# Complexity Results for Checking Distributed Implementability

Keijo Heljanko[1] and Alin Ştefănescu[2]

[1] Laboratory for Theoretical Computer Science, Helsinki University of Technology
[2] Institute for Formal Methods in Computer Science, University of Stuttgart

**Abstract.** We consider the distributed implementability problem as: Given a labeled transition system $TS$ together with a distribution $\Delta$ of its actions over a set of processes, does there exist a distributed system over $\Delta$ such that its global transition system is 'equivalent' to $TS$? We consider the distributed system models of synchronous products of transition systems [Arn94] and asynchronous automata [Zie87]. 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 [CMT99,Mor99]. We also describe a logic programming implementation which complements the implementation for the synthesis of asynchronous automata initiated in [ŞEM03].

## 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 [Arn94], and Zielonka's asynchronous automata [Zie87]) using a number of 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 [ER90,NRT92,Mor98,CMT99,Vog99,Muk02]. Moreover, the computational complexity of the variants concerning elementary net systems and place/transition Petri nets is well understood [BBD95,BBD97]. 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 [CMT99,Mor99].

Mukund [Muk02] surveys (structural, behavioural) characterizations for synchronous products and asynchronous automata. In this paper we provide (the

**Table 1.** Implementability of synchronous products with one initial state

| Specification ($TS$) | Isomorphism | Language Equivalence | Bisim. (determ. impl.) |
|---|---|---|---|
| Nondeterministic | NP-complete | PSPACE-complete | PSPACE-complete |
| Deterministic | P [Mor98] | | |
| Acyclic & Nondet. | NP-complete | coNP-complete | coNP-complete |
| Acyclic & Determ. | P [Mor98] | | |

**Table 2.** Implementability of asynchronous automata with multiple initial states

| Specification ($TS$) | Isomorphism | Language Equivalence | Bisim. (determ. impl.) |
|---|---|---|---|
| Nondeterministic | NP-complete | PSPACE-complete | P |
| Deterministic | P [Mor98] | P | |
| Acyclic & Nondet. | NP-complete | coNP-complete | P |
| Acyclic & Determ. | P [Mor98] | P | |

missing) lower and upper bounds for all the implementability tests presented in [Muk02]. 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 [Mor98], Morin proved that 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 [CMT99,Mor99] show that the problem is in NP, but precise lower bounds were explicitly left open. We show that the problem is NP-complete, even for acyclic specifications.

In [Muk02], 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). We also obtain the same results for the implementability problem modulo bisimulation *when the implementation is required to be deterministic* (Table 1, column 4). (Notice that this is a natural constraint in many areas of hardware design.)

In [Zie89], Zielonka characterized the transition systems that can be implemented as an asynchronous automata modulo language equivalence. Combining this result with several others from the literature, we show that the implementability 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).

Partly motivated by the complexity results, in the last part of the paper we present new prototype implementations for asynchronous automata synthesis problems. The used approach is based on mapping the problems to the problem of finding a stable model of a logic program (an NP-complete problem) by using the SMODELS logic programming system [SNS02].

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 Section 6 discusses some heuristic implementations for asynchronous automata. The last section is reserved for conclusions and some of the technical details can be found in several appendices.

## 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, $\Sigma$ the nonempty, finite alphabet of *actions*, $\rightarrow \subseteq Q \times \Sigma \times Q$ the transition relation, and $I \subseteq Q$ the 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. All transition systems in this paper are considered to be finite and reachable.

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

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* [Arn94] and a generalization of them, *asynchronous automata* [Zie87].

Let $(\Sigma, \mathit{Proc}, \Delta)$ be a distribution. In the first synchronization model, we associate a local transition system with *each process* in $\mathit{Proc}$. A synchronization on a common action $a \in \Sigma$ occurs only when all the local states of the processes in $\mathit{dom}(a)$ 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 $\mathit{dom}(a)$ 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 $\mathit{dom}(a)$.

**Definition 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

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

**Definition 2.** 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

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

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

*Problem 3.* Given a distribution $(\Sigma, Proc, \Delta)$ and a finite transition system $TS$, does there exist a *distributed transition system* over $\Delta$ *equivalent* to $TS$?

Mukund [Muk02] surveyed solutions for the above problem when the 'distributed transition system' is one of {*synchronous product of transition systems*, *asynchronous automaton*} and 'equivalent' is one of {*isomorphic*, *language equivalent*, *bisimilar*[1]}. 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 [Muk02] 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. 1, 2) to have a set of initial states as opposed to only one initial state in [Muk02].

## 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.

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 [ŞEM03] for the synthesis of asynchronous automata).

[1] This case was only solved for *deterministic* distributed implementations [CMT99].

### 3.1 Synchronous Products of Transition Systems

The *theory of regions* [ER90] proposed an approach of solving the synthesis modulo isomorphism for Petri nets. Along the same lines goes Theorem 4 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. 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 4.** *[CMT99,Muk02] Let $(\Sigma, Proc, \Delta)$ be a distribution and $TS = (Q, \Sigma, \rightarrow, I)^2$ be a transition system. Then, TS is isomorphic to a synchronous product of transition systems over $\Delta$ 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 5.** *The implementability problem for synchronous products modulo isomorphism is NP-complete, even for acyclic specifications.*

*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 4 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

[2] In [CMT99,Muk02], Th. 4 is restricted to the case when $|I| = 1$. By inspecting the proof of [CMT99], it is easy to see that the theorem holds in fact for an arbitrary $I$.

chosen such that a clause will evaluate to `False` if and only if the condition $\mathrm{SP}_3$ will be violated for the triple associated to the given clause. The application of Theorem 4 finishes the job.

Let $\phi$ 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 $(\Sigma_\phi, \mathit{Proc}_\phi, \Delta_\phi)$ and a (nondeterministic) transition system $\mathit{TS}_\phi = (Q_\phi, \Sigma_\phi, \to_\phi, I_\phi)$ such that: $\phi$ is satisfiable if and only if $\mathit{TS}_\phi$ is isomorphic to a synchronous product of transition systems over $\Delta_\phi$. To relieve a bit the notation, we will drop all $\phi$ indices.

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

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

Then, the set $\Sigma$ of actions and their domains (which determine $\Delta$) consist of:

- one action for each variable: $\{a_{x_i} \mid i \in [1..n]\}$ with $\mathit{dom}(a_{x_i}) := \{p_{x_i}, p_{\overline{x_i}}\}$.
- 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 $\mathit{dom}(a_{x_i c_j}) = \mathit{dom}(a'_{x_i c_j}) := \mathit{Proc} \setminus \{p_{x_i}\}$.
- 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 $\mathit{dom}(a_{\overline{x_i} c_j}) = \mathit{dom}(a'_{\overline{x_i} c_j}) := \mathit{Proc} \setminus \{p_{\overline{x_i}}\}$.
- two actions for each clause: $\{a_{c_j}, a'_{c_j} \mid j \in [1..m]\}$ with $\mathit{dom}(a_{c_j}) := \{p_{c_j}\} \cup \{p_{x_i} \mid x_i \in c_j\} \cup \{p_{\overline{x_i}} \mid \overline{x_i} \in c_j\}$ (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 $\mathit{dom}(a'_{c_j}) := \mathit{Proc} \setminus \{p_{c_j}\}$.

Last, we construct the transition system $\mathit{TS}$. 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 $\to \subseteq Q \times \Sigma \times Q$ 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, $I := \{q^0_{x_i} \mid i \in [1..n]\} \cup \{q'_{c_j} \mid j \in [1..m]\}$. [3]

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

[3] It is easy to modify the construction such that there is only *one initial state*.

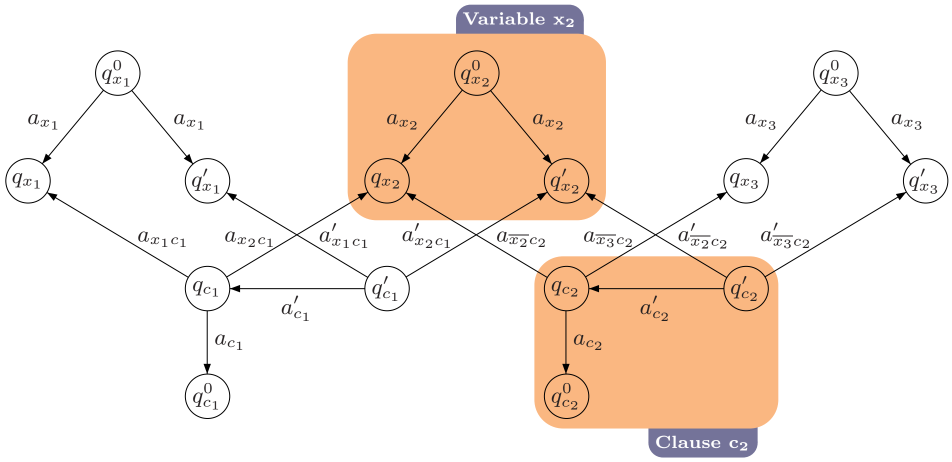



**Fig. 1.** The transition system $TS$ associated to $\phi = (x_1 \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3})$

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 be 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 $a_{c_j}$. Thus, a clause $c_j$ will have all the literals evaluated to `False` if and only if the condition $\mathrm{SP}_3$ will be violated for $a := a_{c_j}$, $q := q'_{c_j}$, and $q_p := q_{c_j}$, $q'_p := q^0_{c_j}$ for all $p \in dom(a_{c_j})$.

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

*First Implication.* We first prove the easier part: $\phi$ is not satisfiable implies $TS$ is not isomorphic to a synchronous product of transition systems over $\Delta$. If $\phi$ is not satisfiable, then for any assignment of the variables $x_1, \ldots, x_n$ there exists a clause that is evaluated to `False`. We must show that in this case, there are no $(\equiv_p)_{p \in Proc}$ satisfying all $\mathrm{SP}_1$–$\mathrm{SP}_3$.

By contradiction, assume that there exist $(\equiv_p)_{p \in Proc}$ satisfying all $\mathrm{SP}_1$–$\mathrm{SP}_3$. For each $i \in [1..n]$, we use first the condition $\mathrm{SP}_3$ 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 $\mathrm{SP}_3$ 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 $\mathrm{SP}_3$ 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 $\mathrm{SP}_1$ 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 $\mathrm{SP}_2$).

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

$$x_i \text{ is evaluated to } \texttt{False} \text{ if and only if } q_{x_i} \equiv_{p_{x_i}} q'_{x_i}.$$

Since $\phi$ 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 $\mathrm{SP}_1$, 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 $\mathrm{SP}_1$ 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 $\mathrm{SP}_3$, 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.* We move now to the second part of the proof, assuming that $\phi$ is satisfiable. Then, there exists an assignment to the variables $x_1, \ldots, x_n$ such that each clause is `True`. We prove that $TS$ is isomorphic to a synchronous product of transition systems over $(\Sigma, Proc, \Delta)$. For that, we construct a family of equivalences $(\equiv_p)_{p \in Proc}$ in the following way:

**Step 0** For each $p \in Proc$, initialize the binary relation $\equiv_p \subseteq Q \times Q$ to $\emptyset$.
**Step 1** For each $q \xrightarrow{a} q'$, choose $q \equiv_p q'$ for each $p \in Proc \setminus dom(a)$.
**Step 2** For each $i \in [1..n]$, if variable $x_i$ is evaluated to `False`, then choose $q_{x_i} \equiv_{p_{x_i}} q'_{x_i}$, otherwise choose $q_{x_i} \equiv_{p_{\overline{x_i}}} q'_{x_i}$.
**Step 3** For each $p \in Proc$, close $\equiv_p$ under reflexivity, symmetry, and transitivity.

Table 3 describes the equivalence classes of our equivalences generated above. Each cell gives the partition of the state space $Q$ into the equivalence classes for each type of process (rows) depending on the value of the associated variable (columns). Each equivalence class is given as a set in curly brackets. It is tedious, but not hard to check the correctness of Table 3 (i.e., the equivalence classes presented are the ones generated by the Steps 0–3 above). Based on it, we prove that the chosen $(\equiv_p)_{p \in Proc}$ satisfy $\mathrm{SP}_3$, $\mathrm{SP}_2$, and $\mathrm{SP}_1$ (in this order):

*Remark 6.* [4] $\mathrm{SP}_3$ holds for action $a := a_{x_i}$, for each $i \in [1..n]$.

Note that each action $a \in \Sigma \setminus \{a_{x_i} \mid i \in [1..n]\}$ has the property that there exists *only one* transition labeled with $a$ in the transition system $TS$. In this case, the condition $\mathrm{SP}_3$ can be simplified as below:

[4]The proof can be found in the Appendix A.1.

**Table 3.** The equivalence classes constructed in the 2nd part of the proof of Th. 5

| | $x_i = \texttt{False}$ | $x_i = \texttt{True}$ |
|---|---|---|
| $\equiv_{p_{x_i}}$ | $\{q^0_{x_i}\}$, $\{q^0_{x_{i'}}, q_{x_{i'}}, q'_{x_{i'}}\}$ for each $i' \neq i$,<br>$\{q_{x_i}, q'_{x_i}\} \cup \{q_{c_j}, q'_{c_j} \mid x_i \in c_j\}$,<br>$\{q^0_{c_j}\}$ for each $c_j$ containing lit. $x_i$,<br>and $\{q_{c_{j'}}, q^0_{c_{j'}}\}$, $\{q'_{c_{j'}}\}$ for each $c_{j'}$<br>not containing the positive literal $x_i$ | $\{q^0_{x_i}\}$, $\{q^0_{x_{i'}}, q_{x_{i'}}, q'_{x_{i'}}\}$ for each $i' \neq i$,<br>$\{q_{x_i}\} \cup \{q_{c_j} \mid x_i \in c_j\}$,<br>$\{q'_{x_i}\} \cup \{q'_{c_j} \mid x_i \in c_j\}$,<br>$\{q^0_{c_j}\}$ for each $c_j$ containing lit. $x_i$,<br>and $\{q_{c_{j'}}, q^0_{c_{j'}}\}$, $\{q'_{c_{j'}}\}$ for each $c_{j'}$<br>not containing the positive literal $x_i$ |
| $\equiv_{p_{\overline{x_i}}}$ | $\{q^0_{x_i}\}$, $\{q^0_{x_{i'}}, q_{x_{i'}}, q'_{x_{i'}}\}$ for each $i' \neq i$,<br>$\{q_{x_i}\} \cup \{q_{c_j} \mid \overline{x_i} \in c_j\}$,<br>$\{q'_{x_i}\} \cup \{q'_{c_j} \mid \overline{x_i} \in c_j\}$,<br>$\{q^0_{c_j}\}$ for each $c_j$ containing lit. $\overline{x_i}$,<br>and $\{q_{c_{j'}}, q^0_{c_{j'}}\}$, $\{q'_{c_{j'}}\}$ for each $c_{j'}$<br>not containing the negative literal $\overline{x_i}$ | $\{q^0_{x_i}\}$, $\{q^0_{x_{i'}}, q_{x_{i'}}, q'_{x_{i'}}\}$ for each $i' \neq i$,<br>$\{q_{x_i}, q'_{x_i}\} \cup \{q_{c_j}, q'_{c_j} \mid \overline{x_i} \in c_j\}$,<br>$\{q^0_{c_j}\}$ for each $c_j$ containing lit. $\overline{x_i}$,<br>and $\{q_{c_{j'}}, q^0_{c_{j'}}\}$, $\{q'_{c_{j'}}\}$ for each $c_{j'}$<br>not containing the negative literal $\overline{x_i}$ |
| $\equiv_{p_{c_j}}$ | $\{q^0_{x_i}, q_{x_i}, q'_{x_i}\}$ for $i \in [1..n]$, $\{q_{c_j}, q'_{c_j}\}$, $\{q^0_{c_j}\}$, and $\{q_{c_{j'}}, q^0_{c_{j'}}\}$, $\{q'_{c_{j'}}\}$ for $j' \neq j$ | |

**Table 4.** Details for the satisfaction of the $\mathrm{SP}_3$ property

| $a$ | $q$ | Why $\mathrm{SP}_3$ holds (a process $p$ from $dom(a)$ is given) |
|---|---|---|
| $a_{x_i}$ | See Remark 6. | |
| $a_{x_i c_j}$ (for $x_i \in c_j$) | $\{q^0_{c_j}\}$ | $p := p_{c_j}$. Indeed, $q_{c_j} \not\equiv_{p_{c_j}} q^0_{c_j}$. |
| | $Q \setminus \{q_{c_j}, q^0_{c_j}\}$ | $p := p_{\overline{x_i}}$. Indeed, $x_i \in c_j$ implies $\overline{x_i} \notin c_j$ (see construction of $\phi$), so $q_{c_j} \not\equiv_{p_{\overline{x_i}}} q$, $\forall q \in Q \setminus \{q_{c_j}, q^0_{c_j}\}$. |
| $a'_{x_i c_j}$ (for $x_i \in c_j$) | $Q \setminus \{q'_{c_j}\}$ | $p := p_{\overline{x_i}}$. Same as above, $x_i \in c_j$ implies $\overline{x_i} \notin c_j$ and in this case $q'_{c_j} \not\equiv_{p_{\overline{x_i}}} q$, $\forall q \in Q \setminus \{q'_{c_j}\}$. |
| $a_{\overline{x_i} c_j}$, $a'_{\overline{x_i} c_j}$ (for $\overline{x_i} \in c_j$) | Similar to the cases $a_{x_i c_j}$, $a'_{x_i c_j}$ above. | |
| $a_{c_j}$ | $\{q'_{c_j}\}$ | Since $c_j$ evaluates to `True`, there exists a literal $\ell$ of $c_j$ evaluated to `True`. Assume $\ell = x_i$, such that $x_i \in c_j$ and $x_i = \texttt{True}$ (a similar analysis is made if $\ell$ is negative). Then, for $p := p_{x_i}$ we have that $p_{x_i} \in dom(a_{c_j})$ and $q_{c_j} \not\equiv_{p_{x_i}} q'_{c_j}$. |
| | $Q \setminus \{q_{c_j}, q'_{c_j}\}$ | $p := p_{c_j}$. Indeed, $q_{c_j} \not\equiv_{p_{c_j}} q$, $\forall q \in Q \setminus \{q_{c_j}, q'_{c_j}\}$. |
| $a'_{c_j}$ | $Q \setminus \{q'_{c_j}\}$ | For a given variable $x_i$, the literals $x_i$ and $\overline{x_i}$ cannot appear both in $c_j$. If $x_i \notin c_j$, then $p := p_{x_i}$ (and indeed, $q'_{c_j} \not\equiv_{p_{x_i}} q$, $\forall q \in Q \setminus \{q'_{c_j}\}$). If $\overline{x_i} \notin c_j$, then $p := p_{\overline{x_i}}$. |

**Table 5.** Details for the $\mathrm{SP}_2$ property (only cases not solved already by Table 4)

| $q_1$ | $q_2$ | Why $\mathrm{SP}_2$ holds (a process $p$ from *Proc* is given) |
|---|---|---|
| $q^0_{x_i}$ | $Q' \setminus \{q^0_{x_i}\}$ | $p := p_{x_i}$. Indeed, $q^0_{x_i} \not\equiv_{p_{x_i}} q$, $\forall q \in Q' \setminus \{q^0_{x_i}\}$. |
| $q_{x_i}$ | $\{q^0_{x_i}, q'_{x_i}\}$ | If $x_i$ is `True` then $p := p_{x_i}$ else $p := p_{\overline{x_i}}$. |
| | $Q' \setminus \{q^0_{x_i}, q_{x_i}, q'_{x_i}\}$ | $p := p_{c_j}$ for an arbitrary $j \in [1..m]$. |
| $q'_{x_i}$ | Similar to the case $q_{x_i}$ above. | |
| $q^0_{c_j}$ | $Q' \setminus \{q^0_{c_j}\}$ | $p := p_{c_j}$. |

*Remark 7.* [4] Let $a \in \Sigma$ such that there is only one transition, say $q_a \xrightarrow{a} q'_a$, labeled with $a$ in $TS$. Then, $\mathrm{SP}_3$ holds for the chosen $a$ if and only if for each state $q \neq q_a$, there exists a process $p \in dom(a)$ such that $q \not\equiv_p q_a$.

Since Remark 6 shows that $\mathrm{SP}_3$ holds for $a \in \{a_{x_i} \mid i \in [1..n]\}$, the remaining cases are solved in Table 4 using Remark 7. The first column picks a value for $a$, while the second one gives a range to the state $q \in Q \setminus \{q_a\}$ from the formulation of Remark 7. In the last column, we give a process $p \in dom(a)$ such that $q_a \not\equiv_p q$, for all $q$ in the range given by second column. The correctness of the solutions provided is verified using the equivalence classes given in Table 3.

*Remark 8.* $\mathrm{SP}_2$ holds iff for each $q_1 \neq q_2$, there exists $p \in Proc$ with $q_1 \not\equiv_p q_2$.

Table 5 presents only the cases for which Table 4 did not give a process to 'distinguish' two different states $q_1$ (column 1) and $q_2$ (in the range given in column 2), i.e., we give a process $p \in Proc$ such that $q_1 \not\equiv_p q_2$. More precisely, we only have to consider pairs of states from the subset:

$$Q' := \{q^0_{x_i}, q_{x_i}, q'_{x_i} \mid i \in [1..n]\} \cup \{q^0_{c_j} \mid j \in [1..m]\}.$$

Condition $\mathrm{SP}_1$ is fulfilled by construction (Step 1). □

Going into the proof details of Theorem 4 given in [CMT99], we can show that if there exist a set of equivalences $(\equiv_p)_{p \in Proc}$ satisfying only conditions $\mathrm{SP}_1$ and $\mathrm{SP}_3$ (but not necessarily $\mathrm{SP}_2$), then we can synthesize a synchronous product of transition systems accepting the same language as the initial transition system.[5] 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 the Appendix A.2):

**Corollary 9.** *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 conditions $\mathrm{SP}_1$ and $\mathrm{SP}_3$ of Theorem 4, is NP-complete.*

**Proposition 10.** *[Mor98] The implementability problem for synchronous products modulo isomorphism becomes decidable in* polynomial *time, if the input transition system is* deterministic*.*

### 3.2 Asynchronous Automata

Similar results to those for synchronous products of transition systems hold for asynchronous automata. The NP-completeness proof of Theorem 12 has a similar structure to the proof of Theorem 5, but the details are different. The proof can be found in the Appendix A.3.

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

**Theorem 11.** *[Mor99,Muk02] Let $(\Sigma, Proc, \Delta)$ be a distribution and $TS = (Q, \Sigma, \rightarrow, I)$ be a transition system. Then, $TS$ is isomorphic to an asynchronous automaton over $\Delta$ 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:*

$\mathrm{AA}_1$ : *If $q_1 \xrightarrow{a} q_2$, then $q_1 \equiv_{Proc \setminus dom(a)} q_2$.*
$\mathrm{AA}_2$ : *If $q_1 \equiv_{Proc} q_2$, then $q_1 = q_2$.*
$\mathrm{AA}_3$ : *If $q_1 \xrightarrow{a} q_1'$ and $q_1 \equiv_{dom(a)} q_2$, then there exists $q_2'$ such that $q_2 \xrightarrow{a} q_2'$ and $q_1' \equiv_{dom(a)} q_2'$.*

**Theorem 12.** *The implementability problem for asynchronous automata modulo isomorphism is NP-complete, even for acyclic specifications.*

**Corollary 13.** *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 conditions* $\mathrm{AA}_1$ *and* $\mathrm{AA}_3$ *of Theorem 11, is NP-complete.*

**Proposition 14.** *[Mor98] The implementability problem for asynchronous automata modulo isomorphism becomes decidable in* polynomial *time, if the input transition system is* deterministic.

## 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, we have:

**Lemma 15.** *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 to be 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 [Muk02] works only for the class of synchronous products with just one initial state ($|I| = 1$ in Def. 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 16.* Given a distribution $(\Sigma, Proc, \Delta)$ and a finite transition system $TS$, does there exist a synchronous product of transition systems over $\Delta$ with only *one initial state* that is language equivalent to $TS$?

We present, following [Muk02], the algorithm of deciding Problem 16:
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 $TS = (Q, \Sigma, \rightarrow, \{q^{in}\})$. For each process $p \in Proc$, we construct a projection $TS_p := (Q, \Sigma_{loc}(p), \rightarrow_p, \{q^{in}\})$ obtained from a copy of $TS$ in which the labels from $\Sigma \setminus \Sigma_{loc}(p)$ are replaced by $\varepsilon$ and $\rightarrow_p$ is the $\varepsilon$-closure of $\rightarrow$ (a polynomial algorithm for $\varepsilon$-closure can be found in [HU79, Chap. 2.4]).
3. Problem 16 has a positive answer iff $TS$ is language equivalent to the synchronous product over $\Delta$ of the transition systems $(TS_p)_{p\in Proc}$ with one global initial state $(q^{in}, \ldots, q^{in})$.

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

*Problem 17.* (*Reachability in synchronous products*) Given $(\Sigma, Proc, \Delta)$ a distribution, a set of local transition systems $(TS_p)_{p\in Proc}$ with $TS_p = (Q_p, \Sigma_{loc}(p), \rightarrow_p, \{q_p^{in}\})$, 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 $\rightarrow_p$'s as in Def. 1?

**Lemma 18.** *The non-reachability problem (i.e., the complement of Problem 17) for synchronous products can be in polynomial time reduced to Problem 16.*

*Proof.* Given a distribution $(\Sigma, Proc, \Delta)$, we suppose $Proc := \{1, \ldots, n\}$. Also, we are given a local transition system $TS_p = (Q_p, \Sigma_{loc}(p), \rightarrow_p, \{q_p^{in}\})$ 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 16 has a solution for $\Delta'$ and $R$ if and only if the global state $q := (q_1, \ldots, q_n)$ is not reachable from the global initial state $(q_1^{in}, \ldots, q_n^{in})$.

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:
  - $\Sigma'_{loc}(p) := \Sigma_{loc}(p) \cup \{a_{p'} \mid p' \in [1..n] \wedge p' \neq p\} \cup \{\surd\}$ for $p \in [1..n]$ and
  - $\Sigma'_{loc}(p_0) := \Sigma' \setminus \{\surd\}$.

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

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

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

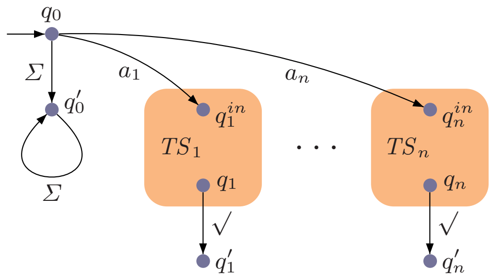



**Fig. 2.** A schematic representation of the reduction in Lemma 18

*Remark 19.* According to Step 2 in the decision algorithm for Problem 16, 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 $\varepsilon$ and applying an $\varepsilon$-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 $\Delta'$ 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 $\Delta$.

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 $\Delta'$ of its projections $R_p$ as described in the decision algorithm for Problem 16:

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. 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 19. 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 $\Delta'$.

*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 $\Delta'$. 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 19, $L(R_{p_0}) \subseteq L(R)$, 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 $v \in L(R)$.

$\square$

The next results are based on the complexity results for checking non-reachability and language equivalence of synchronous products from [SHRS96]:

**Theorem 20.** *The implementability problem for synchronous products with $|I| = 1$ modulo language equivalence is PSPACE-complete.*

*Proof.* The PSPACE-hardness follows from Lemma 18 and the PSPACE-hardness of the non-reachability problem for synchronous products [SHRS96, Th. 3.10].

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 [SHRS96, Th. 3.12]. $\square$

**Proposition 21.** *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 [SHRS96, Th. 3.10] works in fact for *deterministic* $TS_p$'s. The reduction of Lemma 18 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 [SHRS96, Th. 3.16] and from Lemma 18 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 [SHRS96, 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 [Zie87] 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 $\Sigma$: $a \| b$ 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 [Zie87], 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 [Zie89] 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. [6] 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 [Zie89] we easily get the following characterization (see proof in the Appendix A.4):

**Proposition 22.** *A language $T \subseteq \Sigma^*$ is accepted by an asynchronous automaton if and only if $T$ is a prefix-closed regular trace language.*

**Theorem 23.** *The implementability problem for asynchronous automata modulo language equivalence is PSPACE-complete.*

*Proof.* Our implementability problem is in PSPACE, due to Proposition 22 and [PWW98, 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.

For the PSPACE-hardness part, we use a simple reduction from the totality problem '$= \Sigma^*$?' for nondeterministic finite automata, which is known to be

[6] However, for a regular trace language, there exists a *deterministic* asynchronous automaton accepting it [Zie87], but not necessarily a deterministic safe one [Zie89].

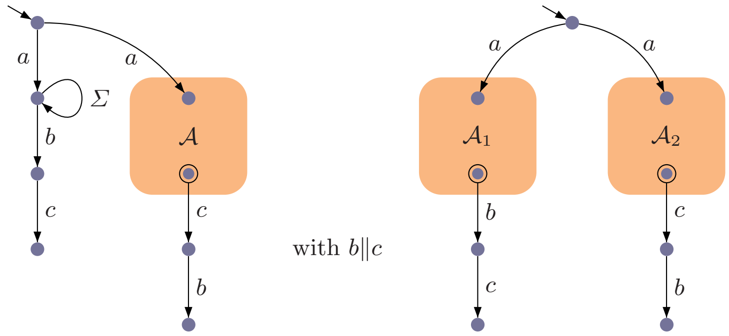



**Fig. 3.** Schematic representations of the reductions in Th. 23 (left) and Prop. 24 (right)

PSPACE-hard [GJ79]. For each nondeterministic finite automaton $\mathcal{A}$ over $\Sigma$, which we can suppose w.l.o.g. that has only one initial $q^{in}$, 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 22). The details are given in the Appendix A.4. □

**Proposition 24.** *The implementability problem for asynchronous automata modulo language equivalence for* deterministic *specifications is decidable in polynomial time. For* acyclic *specifications, the problem is coNP-complete.*

*Proof.* The first part follows directly from Proposition 22 and [PWW98, 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 [GJ79]. The proof is similar to the proof of Theorem 23 and the details are given in the Appendix A.4. □

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 complexity bounds for synchronous products with multiple initial states. (The proof details are given in the Appendix A.4.)

**Theorem 25.** *The implementability problem for synchronous products of transition systems modulo language equivalence is PSPACE-hard. For* acyclic *specifications, the problem is coNP-complete.*

However, we suspect that the above general problem is much harder than PSPACE. Moreover, we do not know anything about its complexity when the specification is deterministic.

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

The proof (given in the Appendix A.4) uses the fact that the set of invalid computations of a Turing machine is a context-free language [HU79, Lemma 8.7], together with the trick of making the first two letters of an accepting computation independent (see the proof technique of [PWW98, Th. 11]). Given the fact that the language of any distributed system exhibiting independence of actions is a trace language, one could dare to interpret this result in a programming language framework as: The synthesis problem for distributed programs with (possibly recursive) *procedures* and variables over finite domains is undecidable.

## 5 Deterministic Implementability modulo Bisimulation

Based on the observation that bisimilarity and language equivalence coincide for deterministic transitions systems, [Muk02] 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 21 and 24. Appendix A.5 gives the details.

However, the synthesis problem modulo bisimulation is still open in the case of nondeterministic implementations.

## 6 Prototype Implementations

Comparing the deterministic and nondeterministic assumptions, dealing with deterministic specifications/implementations has a computational advantage. Nevertheless, it is worth considering also the nondeterministic case: In case the original specification is given as a regular language, a nondeterministic transition system exhibiting the given behavior may be exponentially more succinct than a deterministic one with the same behavior. Moreover, the nondeterministic distributed systems that we considered are *strictly more expressive* than their deterministic counterparts as the simple example below shows:

*Example 27.* Let us choose the language $L = \{\varepsilon, a, b\}$ with the distribution $\Sigma := \{a, b\}$, $Proc := \{1, 2\}$, and $\Delta := \{(a, 1), (b, 2)\}$. $L$ is accepted by a nondeterministic synchronous product, but not by a deterministic one and $L$ is accepted by a nondeterministic asynchronous automaton, but not by a deterministic one.

Briefly, $L$ is accepted by the nondeterministic transition system of Fig. 4 which is isomorphic to a synchronous product (using Th. 4). On the other hand, the only deterministic transition systems accepting $L$ are given in Fig. 4, but none of them is isomorphic to a synchronous product (using again Th. 4). The same holds for asynchronous automata (using Th. 11).

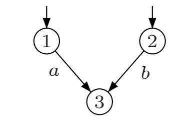
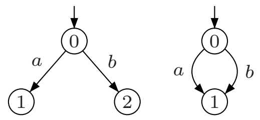
| Distribution | Nondeterministic | Deterministic |
|---|---|---|
| $\Sigma := \{a, b\}$<br>$Proc := \{1, 2\}$<br>$\Delta := \{(a, 1), (b, 2)\}$ |  |  |

**Fig. 4.** Transition systems for Example 27

In this section, we will present some prototype implementations intended to complement the tool Synasync [ŞEM03] which synthesizes asynchronous automata from deterministic specifications. [7] More precisely, we consider the NP-complete test provided by Theorem 11 (working for general transition systems) and use it as a heuristic for the synthesis of asynchronous automata. Moreover, we provide an alternative implementation for another NP-complete problem (namely, finding an implementable subautomaton) involved in Synasync.

We used a constraint-based logic programming framework called Smodels (http://www.tcs.hut.fi/Software/smodels/). It consists of `smodels`, an efficient implementation of the stable model semantics for normal logic programs, and `lparse`, a grounder front-end that transforms normal logic programs (with variables) to ground logic programs (without variables). We translate the synthesis problem into the problem of finding a stable model of a logic program. The program itself is written in the input syntax of `lparse`. The synthesis solutions (if any!) are given as stable models of the input program. A more detailed description of the Smodels system can be found in the `lparse 1.0` User's Manual (http://www.tcs.hut.fi/Software/smodels/lparse.ps) and a general tutorial on the stable model semantics in [SNS02].

We chose to use logic programs with stable model semantics over the more widely used SAT for the main reason that in the stable model semantics the notion of a least fixpoint is trivial to express (we need this e.g. for the reachability property). However, in a SAT encoding we would have to encode the least fixpoint computation procedure itself (e.g. as a series of approximations) which would have resulted in additional blow-up of the translation not to mention implementation effort. An additional advantage of using Smodels is that the code is very concise. The actual implementation of synthesis without any optimizations has no more than 20 lines of code and it is given in the Appendix B.1.

The existing tools implementing the notion of regions for the synthesis of a distributed system (see Synet [Cai97], Petrify [CKK$^+$97], and Synasync [ŞEM03]) do not handle nondeterministic specifications. In this sense, we can say that our implementation widens the class of systems tackled. Of course, one can always determinize the specification and the behavior is preserved, but then the tests of *isomorphism* to a distributed system are incomparable w.r.t. the two inputs because the *structure* is changed in the process of determinization.

[7] The implementation for synchronous products is left for future investigations.

Experience has shown that in most cases the initial specification is not isomorphic to the global state space of a distributed system (see, for example, [Cai97,ŞEM03,Yak98]). Several approaches to tackle this problem have been proposed (most of them are at least NP-hard and at the level of heuristics): label-splitting, introduction of silent events (both not really suitable for our framework[8]), changing the distribution, cutting transitions followed by unfolding. We opt for the last type of heuristics which proved fruitful in the experiments of [ŞEM03] in the context of asynchronous automata.

We introduce first some definitions needed in the presentation of the heuristic. Let $TS = (Q, \Sigma, \rightarrow, I)$ be a transition system. $TS$ is *deadlock-free* if: $\forall q \in Q \, \exists a \in \Sigma,\ q' \in Q : q \xrightarrow{a} q'$. The set of actions *appearing* in $TS$ is $\Sigma(TS) := \{a \in \Sigma \mid \exists q, q' \in Q : q$ reachable and $q \xrightarrow{a} q'\}$. Also, we say that $TS' = (Q', \Sigma, \rightarrow', I')$ is a *subautomaton* of $TS$ if $Q' \subseteq Q$, $I' \subseteq I$, and $\rightarrow' \subseteq \rightarrow$. Finally, $TS$ satisfies the *independent* and *forward diamond rules* if:

**ID** : $q_1 \xrightarrow{a} q_2 \xrightarrow{b} q_4 \wedge a \| b \Rightarrow \exists q_3 : q_1 \xrightarrow{b} q_3 \xrightarrow{a} q_4$, and

**FD** : $q_1 \xrightarrow{a} q_2 \wedge q_1 \xrightarrow{b} q_3 \wedge a \| b \Rightarrow \exists q_4 : q_2 \xrightarrow{b} q_4 \wedge q_3 \xrightarrow{a} q_4$.

(The global transition system of a distributed system satisfies ID and FD.)

The (relaxed) synthesis problem considered in [ŞEM03] is: Given a distribution $\Delta$ and a *deterministic* transition system $TS$, synthesize an asynchronous automaton $\mathcal{AA}$ over $\Delta$ such that $L(\mathcal{AA}) \subseteq L(TS)$ and $\Sigma(\mathcal{AA}) = \Sigma(TS)$ [9]. This problem was shown to be undecidable in general and the heuristic proposed was to search only for asynchronous automata language equivalent to a *subautomaton* $\mathcal{A}$ of $TS$ satisfying ID and FD. Therefore, the implementation of Synasync [ŞEM03] includes a heuristic to solve the following NP-complete problem:

*Problem 28.* (SubautIDFD) Given a transition system $TS$, find a subautomaton $\mathcal{A}$ with $\Sigma(\mathcal{A}) = \Sigma(TS)$, reachable, deadlock-free, and satisfying ID and FD.

The basic idea is then to 'unfold' the subautomaton into an asynchronous automaton accepting the same language (see [ŞEM03]). Here we propose the following new heuristic, which has also been implemented with Smodels, that tries to *directly* find an asynchronous automaton in the class of subautomata (with this heuristic, we try to avoid the unfolding procedure which may, theoretically, produce asynchronous automata superexponentially larger than the input):

*Problem 29.* (SubautAsync) Given a transition system $TS$, find a subautomaton $\mathcal{A}$ with $\Sigma(\mathcal{A}) = \Sigma(TS)$, reachable, deadlock-free, and isomorphic to an asynchronous automaton.

The main idea behind the implementation is to 'guess' the transitions of the subautomaton and then to test the required properties. The logic program code was optimized to include some rudimentary symmetry reductions and the relaxation obtained by dropping the condition $\mathrm{AA}_2$ (see Corollary 13).

[8] The label splitting cannot solve the conflicts without changing the distribution and the silent events must have the whole set *Proc* as domain, and so we force a global synchronization, which is not something that we would like do in a concurrent setting.

[9] This condition is used to rule out some unwanted (trivial) synthesis solutions.

We have implemented using SMODELS the above heuristics SubautIDFD and SubautAsync (Problems 28,29) and applied them to parametrized benchmarks of the mutual exclusion and dining philosophers problems. The experimental results can be found in the Appendix B.2 together with a short discussion.

We end the section with some remarks. There are alternative ways to find stable models of a logic program. The CMODELS2 system is a viable alternative to SMODELS in this domain as shown by the experiments based on earlier versions of our synthesis implementations [GLM04] (neither of the systems is consistently better than the other). It would also be interesting to see how a recently introduced new translation of the stable model semantics to SAT [Jan03] performs on our examples.

## 7 Conclusions

A summary of the complexity results obtained is presented in the introduction. We discover that the models of synchronous products of transition systems and asynchronous automata have similar 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 16). 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 [ŞEM03], solutions for the synthesis problem were obtained for asynchronous automata, but not for synchronous products, due to the expressiveness power of the former).

*Acknowledgments* We are grateful to Javier Esparza and Petr Jančar for useful discussions. The financial support from Academy of Finland (Project 53695, 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


[Arn94] A. Arnold. *Finite transition systems and semantics of communicating systems*. Prentice Hall, 1994.

[BBD95] 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.

[BBD97] E. Badouel, L. Bernardinello, and P. Darondeau. The synthesis problem for elementary net systems is NP-complete. *TCS*, 186(1–2):107–134, 1997.

[Cai97] B. Caillaud. SYNET: un outil de synthèse de réseaux de Petri bornés, applications. Technical Report 3155, INRIA, 1997.

[CKK+97] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. PETRIFY: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Trans. Information and Systems*, E80-D(3):315–325, 1997.

[CMT99] I. Castellani, M. Mukund, and P.S. Thiagarajan. Synthesizing distributed transition systems from global specifications. In *FSTTCS19*, volume 1739 of *LNCS*, pages 219–231. Springer, 1999.

[Dub86] C. Duboc. Mixed product and asynchronous automata. *TCS*, 48:183–199, 1986.

[ER90] A. Ehrenfeucht and G. Rozenberg. Partial (set) 2-structures I and II. *Acta Informatica*, 27(4):315–368, 1990.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness*. Freeman, 1979.

[GLM04] E. Giunchiglia, Y. Lierler, and M. Maratea. SAT-based answer set programming. In *Proc. of AAAI'04*, pages 61–66. AAAI Press, 2004.

[HU79] J. Hopcroft and J. Ullman. *Introduction to automata theory, languages, and computation*. Addison Wesley, 1979.

[Jan03] T. Janhunen. A counter-based approach to translating logic programs into set of clauses. In *Proceedings of the 2nd International Workshop on Answer Set Programming (ASP'03)*, volume 78, pages 166–180. Sun SITE Central Europe (CEUR), 2003.

[Mor98] R. Morin. Decompositions of asynchronous systems. In *CONCUR'98*, volume 1466 of *LNCS*, pages 549–564. Springer, 1998.

[Mor99] R. Morin. Hierarchy of asynchronous automata. In *WDS'99*, volume 28 of *Electronic Notes in Theoretical Computer Science*, pages 59–75, 1999.

[Muk02] 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.

[Mus94] A. Muscholl. *Über die Erkennbarkeit unendlicher Spuren*. PhD thesis, Universität Stuttgart, 1994. Published by Teubner, 1996.

[NRT92] M. Nielsen, G. Rozenberg, and P.S. Thiagarajan. Elementary transition systems. *TCS*, 96:3–33, 1992.

[PWW98] D. Peled, T. Wilke, and P. Wolper. An algorithmic approach for checking closure properties of temporal logic specifications and $\omega$-regular languages. *TCS*, 195(2):183–203, 1998.

[ŞEM03] 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.

[SHRS96] S.K. Shukla, H.B. Hunt III, D.J. Rosenkrantz, and R.E. Stearns. On the complexity of relational problems for finite state processes. In *ICALP'96*, volume 1099 of *LNCS*, pages 466–477. Springer, 1996.

[SNS02] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1–2):181–234, 2002.

[Vog99] W. Vogler. Concurrent implementation of asynchronous transition systems. In *ICAPTN'99*, volume 1639 of *LNCS*, pages 284–303. Springer, 1999.

[Yak98] A. Yakovlev. Designing control logic for counterflow pipeline processor using Petri nets. *Formal Methods in System Design*, 12(1):39–71, 1998.

[Zie87] W. Zielonka. Notes on finite asynchronous automata. *R.A.I.R.O. Inform. Théor. Appl.*, 21:99–135, 1987.

[Zie89] 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.

# A Appendix

## A.1 Proofs of Remarks 6 and 7

*Proof of Remark 6.*
Let $i \in [1..n]$ and $q \in Q$. Following $SP_3$, for each $p \in dom(a_{x_i}) = \{p_{x_i}, p_{\overline{x_i}}\}$, we try to choose $q_p$ and $q'_p$ such that $q_p \xrightarrow{a_{x_i}} q'_p$ and $q \equiv_p q_p$. Since the only transitions labeled with $a_{x_i}$ are $q^0_{x_i} \xrightarrow{a_{x_i}} q_{x_i}$ and $q^0_{x_i} \xrightarrow{a_{x_i}} q'_{x_i}$, $q_p$ must be equal to $q^0_{x_i}$ and $q'_p$ must belong to $\{q_{x_i}, q'_{x_i}\}$, for each $p \in dom(a_{x_i})$. Moreover, if in $q \equiv_p q_p = q^0_{x_i}$, we instantiate $p$ with $p_{x_i}$ (from $dom(a_{x_i})$), we obtain $q \equiv_{p_{x_i}} q^0_{x_i}$. Since the $\equiv_{p_{x_i}}$-equivalence class of $q^0_{x_i}$ is $\{q^0_{x_i}\}$ (see Table 3), we deduce that $q = q^0_{x_i}$. We have now that the hypotheses of $SP_3$ are satisfied.

For the various choices for $q'_p \in \{q_{x_i}, q'_{x_i}\}$ with $p \in \{p_{x_i}, p_{\overline{x_i}}\}$, we will give a $q'$ such that $q = q^0_{x_i} \xrightarrow{a_{x_i}} q'$ and $q' \equiv_p q'_p$ for each $p \in \{p_{x_i}, p_{\overline{x_i}}\}$: If $q'_{p_{x_i}} = q'_{p_{\overline{x_i}}} := q_{x_i}$, take $q' := q_{x_i}$. If $q'_{p_{x_i}} = q'_{p_{\overline{x_i}}} := q'_{x_i}$, take $q' := q'_{x_i}$. If $q'_{p_{x_i}} := q_{x_i}$ and $q'_{p_{\overline{x_i}}} := q'_{x_i}$, we have two subcases: If $x_i$ is `False`, take $q' := q'_{x_i}$. This is correct, because we have $q'_{x_i} \equiv_{p_{x_i}} q_{x_i}$ (by Step 2) and $q'_{x_i} \equiv_{p_{\overline{x_i}}} q'_{x_i}$ (by reflexivity). If $x_i$ is `True`, take $q' := q_{x_i}$. This is correct for similar reasons as above. The last case, $q'_{p_{x_i}} := q'_{x_i}$ and $q'_{p_{\overline{x_i}}} := q_{x_i}$, is similar to the above one. $\square$

*Proof of Remark 7.*
($\Rightarrow$) Assume that $SP_3$ holds for the given action $a$. By contradiction, assume that there exists $q \neq q_a$ such that $q \equiv_{dom(a)} q_a$. The hypothesis of $SP_3$ holds for the above $a$, $q$, and $q_p, q'_p$ chosen to be $q_a, q'_a$, respectively. Then, there must exist $q'$ such that $q \xrightarrow{a} q'$ (and also $q' \equiv_p q'_a$, $\forall p \in dom(a)$). But this is a contradiction, because $q \xrightarrow{a} q'$ would be an $a$-labeled transition different than the supposedly unique transition $q_a \xrightarrow{a} q'_a$.

($\Leftarrow$) Assume now that $\forall q \neq q_a \; \exists p \in dom(a) : q \not\equiv_p q_a$. We must prove that $SP_3$ holds for the given $a$. Let $q, q_p, q'_p \in Q$ such that $q_p \xrightarrow{a} q'_p$ and $q \equiv_p q_p$, for each $p \in dom(a)$. Since $q_a \xrightarrow{a} q'_a$ is the only $a$-labeled transition, we have that $q_p = q_a$ and $q'_p = q'_a$, $\forall p \in dom(a)$. This implies $q \equiv_{dom(a)} q_a$ (because $q \equiv_p q_p$, $\forall p \in dom(a)$). Using the ($\Leftarrow$)-assumption above, we necessarily have that $q = q_a$. Now, it is easy to find a state $q'$ satisfying $q \xrightarrow{a} q'$ and $q' \equiv_p q'_p = q'_a$, $\forall p \in dom(a)$: We simply choose $q' := q'_a$. $\square$

## A.2 Proof of Corollary 9

The problem is in NP for the same reasons given in the proof of Theorem 5. To prove that the problem is NP-hard, we use a reduction from the implementability problem for synchronous products modulo isomorphism. Given a distribution $(\Sigma, Proc, \Delta)$ and a (possibly nondeterministic) transition system $TS$, we construct a new distribution $(\Sigma', Proc, \Delta')$ (over *the same* set of processes $Proc$) and a new transition system $TS'$ such that: There exist $(\equiv_p)_{p \in Proc}$ on the states of $TS$ satisfying $SP_1$-$SP_2$-$SP_3$ in Theorem 4 (i.e., $TS$ is isomorphic to a synchronous product of transition systems over $(\Sigma, Proc, \Delta)$) if and only if there exist $(\equiv'_p)_{p \in Proc}$ on the states of $TS'$ satisfying $\mathrm{SP}_1$ and $\mathrm{SP}_3$ for the new $(\Sigma', Proc, \Delta')$.

For each $(\Sigma, Proc, \Delta)$ and $TS = (Q, \Sigma, \rightarrow, I)$, we build $(\Sigma', Proc, \Delta')$ and $TS' = (Q', \Sigma', \rightarrow', I')$ in the following way:

- We add one new action for each state of $TS$: $\Sigma' := \Sigma \cup \{a_q \mid q \in Q\}$ with the new distribution relation: $\Delta' := \Delta \cup \{(a_q, p) \mid q \in Q,\ p \in Proc\}$.
  The domains generated by $\Delta'$ are $dom'(a) := dom(a)$ for all $a \in \Sigma$ and $dom'(a_q) := Proc$ for all $q \in Q$.
- $Q' := Q \cup \{q_0\}$ where $q_0$ is a new state and $I' := I$.
- $\rightarrow' := \rightarrow \cup \{(q, a_q, q_0) \mid q \in Q\}$. For simplicity, we denote $\rightarrow'$ also by $\rightarrow$.

For the direct implication, let us assume that there exist $(\equiv_p)_{p \in Proc}$ on the states of $TS$ satisfying $\mathrm{SP}_1$-$\mathrm{SP}_2$-$\mathrm{SP}_3$. We extend each $\equiv_p \subseteq Q \times Q$ to $\equiv'_p \subseteq Q' \times Q'$, simply by choosing $\equiv'_p := \equiv_p \cup \{(q_0, q_0)\}$ (i.e., the new state $q_0$ is equivalent only to itself). We prove that $\mathrm{SP}_1$ and $\mathrm{SP}_3$ hold for $(\equiv'_p)_{p \in Proc}$:

$\mathrm{SP}_1$ : Because $\mathrm{SP}_1$ holds for the transitions in $TS$ and $dom'(a) := dom(a)$ for all $a \in \Sigma$, we only have to prove that $\mathrm{SP}_1$ holds for the newly added transitions $q \xrightarrow{a_q} q_0$, but this is trivial given the fact that $dom'(a_q) := Proc$.

$\mathrm{SP}_3$ : We prove that $\mathrm{SP}_3$ holds for each $a \in \Sigma'$ distinguishing two cases:
- For $a \in \Sigma$, $\mathrm{SP}_3$ holds in $TS'$ because $\mathrm{SP}_3$ holds for $TS$ (this is easy).
- For $a \in \Sigma' \setminus \Sigma$, by construction, $dom(a) = Proc$ and there exists $q \in Q$ such that $a = a_q$ and the transition $q \xrightarrow{a} q_0$. Since the transition $q \xrightarrow{a} q_0$ is *the only* one labeled with $a = a_q$ in $TS'$, we prove that $\mathrm{SP}_3$ holds for $a$ in $TS'$, applying Remark 7: We have to prove that for each $r \in Q'$ with $r \neq q$, there exists a process $p \in dom(a) = Proc$ such that $r \not\equiv'_p q$. We have two subcases:
  - For $r \in Q$, we have that $r$ and $q$ belong to $Q$ and $r \neq q$. From the hypothesis, $(\equiv_p)_{p \in Proc}$ satisfies $\mathrm{SP}_2$ In this case, we can apply Remark 8 for $r \neq q$ and we obtain that there exists a process $p \in Proc$ such that $r \not\equiv_p q$, so also $r \not\equiv'_p q$.
  - For $r = q_0$, by construction, $r = q_0 \not\equiv'_p q$ for any $q \in Q$ and $p \in Proc$.

For the reverse implication, assume that there exist $(\equiv'_p)_{p \in Proc}$ on the states of $TS'$ satisfying $\mathrm{SP}_1$ and $\mathrm{SP}_3$. Let us choose $(\equiv_p)_{p \in Proc}$ as the projection of $(\equiv'_p)_{p \in Proc}$ on $Q \times Q$. It is easy to see that $\mathrm{SP}_1$ and $\mathrm{SP}_3$ are satisfied for $TS$ and $(\equiv_p)_{p \in Proc}$ because the same properties hold for $TS'$ and $(\equiv'_p)_{p \in Proc}$. To prove $\mathrm{SP}_2$, we use the equivalent condition given by Remark 8: $\mathrm{SP}_2$ holds for $TS$ and $(\equiv_p)_{p \in Proc}$ iff for each pair of states $q \neq r$ from $Q$, there exists $p \in Proc$ such that $q \not\equiv_p r$. Let $q \neq r$ from $Q$. Since $q \xrightarrow{a_q} q_0$ is the only transition labeled with $a_q$ from $TS'$ and $\mathrm{SP}_3$ holds for $TS'$ and $(\equiv'_p)_{p \in Proc}$, we can apply Remark 7 and we obtain that there exists a process $p \in dom'(a_q) = Proc$ such that $q \not\equiv'_p r$. This implies also that $q \not\equiv_p r$ (because $q, r \in Q$). □

## A.3 Proof of Theorem 12

First, it is easy to see that the problem is in NP: Given a distribution $(\Sigma, \mathit{Proc}, \Delta)$ and a transition system $TS$, a nondeterministic machine can 'guess' a family of equivalences $(\equiv_k)_{k\in \mathit{Proc}}$ and then verify in *polynomial* time (in the size of the distribution and of the transition system) whether the properties $\mathrm{AA}_1$–$\mathrm{AA}_3$ from Theorem 11 are satisfied or not.

For the NP-hardness part, we use a polynomial reduction from the classical SAT problem. Let $\phi$ 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, we construct first a new formula $\phi'$ that satisfy the following properties:

1. each variable appears in at least two different clauses,
2. each clause contains at least two different literals, and
3. $\phi'$ is satisfiable if and only if $\phi$ is satisfiable.

The formula $\phi'$ is constructed from $\phi$ as follows. We start with $\phi'$ being equal to a new clause $c_0 := x_0 \vee \overline{x_0} \vee x_1 \vee x_2 \vee \ldots \vee x_n$, where $x_0$ is a fresh variable. Then we add to $\phi'$ the clauses of $\phi$ that contain at least two different literals. Finally, if $c_j := x_i$ (resp. $c_j := \overline{x_i}$) is a clause of $\phi$, we add to $\phi'$ two new clauses $x_i \vee x_0$ and $x_i \vee \overline{x_0}$ (resp. $\overline{x_i} \vee x_0$ and $\overline{x_i} \vee \overline{x_0}$). It is easy to see that $\phi'$ satisfies the three properties above. For convenience, we denote $\phi'$ also by $\phi$ in the following.

We will construct a distribution $(\Sigma_\phi, \mathit{Proc}_\phi, \Delta_\phi)$ and a (nondeterministic) transition system $TS_\phi = (Q_\phi, \Sigma_\phi, \rightarrow_\phi, I_\phi)$ such that: $\phi$ is satisfiable if and only if $TS_\phi$ is isomorphic to an asynchronous automaton over $\Delta_\phi$. To relieve a bit the notation, we will drop all $\phi$ indices.

First, the set of processes $\mathit{Proc}$ consists of one process for each (positive or negative) literal from each clause:

$$\mathit{Proc} := \{p_{x_i c_j} \mid j \in [0..m],\ x_i \in c_j\} \cup \{p_{\overline{x_i} c_j} \mid j \in [0..m],\ \overline{x_i} \in c_j\}.$$

Then, the set $\Sigma$ of actions and their domains (which determine $\Delta$) consist of:

- one action for each positive literal from each clause: $\{a_{x_i c_j} \mid j \in [0..m],\ x_i \in c_j\}$ with $\mathit{dom}(a_{x_i c_j}) := \mathit{Proc} \setminus \{p_{x_i c_j}\}$.
- one action for each negative literal from each clause: $\{a_{\overline{x_i} c_j} \mid j \in [0..m],\ \overline{x_i} \in c_j\}$ with $\mathit{dom}(a_{\overline{x_i} c_j}) := \mathit{Proc} \setminus \{p_{\overline{x_i} c_j}\}$.
- one action for each variable: $\{a_{x_i} \mid i \in [0..n]\}$ with the domain of $a_{x_i}$ consisting of the processes associated to the literals where $x_i$ appears:
  $\mathit{dom}(a_{x_i}) := \bigcup_{j\,:\,x_i \in c_j} \{p_{x_i c_j}\} \cup \bigcup_{j\,:\,\overline{x_i} \in c_j} \{p_{\overline{x_i} c_j}\}$.
- one action for each clause: $\{a_{c_j} \mid j \in [0..m]\}$ with the domain of $a_{c_j}$ consisting of the processes associated to the literals of $c_j$:
  $\mathit{dom}(a_{c_j}) := \bigcup_{i\,:\,x_i \in c_j} \{p_{x_i c_j}\} \cup \bigcup_{i\,:\,\overline{x_i} \in c_j} \{p_{\overline{x_i} c_j}\}$.

Last, we construct the transition system $TS$: The state space $Q$ consist of:

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

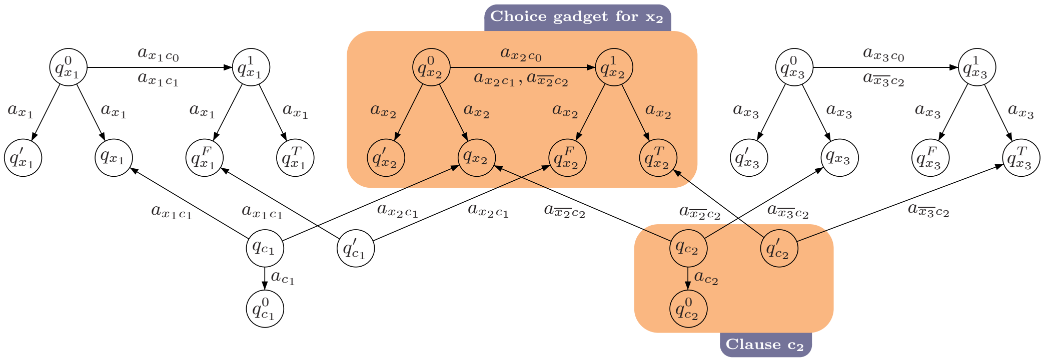



**Fig. 5.** The transition system $TS$ associated to $\phi = (x_1 \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3})$ (without the states and transitions associated to clause $c_0$)

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

- for each $i \in [0..n]$: $q^0_{x_i} \xrightarrow{a_{x_i}} q'_{x_i}$, $q^0_{x_i} \xrightarrow{a_{x_i}} q_{x_i}$, and $q^0_{x_i} \xrightarrow{a} q^1_{x_i}$ for each action $a \in \{a_{x_i c_j} \mid x_i \in c_j\} \cup \{a_{\overline{x_i} c_j} \mid \overline{x_i} \in c_j\}$.
- for each $i \in [0..n]$: $q^1_{x_i} \xrightarrow{a_{x_i}} q^F_{x_i}$, $q^1_{x_i} \xrightarrow{a_{x_i}} q^T_{x_i}$.
- for each $j \in [0..m]$: $q_{c_j} \xrightarrow{a_{x_i c_j}} q_{x_i}$ 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 $j \in [0..m]$: $q'_{c_j} \xrightarrow{a_{x_i c_j}} q^F_{x_i}$ for $x_i \in c_j$ and $q'_{c_j} \xrightarrow{a_{\overline{x_i} c_j}} q^T_{x_i}$ for $\overline{x_i} \in c_j$.

The set of initial states $I$ is chosen such that all states of $Q$ are reachable from $I$. For instance, $I := \{q^0_{x_i}, q^1_{x_i} \mid i \in [0..n]\} \cup \{q_{c_j}, q'_{c_j} \mid j \in [0..m]\}$. (It is easy to modify the construction such that there is only one initial state.)

An example is provided in Fig. 5. We omitted, for legibility, the states $q_{c_0}$, $q'_{c_0}$, and $q^0_{c_0}$ and their associated transitions (those states were introduced for some consistency reasons - see construction of $\phi'$). Also, we did not mark the initial states, because they do not play any rôle here.

The 'choice gadget' is provided by the six states for each variable $x_i$ and their associated transitions. The 'value' of each choice 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: A clause $c_j$ has all the literals evaluated to `False` iff the condition $AA_3$ is violated for $q_1 := q_{c_j}$, $q'_1 := q^0_{c_j}$, $q_2 := q'_{c_j}$.

The above construction is polynomial in the size of the initial formula $\phi$ and we claim that $\phi$ is satisfiable if and only if $TS$ is isomorphic to an asynchronous automaton over $\Delta$ (given by $dom$).

*First Implication.* We first prove the easier part: $\phi$ is not satisfiable implies $TS$ is not isomorphic to an asynchronous automaton over $\Delta$. If $\phi$ is not satisfiable, then for any assignment of the variables $x_0, \ldots, x_n$ there exists a clause that evaluates to `False`. We must show that in this case, there are no $(\equiv_p)_{p \in Proc}$ satisfying all $AA_1$–$AA_3$.

By contradiction, assume that there exist $(\equiv_p)_{p \in Proc}$ satisfying all $AA_1$–$AA_3$.

For each $i \in [0..n]$, from $q^0_{x_i} \xrightarrow{a} q^1_{x_i}$ for all $a \in \{a_{x_i c_j} \mid x_i \in c_j\} \cup \{a_{\overline{x_i} c_j} \mid \overline{x_i} \in c_j\}$, using $\mathrm{AA}_1$, we have that $q^0_{x_i} \equiv_p q^1_{x_i}$ for all $p \in \{p_{x_i c_j} \mid x_i \in c_j\} \cup \{p_{\overline{x_i} c_j} \mid \overline{x_i} \in c_j\}$, and so $q^0_{x_i} \equiv_{dom(a_{x_i})} q^1_{x_i}$. Next, from $q^0_{x_i} \equiv_{dom(a_{x_i})} q^1_{x_i}$ and $q^0_{x_i} \xrightarrow{a_{x_i}} q_{x_i}$, using $\mathrm{AA}_3$, we have that either $q_{x_i} \equiv_{dom(a_{x_i})} q^F_{x_i}$, or $q_{x_i} \equiv_{dom(a_{x_i})} q^T_{x_i}$, but *not both in the same time* (otherwise, by transitivity we have $q^F_{x_i} \equiv_{dom(a_{x_i})} q^T_{x_i}$ and also $q^F_{x_i} \equiv_{Proc \setminus dom(a_{x_i})} q^T_{x_i}$ by $\mathrm{AA}_1$ applied to $q^F_{x_i} \xleftarrow{a_{x_i}} q^1_{x_i} \xrightarrow{a_{x_i}} q^T_{x_i}$, so $q^F_{x_i} \equiv_{Proc} q^T_{x_i}$ and this would contradict $\mathrm{AA}_2$). Let us choose an assignment of the variables given by the equivalences in the following way. For each $i \in [0..n]$:

$$x_i \text{ is evaluated to } \texttt{False} \text{ if and only if } q_{x_i} \equiv_{dom(a_{x_i})} q^F_{x_i}.$$

Because $\phi$ 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 $q_{x_i} \equiv_{dom(a_{x_i})} q^F_{x_i}$ and this implies $q_{x_i} \equiv_{p_{x_i c_k}} q^F_{x_i}$ (because $p_{x_i c_k} \in dom(a_{x_i})$). In addition, we have $q_{c_k} \xrightarrow{a_{x_i c_k}} q_{x_i}$ and $q'_{c_k} \xrightarrow{a_{x_i c_k}} q^F_{x_i}$ (see the construction of *TS*) and, using $\mathrm{AA}_1$, we deduce that $q_{c_k} \equiv_{p_{x_i c_k}} q_{x_i}$ and $q'_{c_k} \equiv_{p_{x_i c_k}} q^F_{x_i}$. By transitivity of $\equiv_{p_{x_i c_k}}$, we obtain that $q_{c_k} \equiv_{p_{x_i c_k}} q'_{c_k}$. A similar argument for the negative literals $\overline{x_i}$ in $c_k$ (if any), proves that $q_{c_k} \equiv_{p_{\overline{x_i} c_k}} q'_{c_k}$.

Summing up, $q_{c_k}$ and $q'_{c_k}$ are equivalent on all the processes associated to the literals in $c_k$, and so, by the definition of $dom(a_{c_k})$, we obtain that $q_{c_k} \equiv_{dom(a_{c_k})} q'_{c_k}$. But this contradicts $\mathrm{AA}_3$ 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.* We move now to the second part of the proof, assuming that $\phi$ is satisfiable. Then, there exists an assignment to the variables $x_0, \ldots, x_n$ such that each clause is `True`. We prove that *TS* is isomorphic to an asynchronous automaton over $\Delta$. For that, we construct a family of equivalences $(\equiv_p)_{p \in Proc}$ in the following way:

**Step 0** For each $p \in Proc$, initialize the binary relation $\equiv_p \subseteq Q \times Q$ to $\emptyset$.
**Step 1** For each $q \xrightarrow{a} q'$, choose $q \equiv_p q'$ for each $p \in Proc \setminus dom(a)$.
**Step 2** For each $i \in [0..n]$, if variable $x_i$ is evaluated to `False`, then choose $q_{x_i} \equiv_{dom(a_{x_i})} q^F_{x_i}$ and $q'_{x_i} \equiv_{dom(a_{x_i})} q^T_{x_i}$, otherwise choose $q_{x_i} \equiv_{dom(a_{x_i})} q^T_{x_i}$ and $q'_{x_i} \equiv_{dom(a_{x_i})} q^F_{x_i}$.
**Step 3** For each $p \in Proc$, close $\equiv_p$ under reflexivity, symmetry, and transitivity.

We prove that the chosen $(\equiv_p)_{p \in Proc}$ satisfy all $\mathrm{AA}_1$–$\mathrm{AA}_3$ and this implies, by Theorem 11, that *TS* is isomorphic to an asynchronous automaton over $\Delta$.

Let us start making some remarks:

- Step 1 directly imposes $\mathrm{AA}_1$ to be satisfied.
- Step 2 implements the choice gadget for the variables: For each $i \in [0..n]$, from $q^0_{x_i} \xrightarrow{a} q^1_{x_i}$ for all $a \in \{a_{x_i c_j} \mid x_i \in c_j\} \cup \{a_{\overline{x_i} c_j} \mid \overline{x_i} \in c_j\}$, using Step 1 (i.e., $\mathrm{AA}_1$), we have that $q^0_{x_i} \equiv_{dom(a_{x_i})} q^1_{x_i}$. It is not difficult to check that

**Table 6.** The equivalence classes constructed in the 2nd part of the proof of Th. 12

| | $x_i = \texttt{False}$ | $x_i = \texttt{True}$ |
|---|---|---|
| $\equiv_{p_{x_i c_j}}$ (for $x_i \in c_j$) | $\{q^0_{x_i}, q^1_{x_i}\}$, $\{q'_{x_i}, q^T_{x_i}\}$, $\{q_{x_i}, q^F_{x_i}, q_{c_j}, q'_{c_j}\}$, $\{q^0_{c_j}\}$, for each $i' \neq i$, $\{q^0_{x_{i'}}, q'_{x_{i'}}, q_{x_{i'}}\}$, $\{q^1_{x_{i'}}, q^F_{x_{i'}}, q^T_{x_{i'}}\}$, and for each $j' \neq j$, $\{q_{c_{j'}}, q^0_{c_{j'}}\}$, $\{q'_{c_{j'}}\}$ | $\{q^0_{x_i}, q^1_{x_i}\}$, $\{q'_{x_i}, q^F_{x_i}, q'_{c_j}\}$, $\{q_{x_i}, q^T_{x_i}, q_{c_j}\}$, $\{q^0_{c_j}\}$ for each $i' \neq i$, $\{q^0_{x_{i'}}, q'_{x_{i'}}, q_{x_{i'}}\}$, $\{q^1_{x_{i'}}, q^F_{x_{i'}}, q^T_{x_{i'}}\}$, and for each $j' \neq j$, $\{q_{c_{j'}}, q^0_{c_{j'}}\}$, $\{q'_{c_{j'}}\}$ |
| $\equiv_{p_{\overline{x_i} c_j}}$ (for $\overline{x_i} \in c_j$) | $\{q^0_{x_i}, q^1_{x_i}\}$, $\{q'_{x_i}, q^T_{x_i}, q'_{c_j}\}$, $\{q_{x_i}, q^F_{x_i}, q_{c_j}\}$, $\{q^0_{c_j}\}$ for each $i' \neq i$, $\{q^0_{x_{i'}}, q'_{x_{i'}}, q_{x_{i'}}\}$, $\{q^1_{x_{i'}}, q^F_{x_{i'}}, q^T_{x_{i'}}\}$, and for each $j' \neq j$, $\{q_{c_{j'}}, q^0_{c_{j'}}\}$, $\{q'_{c_{j'}}\}$ | $\{q^0_{x_i}, q^1_{x_i}\}$, $\{q'_{x_i}, q^F_{x_i}\}$, $\{q_{x_i}, q^T_{x_i}, q_{c_j}, q'_{c_j}\}$, $\{q^0_{c_j}\}$, for each $i' \neq i$, $\{q^0_{x_{i'}}, q'_{x_{i'}}, q_{x_{i'}}\}$, $\{q^1_{x_{i'}}, q^F_{x_{i'}}, q^T_{x_{i'}}\}$, and for each $j' \neq j$, $\{q_{c_{j'}}, q^0_{c_{j'}}\}$, $\{q'_{c_{j'}}\}$ |

AA$_3$ holds for the states $q^0_{x_i}$, $q^1_{x_i}$ and each of the $a_{x_i}$-labeled edges coming out of them. For example, let us take $q^0_{x_i}$, $q^1_{x_i}$, for which we have $q^0_{x_i} \xrightarrow{a_{x_i}} q_{x_i}$ and $q^0_{x_i} \equiv_{dom(a_{x_i})} q^1_{x_i}$. The first part of AA$_3$ is satisfied, so there must exist a state $q'$ such that $q^1_{x_i} \xrightarrow{a_{x_i}} q'$ and $q_{x_i} \equiv_{dom(a_{x_i})} q'$. If $x_i$ is `False`, we take $q' := q^F_{x_i}$, otherwise $q' := q^T_{x_i}$. Loosely speaking, we can tell the value of $x_i$ checking which of the states $q^F_{x_i}$ and $q^T_{x_i}$ is equivalent on $dom(a_{x_i})$ with $q_{x_i}$.
- Step 3 ensures that $(\equiv_p)_{p \in Proc}$ are equivalences.
- Moreover, Step 3 transmits further the information from the variables to the literals in the clauses. In the example in Fig. 5, $x_2$ appears in $c_1$. If $x_2$ is evaluated to `False`, then we know that $q_{x_2} \equiv_{dom(a_{x_2})} q^F_{x_2}$ and this implies $q_{x_2} \equiv_{p_{x_2 c_1}} q^F_{x_2}$. We also have $q_{c_1} \xrightarrow{a_{x_2 c_1}} q_{x_2}$ and $q'_{c_1} \xrightarrow{a_{x_2 c_1}} q^F_{x_2}$ and, using Step 1 (i.e., AA$_1$), we deduce that $q_{c_1} \equiv_{p_{x_2 c_1}} q_{x_2}$ and $q'_{c_1} \equiv_{p_{x_2 c_1}} q^F_{x_2}$. By Step 3 (i.e., transitivity of $\equiv_{p_{x_2 c_1}}$), we obtain that $q_{c_1} \equiv_{p_{x_2 c_1}} q'_{c_1}$. Therefore, we have indeed that $x_2 = \texttt{False}$ implies $q_{c_1} \equiv_{p_{x_2 c_1}} q'_{c_1}$ (at the level of equivalences).

Table 6 describes the equivalence classes of our equivalences generated above. Each cell gives the partition of the state space $Q$ into the equivalence classes for each type of process (rows) depending on the value of the associated variable (columns). Each equivalence class is given as a set in curly brackets. It is tedious, but not hard to check the correctness of Table 6. Based on it, we prove that the chosen $(\equiv_p)_{p \in Proc}$ satisfy AA$_3$, AA$_2$, and AA$_1$ (in this order):

*Remark 30.* A *sufficient* condition for AA$_3$ to hold for $q_1 \neq q_2$ and $q_1 \xrightarrow{a} q'_1$ is that there exists $p \in dom(a)$ such that $q_1 \not\equiv_p q_2$.

First, Table 7 solves the cases for AA$_3$. The first column picks a value for $q_1$, the second one gives the transition $q_1 \xrightarrow{a} q'_1$ we are considering, while the third column gives a range to $q_2$. In the last column we either give a state $q'_2$ as in the formulation of AA$_3$ or we give a process $p \in dom(a)$ such that $q_1 \not\equiv_p q_2$ as in Remark 30. The correctness of the solutions provided is verified using Table 6.

*Remark 31.* AA$_2$ holds iff for each $q_1 \neq q_2$, there exists $p \in Proc$ with $q_1 \not\equiv_p q_2$.

**Table 7.** Details for the satisfaction of the $AA_3$ property

| $q_1$ | $q_1 \xrightarrow{a} q_1'$ | $q_2$ | Why $AA_3$ holds (either a state $q_2'$ or a process $p \in dom(a)$ are given) |
|---|---|---|---|
| $q^0_{x_i}$ | $q^0_{x_i} \xrightarrow{a_{x_i}} q'_{x_i}$ | $\{q^1_{x_i}\}$ | We have $q^0_{x_i} \equiv_{dom(a_{x_i})} q^1_{x_i}$ (because of Step 1; see also Table 6). If $x_i$ is `True`, choose $q_2' := q^F_{x_i}$, otherwise choose $q_2' := q^T_{x_i}$. |
| | | $Q \setminus \{q^0_{x_i}, q^1_{x_i}\}$ | $p := p_{x_i c_0}$. |
| | $q^0_{x_i} \xrightarrow{a_{x_i}} q_{x_i}$ | Similar to the case $q^0_{x_i} \xrightarrow{a_{x_i}} q'_{x_i}$ above. | |
| | $q^0_{x_i} \xrightarrow{a_{x_i c_j}} q^1_{x_i}$ (for $x_i \in c_j$) | $\{q^1_{x_i}\}$ | $p := p_{x_{i'} c_0}$, where $i' \neq i$ (by constr. $x_{i'} \in c_0$). |
| | | $Q \setminus \{q^0_{x_i}, q^1_{x_i}\}$ | $p$ can be any process from $dom(a_{x_i}) \setminus \{p_{x_i c_j}\}$. Such $p$ exists, because $\mathrm{card}(dom(a_{x_i})) \geq 2$ (remember that we forced each variable of $\phi$ to appear in at least two clauses). |
| | $q^0_{x_i} \xrightarrow{a_{\overline{x_i} c_j}} q^1_{x_i}$ (for $\overline{x_i} \in c_j$) | Similar to the case $q^0_{x_i} \xrightarrow{a_{x_i c_j}} q^1_{x_i}$ above. | |
| $q^1_{x_i}$ | Similar to the case $q^0_{x_i}$ above. | | |
| $q_{c_j}$ | $q_{c_j} \xrightarrow{a_{x_i c_j}} q_{x_i}$ (for $x_i \in c_j$) | $\{q^0_{c_j}\}$ | We choose $p$ to be the process associated to a literal $\ell$ of $c_j$ different[a] from $x_i$. So, $p := p_{\ell c_j}$ for $\ell \in c_j$ and $\ell \neq x_i$. |
| | | $Q \setminus \{q_{c_j}, q^0_{c_j}\}$ | $p := p_{\ell c_{j'}}$, for $c_{j'}$ different[b] from $c_j$ and $\ell \in c_{j'}$ |
| | $q_{c_j} \xrightarrow{a_{\overline{x_i} c_j}} q_{x_i}$ (for $\overline{x_i} \in c_j$) | Similar to the case $q_{c_j} \xrightarrow{a_{x_i c_j}} q_{x_i}$ above. | |
| | $q_{c_j} \xrightarrow{a_{c_j}} q^0_{c_j}$ | Since $c_j$ evaluates to `True`, there exists a literal $\ell$ of $c_j$ evaluated to `True`. Assume $\ell = x_i$, such that $x_i \in c_j$ and $x_i =$ `True` (a similar analysis is made if $\ell$ is negative). Then: | |
| | | $\{q_{x_i}, q^T_{x_i}\}$ | $p := p_{\ell c_j}$ for $\ell \in c_j$ and $\ell \neq x_i$ [a]. |
| | | $Q \setminus \{q_{c_j}, q_{x_i}, q^T_{x_i}\}$ | $p := p_{x_i c_j}$. |
| $q'_{c_j}$ | Similar to the case $q_{c_j}$ above. | | |

[a]Such different literal $\ell$ exists because we forced each clause to contain at least two different literals.

[b]Our formula $\phi$ contains at least two clauses.

**Table 8.** Details for the $AA_2$ property (only cases not solved already in Table 7)

| $q_1$ | $q_2$ | Why $AA_2$ holds (a process $p \in Proc$ is given) |
|---|---|---|
| $q'_{x_i}$ | $\{q_{x_i}\}$ | $p := p_{x_i c_0}$ (by construction $x_i \in c_0$). |
| | $Q' \setminus \{q'_{x_i}, q_{x_i}\}$ | $p := p_{x_{i'} c_0}$, where $i' \neq i$ (by construction $x_{i'} \in c_0$). |
| $q_{x_i}$ | $Q' \setminus \{q_{x_i}, q'_{x_i}\}$ | $p := p_{x_{i'} c_0}$, where $i' \neq i$ (by construction $x_{i'} \in c_0$). |
| $q^F_{x_i}$ | $\{q^T_{x_i}\}$ | $p := p_{x_i c_0}$ (by construction $x_i \in c_0$). |
| | $Q' \setminus \{q^F_{x_i}, q^T_{x_i}\}$ | $p := p_{x_{i'} c_0}$, where $i' \neq i$ (by construction $x_{i'} \in c_0$). |
| $q^T_{x_i}$ | $Q' \setminus \{q^T_{x_i}, q^F_{x_i}\}$ | $p := p_{x_{i'} c_0}$, where $i' \neq i$ (by construction $x_{i'} \in c_0$). |
| $q^0_{c_j}$ | $Q' \setminus \{q^0_{c_j}\}$ | $p := p_{\ell c_j}$, where $\ell$ is a literal of $c_j$. |

Table 8 presents only the cases for which Table 7 did not give a process to 'distinguish' two different states $q_1$ (column 1) and $q_2$ (in the range given in column 2), i.e., we give $p \in \mathit{Proc}$ such that $q_1 \not\equiv_p q_2$. More precisely, we only have to consider pairs of states from the subset:

$$Q' := \{q'_{x_i}, q_{x_i}, q^F_{x_i}, q^T_{x_i} \mid i \in [0..n]\} \cup \{q^0_{c_j} \mid j \in [0..m]\}.$$

(Note that the elements of $Q'$ are those state with no outgoing edges.)

Condition $\mathrm{AA}_1$ is fulfilled by construction (Step 1). $\square$

### A.4 Details for the proofs in Section 4.2

*Proof of Proposition 22*

The direct implication is easy. For the reverse, let $T$ be a prefix-closed regular trace language. From [Zie89, Th. 4.8], there exists a safe asynchronous automaton $\mathcal{AA}$ with a set of global initial states $I$ and a set of global accepting states $F$ such that $L(\mathcal{AA}, F) = T$, where $L(\mathcal{AA}, F) := \{w \in \Sigma^* \mid \exists q^{in} \in I,\ q \in F,\ q^{in} \xrightarrow{w} q\}$.

We show that $L(\mathcal{AA}, F) = L(\mathcal{AA})$, where $L(\mathcal{AA})$ is our definition of language containing all the runs starting in an initial state of $I$.

Since inclusion $L(\mathcal{AA}, F) \subseteq L(\mathcal{AA})$ is obvious, we only prove $L(\mathcal{AA}) \subseteq L(\mathcal{AA}, F)$ using the hypothesis that $T = L(\mathcal{AA}, F)$ is prefix-closed. Let $w \in L(\mathcal{AA})$. Then, there exists the run $q^{in} \xrightarrow{w} q$ with $q^{in} \in I$ in $\mathcal{AA}$. By construction, $\mathcal{AA}$ is a *safe* asynchronous automaton, so any run of $\mathcal{AA}$ from an initial state can be extended to an accepting run. In particular, $w$ can be extended to a run $q^{in} \xrightarrow{w'} q'$ with $q' \in F$. Hence $w$ is a prefix of $w' \in L(\mathcal{AA}, F)$ and since $L(\mathcal{AA}, F)$ is prefix-closed, we conclude that $w \in L(\mathcal{AA}, F)$. $\square$

*Proof of Theorem 23*

The 'totality problem for regular languages' is defined as: Given a nondeterministic finite automata $\mathcal{A}$ over the alphabet $\Sigma$, is the language accepted by $\mathcal{A}$ (given by a set of accepting states) equal to $\Sigma^*$? It is a classical result that this problem is PSPACE-complete [GJ79].

We reduce the above totality problem to the implementability problem for asynchronous automata modulo language equivalence. For each nondeterministic finite automaton $\mathcal{A} = (Q, \Sigma, \rightarrow, \{q^{in}\}, \{q^{acc}\})$ where $Q$ is the set of states, $\Sigma$ the alphabet, $\rightarrow \subseteq Q \times \Sigma \times Q$ the transition relation, $q^{in}$ the initial state, and $q^{acc}$ the accepting state (we can suppose w.l.o.g. that $\mathcal{A}$ has only one initial, respectively only one accepting state), we build a distribution $(\Sigma', \mathit{Proc}, \Delta)$ and a transition system $\mathit{TS}$ over $\Sigma'$ such that $L(\mathcal{A}, \{q^{acc}\}) = \Sigma^*$ iff $L(\mathit{TS})$ is a trace language (this is enough according to Proposition 22).

We choose $\Sigma' := \Sigma \cup \{a, b, c\}$ with $a, b, c \notin \Sigma$, $\mathit{Proc} := \{1, 2\}$, and $\Delta$ such that $\Sigma_{loc}(1) := \Sigma \cup \{a, b\}$ and $\Sigma_{loc}(2) := \Sigma \cup \{a, c\}$. We can see that $b \| c$ is the only independence generated by $\Delta$. Then, we choose the transition system $\mathit{TS} := (Q', \Sigma', \rightarrow', \{q_0\})$ as in Fig. 3 (left). More precisely, we have

- $Q' := Q \cup \{q_0, q_1, q_2, q_3, q'_2, q'_3\}$ and

$$
\begin{aligned}
- \to' := \{q_0 \xrightarrow{a} q_1\} \cup \{q_1 \xrightarrow{\alpha} q_1 \mid \alpha \in \Sigma\} \cup \{q_1 \xrightarrow{b} q_2 \xrightarrow{c} q_3\} \\
\cup \{q_0 \xrightarrow{a} q^{in}\} \cup (\to) \cup \{q^{acc} \xrightarrow{c} q_2' \xrightarrow{b} q_3'\}.
\end{aligned}
$$

For a language $L$, we denote by $\mathit{Prefix}(L)$ the language which contains all the prefixes of the words of $L$. Then, it is easy to see that

$$L(\mathit{TS}) = \mathit{Prefix}(a\Sigma^* bc + a\, L(\mathcal{A}, \{q^{acc}\})\, cb).$$

If we assume that $L(\mathcal{A}, \{q^{acc}\}) = \Sigma^*$, then $L(\mathit{TS}) = \mathit{Prefix}(a\Sigma^*(bc + cb))$, which is obviously a trace language (the only independence is $b \| c$). On the other hand, if $L(\mathcal{A}, \{q^{acc}\}) \neq \Sigma^*$, there exists $w \in \Sigma^* \setminus L(\mathcal{A}, \{q^{acc}\})$, which implies that $L(\mathit{TS})$ is not a trace language because $awbc \in L(\mathit{TS})$ and $b \| c$, but $awcb \notin L(\mathit{TS})$.

We mention that our proof is reminiscent of the PSPACE-hardness proof for checking the trace closure of the language of a nondeterministic I-diamond Büchi automaton [Mus94, Th. 7.2.3]. An alternative proof to our result can be obtained modifying the PSPACE-hardness proof of checking the trace closure of the language of a nondeterministic automaton [PWW98, Th. 11]. The difference is that the language of a transition system is always prefix-closed, so the constructions must accommodate this detail (and this is not immediate). $\square$

*Proof of Proposition 24*
First, given an *acyclic* transition system $\mathit{TS}$ together with a distribution $\Delta$, checking that $L(\mathit{TS})$ is *not* trace-closed is in NP. A machine can guess a pair of independent actions $a \| b$ and two strings $w, w'$ and check in polynomial time whether $wabw' \in L(\mathit{TS})$ and $wbaw' \notin L(\mathit{TS})$ (notice that, since $\mathit{TS}$ is acyclic, the length of $w$ and $w'$ is bounded by the size of $\mathit{TS}$).

For the coNP-hardness part, we use a reduction from the NP-complete problem of 'language inequivalence for acyclic finite nondeterministic automata' (see [GJ79]): Given two *acyclic* nondeterministic finite automata $\mathcal{A}_1$ and $\mathcal{A}_2$ over the same alphabet $\Sigma$, do they accept different languages?

Given $\mathcal{A}_1 = (Q_1, \Sigma, \to_1, \{q_1^{in}\}, \{q_1^{acc}\})$ and $\mathcal{A}_2 = (Q_2, \Sigma, \to_2, \{q_2^{in}\}, \{q_2^{acc}\})$ two acyclic nondeterministic finite automata assumed w.l.o.g. to have only one initial, respectively one accepting state, we construct a transition system $\mathit{TS}$ as in Fig. 3 (right) and a distribution such that $b \| c$ (cf. the proof of Th. 23). We have then:

$$L(\mathit{TS}) = \mathit{Prefix}(a\, L(\mathcal{A}_1, \{q_1^{acc}\}) bc + a\, L(\mathcal{A}_2, \{q_2^{acc}\})\, cb),$$

which easily implies that $L(TS)$ is a not a trace language if and only if the languages $L(\mathcal{A}_1, \{q_1^{acc}\})$ and $L(\mathcal{A}_2, \{q_2^{acc}\})$ are different. $\square$

*Proof of Theorem 25*
To prove the PSPACE-hardness of the implementability problem for synchronous products of transition systems modulo language equivalence, we use the construction of $\mathit{TS}$ and the distribution $(\Sigma', \mathit{Proc}, \Delta)$ from the proof of Th. 23 (see

Fig. 3). We show that $L(\mathcal{A}, \{q^{acc}\}) = \Sigma^*$ iff $L(TS)$ is the language of a synchronous product over $\Delta$: If $L(\mathcal{A}, \{q^{acc}\}) = \Sigma^*$, then $L(TS) = \mathit{Prefix}(a\Sigma^*(bc + cb))$ and we can easily construct a synchronous product accepting $L(TS)$ (we choose the two local components accepting the projections $\mathit{Prefix}(a\Sigma^*b)$, respectively $\mathit{Prefix}(a\Sigma^*c)$). If $L(\mathcal{A}, \{q^{acc}\}) \neq \Sigma^*$, then $L(TS)$ is not a trace language and therefore cannot be the language of a synchronous product (because the language of a synchronous product of transition systems is always a trace language).

For acyclic specifications, it was proved in [Dub86] that the class of *finite* languages accepted by synchronous products is equal to the class of *finite* languages accepted by asynchronous automata, viz. the class of finite prefix-closed trace languages. Therefore the complexity of the implementability problem for acyclic specifications is settled by the coNP-completeness result of Proposition 24.

As a remark, the above constructions cannot be used for the proof of Proposition 21 treating the case of *deterministic* specifications. □

*Proof of Proposition 26*
We use a reduction from the emptiness problem for Turing machines:

Given an arbitrary Turing machine $M = (Q, \Gamma, \Gamma_0, \delta, q_0, B, F)$, we construct a context-free grammar $G$ such that $L(G)$ is a trace language iff $L(M) = \emptyset$.

By definition, a *valid computation* of $M$ is a string $\#w_1\#w_2^R\#w_3\#w_4^R\#...$ over the alphabet $Q \cup \Gamma \cup \{\#\}$ (with $Q$ and $\Gamma$ disjoint and $\#$ belonging to none of them) such that:

1. each $w_i$ is an instantaneous description (ID) of $M$, that is, a string in $\Gamma^*Q\Gamma^*$ not ending with $B$ (where $B$ is the blank symbol),
2. $w_1$ is an initial ID, one of the form $q_0x$ for $x \in \Gamma_0^*$,
3. $w_n$ is a final ID, that is, one in $\Gamma^*F\Gamma^*$, and
4. $w_i \vdash_M w_{i+1}$ for $1 \leq i < n$ ($w_i$ and $w_{i+1}$ are consecutive IDs).

Let $\Sigma := Q \cup \Gamma \cup \{\#\}$. We choose a context-free grammar $G$ over $\Sigma$ able to compute exactly *all* the invalid computations of $M$. Such a grammar $G$ exists according to the construction of [HU79, Lemma 8.7]. We have then that:

- if $L(M) = \emptyset$, then $L(G) = \Sigma^*$
- if $L(M) \neq \emptyset$, then $L(G) = \Sigma^* \setminus \{$the set of all valid computations of $M\}$.

The trick now is to choose a distribution over $\Sigma$ such that $\#$ and $q_0$ are independent. We show that $L(M) = \emptyset$ iff $L(G)$ is a trace language. If $L(M) = \emptyset$, it is obvious that $L(G) = \Sigma^*$ is a trace language. If $L(M) \neq \emptyset$, then there is a valid computation of $M$, say $w$, which will not belong to $L(G)$. On one hand, by definition, $w$ has the form $\#q_0w'$. On the other hand, $q_0\#w'$ is an invalid computation (any valid computation starts with $\#$) and therefore belongs to $L(G)$. So, we have that $q_0$ and $\#$ are independent and $q_0\#w' \in L(G)$, but $\#q_0w' \notin L(G)$, which means that $L(G)$ is not a trace language. □

### A.5 Details for Section 5

**Definition 32.** A (strong) *bisimulation* between a pair of transition systems $TS_1 = (Q_1, \Sigma, \rightarrow_1, I_1)$ and $TS_2 = (Q_2, \Sigma, \rightarrow_2, I_2)$ is a binary relation $\sim \subseteq Q_1 \times Q_2$ (for which we use the infix notation) such that:

- For each $q_1^{in} \in I_1$, there exists $q_2^{in} \in I_2$ such that $q_1^{in} \sim q_2^{in}$.
- For each $q_2^{in} \in I_2$, there exists $q_1^{in} \in I_1$ such that $q_1^{in} \sim q_2^{in}$.
- If $q_1 \sim q_2$ and $q_1 \xrightarrow{a}_1 q_1'$, there exists $q_2'$ such that $q_2 \xrightarrow{a}_2 q_2'$ and $q_1' \sim q_2'$.
- If $q_1 \sim q_2$ and $q_2 \xrightarrow{a}_2 q_2'$, there exists $q_1'$ such that $q_1 \xrightarrow{a}_1 q_1'$ and $q_1' \sim q_2'$.

Having defined the bisimulation, for a transition system $TS$ one can define $\sim_{TS}$ as the *largest* bisimulation between $TS$ and itself. Constructing the largest bisimulation can be done in polynomial time. Since $\sim_{TS}$ defines an equivalence relation on $Q$, we can construct (in the usual way) the quotient transition system $TS/_{\sim_{TS}}$.

**Synchronous Products of Transition Systems** The synthesis problem modulo bisimulation in solved in [Muk02] for *deterministic* synchronous products by:

**Theorem 33.** *[CMT99,Muk02] Let $(\Sigma, Proc, \Delta)$ be a distribution and $TS$ a transition system over $\Sigma$. Then, $TS$ is bisimilar to a* deterministic *synchronous product of transition systems over $\Delta$ if and only if the bisimulation quotient $TS/_{\sim_{TS}}$ is deterministic and $TS$ is language equivalent to a* deterministic *synchronous product over $\Delta$.*

For a given transition system, checking the language equivalence with a *deterministic* synchronous product uses the same algorithm as for checking language equivalence with a synchronous product with only one initial state, i.e., the decision algorithm for Problem 16 (this follows from the observations that a deterministic synchronous product has only one initial state and that the local components $(TS_p)_{p \in Proc}$ can be determinized to build up a deterministic synchronous product).

**Theorem 34.** *The implementation problem for* deterministic *synchronous products modulo bisimulation is PSPACE-complete.*

*Proof.* The problem is in PSPACE: Checking the language equivalence of $TS$ with a deterministic synchronous product is in PSPACE (the decision algorithm for Problem 16 is in PSPACE) and checking that $TS/_{\sim_{TS}}$ is deterministic takes polynomial time (constructing the largest bisimulation takes polynomial time).

The PSPACE-hardness proof is the similar to the proof of Proposition 21 observing that $TS/_{\sim_{TS}}$ is deterministic, whenever $TS$ is deterministic. $\square$

**Asynchronous Automata** Given a distribution $(\Sigma, Proc, \Delta)$, we say a language $L \subseteq \Sigma^*$ is *forward-independence closed* iff $\forall w \in \Sigma^*,\ a, b \in \Sigma : wa, wb \in L \wedge a \| b \Rightarrow wab \in L$. It is easy to see that the languages of *deterministic* asynchronous automata are forward-closed. Moreover, Zielonka's construction [Zie87] can be slightly modified to obtain the following characterization:

**Proposition 35.** *[Muk02,ŞEM03] A language $L \subseteq \Sigma^*$ is accepted by a* deterministic *asynchronous automaton if and only if $L$ is a prefix-closed regular forward-closed trace language.* [10]

The synthesis problem modulo bisimulation is solved in [Muk02] for *deterministic* asynchronous automata by:

**Theorem 36.** *[Muk02] Let $(\Sigma, Proc, \Delta)$ be a distribution and $TS$ a transition system over $\Sigma$. Then, $TS$ is bisimilar to a* deterministic *asynchronous automaton over $\Delta$ if and only if the bisimulation quotient $TS/_{\sim_{TS}}$ is deterministic and $TS$ is language equivalent to a* deterministic *asynchronous automaton over $\Delta$.*

**Theorem 37.** *The implementation problem for* deterministic *asynchronous automata modulo bisimulation can be decided in polynomial time.*

*Proof.* A polynomial algorithm to decide this problem works as follows:

**Input**: a distribution $(\Sigma, Proc, \Delta)$ and a transition system $TS$
**Output**: Is $TS$ bisimilar to a deterministic asynchronous automaton?

Construct $TS/_{\sim_{TS}}$
**if** $TS/_{\sim_{TS}}$ is deterministic **then**
  **if** $L(TS/_{\sim_{TS}})$ is a forward-closed trace language **then**
    output 'yes'
  **else** output 'no'
**else** output 'no'.

The algorithm is polynomial because the construction of $TS/_{\sim_{TS}}$ is polynomial, the first test is obviously polynomial, and the second test is polynomial for transition systems that are deterministic (which is the case for $TS/_{\sim_{TS}}$, because of the if-nesting). The algorithm is correct according to Theorem 36, Proposition 35, and the language equality $L(TS/_{\sim_{TS}}) = L(TS)$. □

# B Implementation

## B.1 Code of the implementation described in Section 6

```
% Input:    a distribution (as a domain) and
%           a (possible nondeterministic) transition system TS
```

[10] This result together with Proposition 22 show that the nondeterministic asynchronous automata are strictly more expressive than their deterministic counterparts (as opposed to [Zie87] where they have same expressivity due to the flexibility of choosing a set of global accepting states).

```
% Question: is TS reachable, deadlock-free, and isomorphic
%           to an asynchronous automaton?
% Strategy: guess a set of equivalences satisfying AA1-AA2-AA3

%===specify the input as a database of facts
%...the translation of the distribution from Fig. 4
dom(a,1). dom(b,2).
%...the translation of the nondeterministic TS from Fig. 4
initialstate(1). initialstate(2).
trans(1,a,3). trans(2,b,3).
trans(3,c,3). dom(c,1). dom(c,2). % added only for deadlock freedom
%...fix an upper bound on the number of local states
const max_local_state = 3.


%=== code for the implementation of Problem 29 (by Th. 11) begins here

%...derive the actions and processes of the distributed alphabet
action(A) :- dom(A,K).
process(K):- dom(A,K).
%...derive the states of TS
state(Q1) :- trans(Q1,A,Q2).
state(Q2) :- trans(Q1,A,Q2).

%===reachability (least fixpoint procedure)
reach(Q)  :- initialstate(Q).
reach(Q2) :- reach(Q1), trans(Q1,A,Q2), neq(Q1,Q2).
%...rule out solutions which contain unreachable states
:- state(Q), not reach(Q).

%===deadlock-freedom
live(Q1) :- trans(Q1,A,Q2).
%...rule out solutions which contain reachable deadlocks
:- state(Q), reach(Q), not live(Q).

%===choose for each state a local representative from a set of local states.
%   Two states are equivalent iff they have the same local representative.
%...derive the set of local states
local_state(1..max_local_state).
%...guess exactly one k-representative lq for the each state q
%...(choice gadget)
1 {local_repr(K,Q,LQ) : local_state(LQ)} 1 :- process(K), state(Q).
%...(q1 equiv_k q2) iff they have the same local representative
equal_local(K,Q1,Q2) :- local_repr(K,Q1,LQ), local_repr(K,Q2,LQ),
   process(K), state(Q1), state(Q2), local_state(LQ).
%...equivalence of q1 and q2 on dom(a)
equiv_dom(A,Q1,Q2) :- equal_local(K,Q1,Q2):dom(A,K),
   action(A), state(Q1), state(Q2).

%===AA1: q1-a->q2 and k not in dom(a) => local_state_k(q1)=local_state_k(q2)
```

**Table 9.** Experimental results for SubautIDFD and SubautAsync (Problems 28,29)

| Problem | Input | | | | SubIDFD -synasync | SubIDFD -logic | SubAsync -determ. | SubAsync -nondeterm. |
|---|---|---|---|---|---|---|---|---|
| | $\|Q\|$ | $\|\rightarrow\|$ | $\|\Sigma\|$ | $\|P\|$ | | | | |
| Mutex(2) | 14 | 22 | 6 | 4 | 0.01 | 0.01 | 0.47 | 0.24 |
| Mutex(3) | 107 | 210 | 9 | 6 | 0.01 | 0.10 | 349.52 | – |
| Mutex(4) | 1340 | 3040 | 12 | 8 | 2.21 | 20.87 | – | – |
| Phil(2) | 7 | 10 | 6 | 4 | 0.01 | 0.01 | 0.09 | 0.05 |
| Phil(3) | 27 | 63 | 9 | 6 | x | 0.03 | 6.39 | 4.69 |
| Phil(4) | 81 | 252 | 12 | 8 | x | 0.27 | – | – |

```
%...rule out solutions not satisfying AA1
:- trans(Q1,A,Q2), not dom(A,K),
   process(K), local_state(LQ1), local_state(LQ2),
   local_repr(K,Q1,LQ1), local_repr(K,Q2,LQ2), neq(LQ1,LQ2).

%===AA2: (for all k in Proc: q1 equiv_k q2) => q1 = q2
%...rule out solutions not satisfying AA2
:- neq(Q1,Q2), equal_local(K,Q1,Q2):process(K), state(Q1), state(Q2).

%===AA3: q1-a->q11 and equiv_dom(a)(q1,q2)
%        => exists q22 s.t. q2-a->q22 and equiv_dom(q11,q22)
matched(A,Q1,Q11,Q2) :- trans(Q1,A,Q11), trans(Q2,A,Q22),
   equiv_dom(A,Q1,Q2), equiv_dom(A,Q11,Q22).
%...rule out solutions not satisfying AA3
:- trans(Q1,A,Q11), state(Q2), equiv_dom(A,Q1,Q2),
   not matched(A,Q1,Q11,Q2).
```

### B.2 Experimental results

Some preliminary experiments are presented in Table 9. We considered as benchmarks the parametrized specifications for the mutual exclusion problem given in [ŞEM03] and also for the dining philosophers problem[11]. We present the times for running `smodels 2.27` on a Linux machine with a 2.4 GHz Intel processor and 2 GB of RAM. In case an execution did not terminate within 30 minutes, a '–' is assigned in the table.

The first and second column of Table 9 give the instances of the problems considered together with the sizes of the *deterministic*[12] transitions systems modeling them and the number of processes $|Proc|$ (shortly, $|P|$) of the distributions.

[11] We considered only the safety requirement that no shared fork can be used simultaneously by the two philosophers that can have access to it.

[12] Despite our efforts to obtain nondeterministic specifications for the given problems (i.e., transition systems where nondeterministic choice really occurs), we ended up with deterministic transition systems (mainly due to complementation operations in the specifications).

Columns three and four show the times for solving Problem 28 by the heuristic of SYNASYNC that search for an ID and FD subautomaton, respectively a new SMODELS-based logic implementation (using the idea of guessing the transitions of the subautomaton). For the instances of Mutex($N$), the heuristic of SYNASYNC performs better than the new one (the original transition systems already satisfies ID and FD and apparently this helps the heuristic). For Phil($N$) (where ID and FD does not hold initially), no solution for $N = 3, 4$ is found by SYNASYNC (fact denoted by 'x' in Table 9), because its current implementation employs a heuristic greedy algorithm which is *not* a *complete* method. The new implementation finds, in turn, solutions for $N = 3, 4$.

Columns five and six present the times for solving Problem 29 by two logic implementations in SMODELS, based on Proposition 14 and Theorem 11, respectively. Since the specifications are *deterministic*, after we guess a subautomaton, we can implement either the (polynomial) characterization mentioned in Proposition 14 (see the details in [Mor99]) or the characterization of Theorem 11 (see the implementation given in the Appendix). As expected, due to the more difficult algorithms, we perform worse in the nondeterministic version. One optimization worth mentioning is that the logic program for the nondeterministic case has a tunable constant `max_local_state` which gives the maximum number of local states for each process (for Table 9, we used `max_local_state=3`). By tuning this constant, we can limit the logic program to look for synthesis solutions which are systems composed out of 'small' components.

To conclude, using a logic implementation we achieved the following:

– For SubautIDFD we have an implementation doing a complete search in the state space of solutions able to find solutions missed by the current implementation of the polynomial heuristic of SYNASYNC.
– For SubautAsync we are able to find solutions for $N = 3$. This is not impressive compared with SYNASYNC which for Mutex can go up to $N = 5$ (see [ŞEM03] for details). However, it beats the original construction of Zielonka which only works for $N = 2$ due to state space explosion. Moreover, the gain of this heuristic is that it makes a complete search in the state space of 'small' asynchronous automata whose global transition system is isomorphically embedded in the specification (so we are guaranteed solutions not exceeding the size of the specification).

The prototype implementations given in this work can be used as a starting point for more optimized synthesis procedures. Namely, the memory overheads involved in the implementation are (polynomial but still) very significant in larger instances. A special purpose synthesis procedure (i.e., a special purpose NP-solver) could potentially eliminate quite a lot of this memory overhead.

The benchmarks together with the above mentioned SMODELS-based logic implementations can be found at:
`http://www.fmi.uni-stuttgart.de/szs/tools/synasync/smodels/`