ the largest bisimulation on TS) is deterministic and then checking deterministic implementability modulo language equivalence. As a consequence, we can infer basically the same complexity results as for implementability with deterministic specifications from Propositions 4.7 and 4.10. The full paper [7] gives more details.

However, the synthesis problem modulo bisimulation is still open in the case of *nondeterministic implementations* for both synchronous products and asynchronous automata.

6. Conclusions

We discover that the models of synchronous products of transition systems and asynchronous automata have similar

[8] Idea: For E, construct in polynomial time a nondeterm. finite automaton $\mathcal{A} = (Q, \Sigma, \rightarrow, I, F)$ s.t. $L(\mathcal{A}, F) = L(E)$. Then, $L(E)$ is prefix-closed iff $L(\mathcal{A}, F) = L(\mathcal{A}, F')$, where F' consists of all the states that are on a path from an initial state in I and an accepting one in F.

complexities for the implementability test. For both models, deciding the implementability modulo isomorphism provides a distributed implementation *for free*. For implementability modulo language equivalence, this bonus is still available for synchronous products (recall the algorithm of deciding Problem 4.2). This is not the case for asynchronous automata for which the computationally expensive construction of Zielonka is the best known approach to be used after we have decided that the specification is implementable. However, there is a balance. The complexity results suggest that starting with a deterministic specification is an advantage for asynchronous automata. Also, the asynchronous automata are strictly more expressive than the synchronous products (for the case studies in [16], solutions for the synthesis problem were obtained only for asynchronous automata, but not for synchronous products, due to the expressiveness of the former).

Our motivation for exploring the complexity issues surrounding synthesis is based on the need to select the most appropriate implementation methods for synthesis tools. When we know the exact complexity of the subproblems of synthesis at hand we can use general rules of thumb for selecting suitable implementation techniques. For example, in the full version of this paper [7], we map the synthesis problem modulo isomorphism for asynchronous automata (shown to be NP-complete) to the problem of finding a stable model of a logic program (another NP-complete problem) by using the SMODELS logic programming system [15]. This implementation [7] complements the prototype tool for synthesis of asynchronous automata from [16]. Furthermore, the PSPACE-hardness result of synthesis modulo language equivalence for synchronous products combined with the construction used for solving Problem 4.2 suggests that using model checking algorithms to solve Problem 4.2 can be fully appropriate. Work on this topic is left for further study.

Acknowledgments We are grateful to Javier Esparza and Petr Jančar for useful discussions. The financial support from Academy of Finland (Projects 53695, 211025, 213397, grant for research work abroad, research fellow post) is gratefully acknowledged. This work has also been financially supported by FET project ADVANCE contract No IST-1999-29082 and EPSRC grants 64322/01 (Automatic Synthesis of Distributed Systems) and 93346/01 (An Automata Theoretic Approach to Software Model Checking) while the first author was affiliated with University of Stuttgart, Institute for Formal Methods in Computer Science.

References

[1] A. Arnold. *Finite transition systems and semantics of communicating systems*. Prentice Hall, 1994.

[2] E. Badouel, L. Bernardinello, and P. Darondeau. Polynomial algorithms for the synthesis of bounded nets. In *TAPSOFT'95*, volume 915 of *LNCS*, pages 364–378. Springer, 1995.

[3] E. Badouel, L. Bernardinello, and P. Darondeau. The synthesis problem for elementary net systems is NP-complete. *TCS*, 186(1–2):107–134, 1997.

[4] I. Castellani, M. Mukund, and P. Thiagarajan. Synthesizing distributed transition systems from global specifications. In *FSTTCS19*, volume 1739 of *LNCS*, pages 219–231. Springer, 1999.

[5] A. Ehrenfeucht and G. Rozenberg. Partial (set) 2-structures I and II. *Acta Informatica*, 27(4):315–368, 1990.

[6] M. Garey and D. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness*. Freeman, 1979.

[7] K. Heljanko and A. Ştefănescu. Complexity results for checking distributed implementability. Technical Report 05/2004, Universität Stuttgart, 2004.
Available on-line at **http://www.fmi.uni-stuttgart.de/szs/people/stefanescu/tr-05-2004.pdf** .

[8] J. Hopcroft and J. Ullman. *Introduction to automata theory, languages, and computation*. Addison Wesley, 1979.

[9] R. Morin. Decompositions of asynchronous systems. In *CONCUR'98*, volume 1466 of *LNCS*, pages 549–564. Springer, 1998.

[10] R. Morin. Hierarchy of asynchronous automata. In *WDS'99*, volume 28 of *Electronic Notes in Theoretical Computer Science*, pages 59–75, 1999.

[11] M. Mukund. From global specifications to distributed implementations. In B. Caillaud, P. Darondeau, and L. Lavagno, editors, *Synthesis and control of discrete event systems*, pages 19–34. Kluwer, 2002.

[12] M. Nielsen, G. Rozenberg, and P. Thiagarajan. Elementary transition systems. *TCS*, 96:3–33, 1992.

[13] D. Peled, T. Wilke, and P. Wolper. An algorithmic approach for checking closure properties of temporal logic specifications and ω-regular languages. *TCS*, 195(2):183–203, 1998.

[14] S. Shukla, H. Hunt III, D. Rosenkrantz, and R. Stearns. On the complexity of relational problems for finite state processes. In *ICALP'96*, volume 1099 of *LNCS*, pages 466–477. Springer, 1996.

[15] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1–2):181–234, 2002.

[16] A. Ştefănescu, J. Esparza, and A. Muscholl. Synthesis of distributed algorithms using asynchronous automata. In R. Amadio and D. Lugiez, editors, *CONCUR'03*, volume 2761 of *LNCS*, pages 27–41. Springer, 2003.

[17] W. Vogler. Concurrent implementation of asynchronous transition systems. In *ICAPTN'99*, volume 1639 of *LNCS*, pages 284–303. Springer, 1999.

[18] W. Zielonka. Notes on finite asynchronous automata. *R.A.I.R.O. Inform. Théor. Appl.*, 21:99–135, 1987.

[19] W. Zielonka. Safe executions of recognizable trace languages by asynchronous automata. In *Logical Foundations of Computer Science*, volume 363 of *LNCS*, pages 278–289. Springer, 1989.