Checking Equivalences Between Concurrent Systems of Finite Agents (Extended Abstract)

Alexander Rabinovich
IBM Research Division
T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
e.mail: alik@watson.ibm.com

Abstract

Consider two synchronously communicating systems p and q over finite agents. Assume that one wants to check whether $p \sim q$ for one of the commonly used equivalences. We show that this question is PSPACE hard for all equivalences which lie between strong bisimulation and trace equivalences. For some equivalences exponential lower and upper bounds are proven. We also show that this problem is NP hard and co-NP hard even for a class of very simple finite agents.

Dataflow nets are the simplest systems of asynchronously communicating agents. Let $\sim$ be an equivalence which lies between strong bisimulation and trace equivalences. We show that there is no algorithm for checking $\sim$-equivalence between dataflow nets over finite state agents.

1. Introduction

1.1. Equivalences

There is a variety of semantics for concurrency which reflect the alternatives: linear time vs branching time, interleaving vs causality. Here we consider only interleaving semantics. In such a setting concurrent systems are described by labeled transition systems [Plo] or in a more classical terminology by automata (may be with infinite number of states). Usually a behavior equivalence $\sim$ is introduced on labeled transition systems and the semantical objects are the $\sim$ equivalence classes. The question whether system p behaves like (or implements) system q is mathematically reformulated as a question whether $p \sim q$. In the literature on concurrency many such equivalences were proposed. For example, in classical automata theory, two automata are equivalent iff they accept the same language. This equivalence is sometimes called *language* equivalence. *Weak trace* equivalence is an equivalence on labeled transition systems which

is even coarser than language equivalence. Two labeled transition systems are (weak) trace equivalent if the automata obtained from them by marking all states as accepting states are language equivalent. Weak trace equivalence is considered as the coarsest behavior equivalence of interest.

Language and weak trace equivalences completely ignore branching. For example, they identify the automata *described* by expressions $a(b+c)$ and $ab+ac$, which are considered as different in most theories of concurrency. Informally, their difference is justified as follows: the first automaton after performing a can choose between b and c; on the other hand, the second automaton after performing a is unable to choose; only one of actions b, c is available in the state it has reached.

Another extreme equivalence is *(strong) bisimulation* equivalence. It catches very subtle differences between labeled transition systems on the basis of their branching structure. It has a very appealing mathematical theory and is accepted as the finest behavior equivalence of interest for concurrency (it is often argued that strongly bisimilar labeled transition systems are indistinguishable for all reasonable notions of observations).

Many equivalences on labeled transition systems were studied in the literature. Failure equivalence [BHR], acceptance equivalence [He], weak bisimulation equivalence [Mi], observational congruence [Mi] are only few among many well investigated equivalences. There is no consensus what is the best equivalence or what criteria it should satisfy. But it seems that there exists a consensus that a good equivalence should lie between trace and bisimulation equivalences. This consensus is supported by

Empirical Fact: All equivalences studied in the literature lie between bisimulation and trace equivalences.[1]

1.2. Complexity of Checking Equivalences between Finite Agents

P. Kanellakis and S. Smolka examined in [KS] the computational complexity of checking different equivalences between finite state automata. The classical result of L. Stockmeyer and A. R. Meyer [SM] states that the complexity of checking language equivalence is PSPACE complete. It is shown in [KS] that trace equivalence, failure equivalence and a number of other equivalences are PSPACE complete.

However, polynomial algorithms for checking bisimulation equivalence, weak bisimulation equivalence and observational congruence were given in [KS]. Paige and Tarjan [PT] gave a very efficient algorithm for checking bisimulation equivalence; its complexity is $m+n\log m$, where n is a number of transitions and m is number of states. In [GV] and [Bl] polynomial algorithms are given for checking branching bisimulation equivalence and readiness equivalence. It is proved in [ABGS] that the bisimulation equivalence problem is PTIME complete.

[1] As every empirical fact this one also has an exception. Some equivalences which are congruences wrt action refinement distinguish between strong bisimilar automata

1.3. State Explosion

Concurrent CCS-like languages, in addition to sequential combinators like prefixing, choice and iteration use two new fundamental combinators: parallel composition ($||$) and hiding. Roughly speaking, parallel composition describes communication between components of a systems and hiding describes which communication events will become invisible and which will remain observable.

Complex systems are assembled from simple agents by applying parallel composition and hiding. The role of these operations is prominent for nets of communicating agents. For example, *hide* $a_1, \ldots a_l$ *in* $(p_1||p_2 \ldots ||p_n)$ describes the network of synchronous communicating agents $p_1, \ldots p_n$ in which the communication events $a_1, \ldots a_n$ are hidden.

From the complexity point of view, parallel composition is different from other operations: the size of automata $p||q$ is of the order $|p| \times |q|$. Therefore, if n_i is the size of p_i then the size of system $p_1||p_2 \ldots ||p_k$ is $n_1 \times n_2 \times \cdots \times n_k$ and it is exponential in the size $n_1 + n_2 \cdots + n_k + k$ of its description. This fact is known as state explosion.

Given descriptions *hide* $a_1, \ldots a_l$ *in* $(p_1||p_2 \ldots ||p_n)$ and *hide* $b_1, \ldots b_m$ *in* $(q_1||q_2 \ldots ||q_k)$ of two synchronously communicating systems p, q. Assume that one wants to check whether $p \sim q$ for one of the commonly used equivalences. A straightforward algorithm will construct automata p and q and then will check their $\sim$-equivalence. Since the sizes of p and q are exponential in the sizes of their description, the complexity of this algorithm is at least EXPTIME. Can descriptions of p, q in terms of their components p_i, q_j be used in order to obtain an efficient algorithm? J. F. Groote and F. Moller [GM] considered the problem of checking bisimulation equivalence between two systems $p = p_1||p_2 \ldots ||p_n$ and $q = q_1||q_2 \ldots ||q_n$ of finite automata. They developed a method that avoids the 'state explosion'-problem. It works only in the case when there is no communication between the components of the systems. Their algorithm is polynomial and works not only for strong bisimulation equivalence, but also for other equivalences which satisfy a certain set of axioms.

1.4. Our Contribution

We investigate the complexity of checking equivalences between networks of communicating finite agents.

1.4.1. Synchronous Communication First we consider the problem of checking equivalences between networks of synchronously communicating finite agents. Such a network p can be described as *hide* $a_1, \ldots a_l$ *in* $(p_1||p_2 \ldots ||p_n)$ where p_i are finite state automata and a_i are communication events. Here not only communications between components of a system are allowed, but also some communication events can be hiding (to become invisible τ-moves).

Given two networks p and q and an equivalence $\sim$ on automata. We show that the problem whether $p \sim q$ is: (1) PSPACE-hard for all equivalences which lie between strong bisimulation and trace equivalence. (2) it is NP hard and co-NP hard for all equivalences which lie between strong bisimulation and trace equivalence even in the

Agents	equivalences	lower bound	upper bound
Finite Automata	language and trace	DSPACE($\Omega(c^{n/\log n})$)	DSPACE(d^n)
	bisimulation	PSPACE	EXPTIME
	any equivalence between trace and bisimulation	PSPACE	
Finite Acyclic Automata	any equivalence between trace and bisimulation	co-NP hard NP hard	

Figure 1: Complexity results for synchronous communication

case when p_i, q_j are acyclic automata. (3) for language and trace equivalences the problem is EXPSPACE-complete and we provide DSPACE($\Omega(c^{n/\log n})$) lower bound and DSPACE($O(d^n)$) upper bound.

Note that the algorithm which first constructs p and q and then checks whether they are bisimulation equivalent using Paige and Tarjan algorithm, has EXPTIME complexity and gives us an upper bound for checking bisimulation. These results are summarized in figure 1.

We use parallel composition and hiding à la Hoare. However, the results stated above are valid for Milner parallel composition and restriction. They also remain valid in any language which is able to describe net operations.

1.4.2. Asynchronous Communication Dataflow is the simplest model of asynchronously communicating agents. A semantics of a dataflow net N over agents $p_1 \dots p_k$ can be expressed as *hide* $c_1, \dots c_m$ *in* $((p_1 \| p_2 \dots \| p_n) \| (ch_1 \| ch_2 \dots \| ch_r))$, where p_i are the components of the nets, ch_j are the communication channels which behave like unbounded buffers, and c_l are the internal ports of the net. Let $\sim$ be an equivalence which lies between strong bisimulation and trace equivalences. We show that there is no algorithm for checking $\sim$-equivalence between dataflow nets over finite state agents.

Section 2 provides basic definitions. Section 3 states main results. Ideas of proofs are only outlined. Section 4 - conclusion.

2. Basic Definitions

Section 2.1 introduces labeled transition systems. In section 2.2 trace and strong bisimulation equivalences are defined. The former is considered as the coarsest equivalence and the later as the most discriminating equivalence for interest of interleaving semantics for concurrency. In section 2.3 synchronization and hiding operations on labeled transition systems are defined. Section 2.4 briefly describes dataflow networks - the simplest asynchronous model.

2.1. Labeled Transition Systems

Definition 1: *A Labeled Transition System (LTS) consists of*

- *Set of states* Q.
- *Initial state* $q_0 \in Q$.
- *Communication alphabet* A *and a special invisible action* τ ($\tau \notin A$).
- *Transition Relation:* $\longrightarrow$ *a subset of* $Q \times \{A \cup \{\tau\}\} \times Q$.

An action is either a communication or τ. We use $q \xrightarrow{a} q'$ as a notation for a transition from state q via action a to state q'; we say that a labeled transition system T is *finite* if it has a finite set of states, a finite alphabet and a finite transition relation. We say that T is *acyclic* if it does not contain a cyclic path (a path of the from $q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \cdots \xrightarrow{a_{k-1}} q_k \xrightarrow{a_k} q_1$).

Remarks: *(Contrasting the above definition with classical automata theory).* (1) The alphabet is implicitly presented in automata theory. Usually it is extracted from the transition diagram which defines an automaton. The role of the alphabet in concurrency is much more important. The appearance of a in the alphabet of T, but not as a label of a transition of T implies that a will be blocked in any system which runs in parallel with T. (2) In our definition, unlike the definition of automaton, the set of accepting states was not mentioned. Implicitly, all states are accepting states. It is a technical decision which simplifies our presentation.

2.2. Trace and Strong Bisimulation Equivalences

A finite alternating sequence $q_0,\ a_0,\ q_1,\ a_1, \cdots a_{n-1},\ q_n \cdots$ of states of LTS T and actions of T is an *execution sequence* of T if q_0 is the initial state of T and $q_{i-1} \xrightarrow{a_i} q_i$ are transitions of T.

A *trace* of T is the sequence of communications which is obtained from an execution sequence by deleting the states of T and τ actions. We use the notation $trace(T)$ for the sets of traces of labeled transition system T.

Definition 2: *Labeled transition systems* T *and* T_1 *over the same alphabet are trace equivalent (notation* $T \sim_{trace} T_1$*) if* $trace(T) = trace(T_1)$.

The notion of strong bisimulation was introduced by D. Park [Pa] and plays a very important role in concurrency [Mi].

Definition 3: *(Strong Bisimulation [Pa].) Let* $T^1 = < Q^1,\ q_0^1,\ A,\ \longrightarrow_1 >$ *and* $T^2 = < Q^2,\ q_0^2,\ A,\ \longrightarrow_2 >$ *be labeled transition systems over the same alphabet. A relation* R *between the states of* T^1 *and* T^2 *is called a (strong) bisimulation if*

- $q_0^1 R q_0^2$
- *Whenever* qRp *and* $q \xrightarrow{a}_1 q'$, *then* $p \xrightarrow{a}_2 p'$, *for some* p' *for which* $q'Rp'$.
- *Whenever* qRp *and* $p \xrightarrow{a}_2 p'$, *then* $q \xrightarrow{a}_1 q'$, *for some* q' *for which* $q'Rp'$.

Definition 4: *Labeled transition systems T and T_1 are (strong) bisimulation equivalent (notation $T \sim_{bis} T_1$) if there exists a (strong) bisimulation relation between their states.*

We say that an equivalence $\sim_1$ refines an equivalence $\sim_2$ (notations $\sim_1 \leq \sim_2$) if $T^1 \sim_1 T^2$ implies $T^1 \sim_2 T^2$. We say that $\sim$ lies between $\sim_1$ and $\sim_2$ if $\sim_1 \leq \sim \leq \sim_2$.

2.3. Operations on Labeled Transition Systems

We define here synchronization (parallel composition) and hiding operations à la Hoare which are more convenient for our purposes. But all results given in this paper are valid if Milner's parallel composition and restriction are used or if one uses combinators from other CCS-like languages which are able to express net operations.

Synchronization. (infix notations - $||$) The synchronization T of two labeled transition systems $T^1 =< Q^1, q_0^1, A^1, \longrightarrow_1 >$ and $T^2 =< Q^2, q_0^2, A^2, \longrightarrow_2 >$ is defined as follows:

- States: $Q = Q^1 \times Q^2$
- The initial state is the pair consisting of the initial state of T^1 and the initial state of T^2.
- Communication alphabet $A = A^1 \cup A^2$.
- Transitions are given by the following inference rules:
  1. if a is not in $A^1 \cap A^2$ then
     $$\frac{q_1 \xrightarrow{a}_1 q_2}{(q_1, q) \xrightarrow{a} (q_2, q)} \qquad \frac{q_1 \xrightarrow{a}_2 q_2}{(q, q_1) \xrightarrow{a} (q, q_2)}$$
  2. τ-transitions
     $$\frac{q_1 \xrightarrow{\tau}_1 q_2}{(q_1, q) \xrightarrow{\tau} (q_2, q)} \qquad \frac{q_1 \xrightarrow{\tau}_2 q_2}{(q, q_1) \xrightarrow{\tau} (q, q_2)}$$
  3. if a is in $A^1 \cap A^2$
     $$\frac{q_1 \xrightarrow{a}_1 q_1' \qquad q_2 \xrightarrow{a}_2 q_2'}{(q_1, q_2) \xrightarrow{a} (q_1', q_2')}$$

Hiding . *hide a in T^1* is the labeled transition system defined as follows:

- States: the same as the states of T^1.
- Communication alphabet: $A = A^1 - \{a\}$.
- The initial state is the initial state of T^1
- Transitions are defined by the following inference rules:
  1. if $a' \neq a$ then
     $$\frac{q_1 \xrightarrow{a'}_1 q_2}{q_1 \xrightarrow{a'} q_2}$$
  2. τ-transitions:
     $$\frac{q_1 \xrightarrow{a}_1 q_2}{q_1 \xrightarrow{\tau} q_2} \qquad \frac{q_1 \xrightarrow{\tau} q_2}{q_1 \xrightarrow{\tau} q_2}$$

Let $A = \{a_1, a_2, \cdots a_n\}$ be a set of actions. We use notation *hide A in T* for *hide a_1 in (hide a_2 in $\cdots$(hide a_n in T)...)*; Since *hide a in (hide b in T))* = *hide b in (hide a in T))* this is a well defined notation.

2.4. Dataflow Networks

A dataflow is the simplest model of asynchronously communicating agents. A dataflow network consists of agents communicating through unbounded buffer channels (see Fig. 2). Agents communicate with each other and with an environment by passing data over the channels. As for the synchronous case, an agent is described by a labeled transition system; the communication alphabet of a dataflow agent is structured as Cartesian product $PORTS \times DATA$. An action $< c,\ d >$ is said to be a communication through port c which passes the data value d. The ports of the net are partitioned into input, output and internal. Data are coming from the environment through input ports and are going to the environment through output ports. For example, I_1, I_2 are the input ports, O_1, O_2 are the output ports and all others are the internal ports for the net in Fig. 2. The channels of a network behave like unbounded buffers (FIFO queues).

The buffer from port I to port O over data set Δ is described by the following labeled transition system:

Definition 5: *(Buffer)* $BUF(I \rightarrow O)$ *over* Δ *has the following labeled transition system:*

- *States:* Δ^* *- finite strings over* Δ.
- *Initial state - the empty string* ϵ.
- *Communication alphabet -* $\{I,\ O\} \times \Delta$.
- *Transition Relation: for every* $d \in \Delta$ *and every state* $s \in \Delta^*$ *there are transitions:* $s \xrightarrow{<I,d>} sd$ *and* $ds \xrightarrow{<O,d>} s$

Therefore, the buffer is a simple infinite state labeled transition system.

A dataflow network has a very clear operational semantics. This semantics is equivalent to the result obtained first by taking synchronization of all its component agents and the buffer channels which connect them and then hiding the actions on the internal ports. For example, the semantics of the net in Fig. 2 is: *hide* Int *in* $(A_1 \| A_2 \| A_3 \| BUF_1 \| \cdots \| BUF_7)$, where Int is the set of the communications through internal ports $c_1 \ldots c_{10}$.

3. Main Results

In all theorems stated in this section $p_1 \ldots p_n$, q are finite labeled transition systems and $\sim$ is an equivalence which lies between strong bisimulation and trace equivalences.

Theorem 3.1: *The problem whether hide* $a_1, \ldots a_k$ in $(p_1 \| p_2 \ldots p_n) \sim$ *hide* $b_1, \ldots b_m$ in $(q_1 \| q_2 \ldots q_r)$ *is NP hard and co-NP hard, even in the case when* p_i *and* q_j *are acyclic transition systems.*

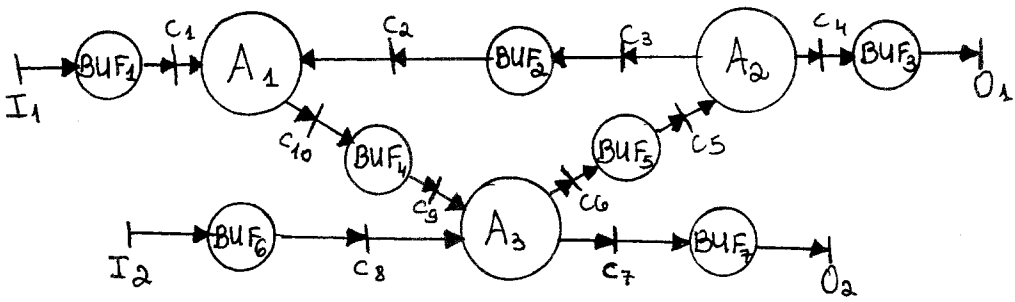

Figure 2: A dataflow net

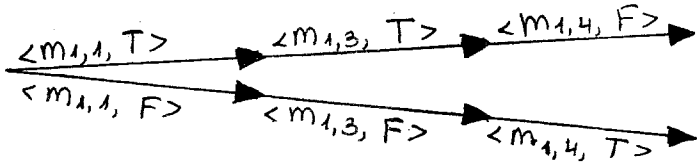

Figure 3: X_1 is over alphabet $\{m_{1,1},\ m_{1,3},\ m_{1,4}\} \times \{T,\ F\}$

Proof: We will show here only co-NP hardness. The proof for NP-hardness is similar. We provide a reduction from the tautology problem. The tautology problem is defined as follows. There is a finite set $\{x_1,\ x_2, \ldots x_n\}$ of propositional variables. A literal is either a variable or its negation. A formula A is in 3-Disjunctive Normal Form if it is of the form $\vee_i\ c_i$, where each c_i is a conjunction of three literals $c_i = \wedge_{j=1}^{3} l_{i,j}$. The problem whether $\forall x_1 \ldots x_n\ A$ is true is known as 3-DNF tautology problem, and it is co-NP complete.

We are going to construct an expression which simulates this formula. It will have the form: $X_1 \| \cdots \| X_n \| C_1 \| C_2 \cdots \| C_k \| D$

The labeled transition system X_i will 'simulate' variable x_i; C_i will 'simulate' conjunct c_i and D will 'simulate' disjunction of conjuncts.

We will use structured alphabet - labels are of the form $< ch,\ v >$.

Rather than giving formal definitions we provide a generic example of the construction.

Assume that x_1 occurs positively in c_1 and c_3 and negatively in c_4. Then X_1 is the LTS in Fig. 3. Assume that c_3 is $x_1 \wedge x_2 \wedge \neg x_4$; C_3 will be as in Fig. 4. Disjunction is simulated by D in Fig. 5. Let us sketch some ideas underlying constructions of X_1, C_3 and D. X_1 is a tree with two branches. The upper (lower) branch corresponds to the choice of value $True$ ($False$) for x_1. For every clause c_i in which x_1 occurs, X_1 sends the value contributed by x_1 to this clause along port $m_{1,i}$; if x_1 occurs positively (negatively) in C_i then X_1 contains action $< m_{1,i},\ T >$ ($< m_{1,i},\ F >$) on its upper branch and action $< m_{1,i},\ F >$ ($< m_{1,i},\ T >$) on its lower branch. C_3 reads the values contributed by the variables to c_3. If all variables contribute value $True$ then C_3 outputs $< d_3,\ T >$, otherwise it outputs $< d_3,\ F >$. D reads the values contributed by all c_i. If at least one clause contributes $True$, then D outputs $< Result,\ T >$, otherwise it outputs $< Result,\ F >$.

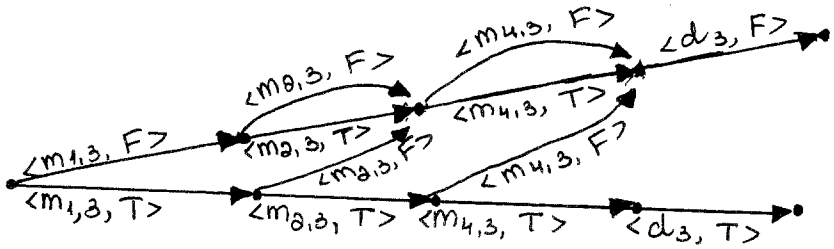

Figure 4: C_3 is over alpabet $\{m_{1,3},\ m_{2,3},\ m_{4,3},\ d_3\} \times \{T,\ F\}$

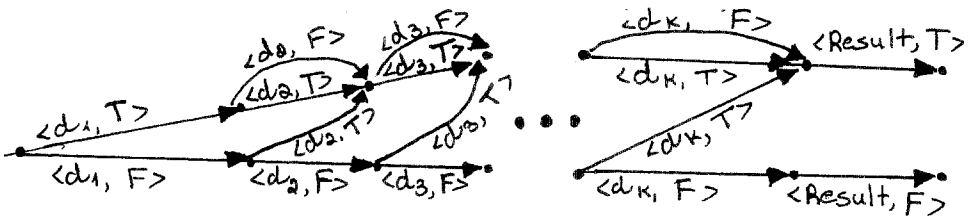

Figure 5: D is over alphabet $\{d_1,\ d_2, \ldots d_k,\ Result\} \times \{T,\ F\}$

Let M be the alphabet of $X_1 \| \cdots \| X_n \| C_1 \| C_2 \cdots \| C_k \| D$ and B is defined as $M - \{< Result,\ T >, < Result,\ F >\}$. Consider labeled transition system $Sys = hide\ B\ in\ X_1 \| \cdots \| X_n \| C_1 \| C_2 \cdots \| C_k \| D$. Its description is polynomial in the size of 'simulated' formmula $\forall x_1 \ldots x_n.(c_1 \vee c_2 \vee \cdots \vee c_k)$. It is easy to observe that

1. *Sys* is an acyclic labeled transition system.
2. All its maximal paths have the same length l which is equal to the number of occurrences of variables +k+1=4k+1.
3. The last action in any maximal path of *Sys* is either $< Result,\ T >$ or $< Result,\ F >$.
4. All last actions in the maximal paths are $< Result,\ T >$ iff the formula is a tautology; in this case the *Sys* is bisimulation equivalent to the system $\tau^l < Result,\ T >$ (see Fig. 6).
5. Moreover, for any equivalence $\sim$ between trace and bisimulation equivalences $Sys \sim \tau^l < Result,\ T >$ iff the original formula is tautology.

□

Remarks: Hiding is not essential for co-NP hardness; one can compare systems $X_1 \| \cdots \| X_n \| C_1 \| C_2 \cdots \| C_k \| D$ with $X_1 \| \cdots \| X_n \| C_1 \| C_2 \cdots \| C_k \| D \| TRUE$, were $TRUE$

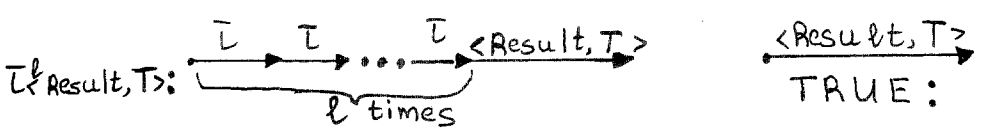

Figure 6: $\tau^l < Result,\ T >$ and $TRUE$ are over alphabet $\{Result\} \times \{T,\ F\}$

is the LTS over alphabet $\{< Result,\ T >,\ < Result,\ F >\}$ given in Fig. 6. They are equivalent iff the simulated formula is a tautology.

Theorem 3.2: *The problem whether hide* $a_1, \ldots a_n$ in $(p_1 \| p_2 \ldots \| p_k)$ *is* $\sim$*-equivalent to hide* $b_1, \ldots b_m$ in $(q_1 \| q_2 \ldots \| q_r)$ *is PSPACE-hard.*

Proof: (Outline) For each deterministic Turing machine M that is space bounded by a polynomial $p(n)$, we give a polynomial algorithm that takes a string x as an input and produces two expressions in the form *hide* $a_1, \ldots a_n$ *in* $(p_1 \| p_2 \ldots p_n)$. The expressions are equivalent iff M accepts x. The central idea is to construct a short description of a system SIM which 'simulates' the behavior of M on x. In our proof, for x of length n the SIM will have a form: $CONTROL_n \| CELL_1 \| CELL_2 \| \cdots \| CELL_{p(n)}$. $CELL_i$ represents the contents of the tape cell i in the computation of M on x. $CONTROL_n$ simulates the control of M. Cells are communicating with control; there is no direct communication among the cells. The size of $CELL_i$ is independent of n and $CONTROL_n$ has a size $O(n \log n)$. Space limitations prevent us from giving more details about the construction. □

Theorem 3.3: *The problem whether hide* $a_1, \ldots a_k$ in $(p_1 \| p_2 \ldots p_n)$ *is trace equivalent to hide* $b_1, \ldots a_m$ in $(q_1 \| q_2 \ldots q_r)$ *is EXPSPACE-complete.*

Proof: (Outline of ideas) H. B. Hunt [Hu] showed that the problem of inequivalence of regular expressions over operations *union, concatanetion, Kleene star,* and *intersection* is EXPSPACE-complete. We emulate[2] the simplified proof of this fact given by Furer [Fu]. The major difficulties in the emulation are: (1) How intersection operation can be 'simulated' by synchronization; (2) To show that the problem is still EXPSPACE-hard even when restricted to the expressions in the *canonical* form *hide A in* $(Prefix(e_1) \| Prefix(e_2) \cdots \| Prefix(e_n))$, where e_i are expressions over *union, concatanetion* and *Kleene star*, and *Prefix(L)* is a notation for the operation on languages which returns all the prefixes of the words in the language L. The proof gives $DSPACE(c^{n/\log n})$ lower bound. The upper bound of $DSPACE(O(d^n))$ is obtained similarly to the proof of theorem 13.14 in [HU]. □

Theorem 3.4: *The problem of* $\sim$*-equivalence of two dataflow nets over finite labeled transition systems is not recursively enumerable.*

Proof: (Hints) In the full paper we show that any Turing machine can be 'simulated' by a finite transition system and two buffers (buffers simulate the tape and a finite transition system simulates the control). The divergence problem for Turing machines is reducible to the equivalence problem for dataflow nets. □

4. Conclusion

We demonstrated that the problem of equivalence of synchronously communicating systems of finite agents is PSPACE-hard for any equivalence between bisimulation and

[2] Our emulation was inspired by A. Mayer and L. Stockmeyer paper [MS].

trace equivalences. As it was mentioned in the introduction, in order to check bisimulation equivalence between $p = hide\ a_1, \ldots a_l\ in\ (p_1 || p_2 \ldots || p_n)$ and $q = hide\ b_1, \ldots b_m\ in\ (q_1 || q_2 \ldots || q_k)$, one can first construct p and q and then apply the algorithm given by Paige and Tarjan [PT]. This procedure requires exponential time. Our conjucture is: EXPTIME-lower bound can be proved for all equivalences between bisimulation and trace. Recently, Larry Stockmeyer [Sto] proved $EXPTIME$ lower bound for checking bisimulation and weak bisimulation equivalences.

We proved EXPSPACE-completness for the problem of checking trace equivalence between synchronous systems of finite agents. Let us mention other equivalences for which we can prove EXPSPACE lower bound. Bisimulation equivalence was originally defined as the limit of the sequence $\sim_0, \sim_1, \cdots \sim_k \cdots$ of successively finer equivalences (see [Mi] page 224). We can show that for every fixed k checking $\sim_k$ equivalence between systems of synchronously communicating agents is EXPSPACE-hard. Language equivalence is defined on automata like in classical automata theory.[3] Synchronization and hiding can be defined in a natural way on the automata. The proof of EXPSPACE-completeness for language equivalence is very similar to the proof for trace equivalence.

Theorems 3.1, 3.2 provide lower bounds on complexity of checking an equivalence between communicating systems of the form $hide\ a_1, \ldots a_k\ in\ (p_1 || \ldots || p_n)$. Actually, our proofs give stronger results. Namely, the lower bounds of these theorems hold even for the systems of the form $p_1 || \ldots || p_n$ with *binary communications* (i.e., every action a is in the alphabets of at most two among $p_1, \ldots p_n$).

The results and proofs of the paper work not only for parallel composition and hiding à la Hoare that were considered here, but also for CCS parallel composition and restriction [Mi] and for other languages in which *net operations* can be described. Moreover, the results hold not only for all equivalences between trace and bisimulation equivalences, but for all equivalences which satisfy a cumbersome semantical requirement. We preferred to state the theorems in this way, a bit sacrificing generality, but without introducing these quite unnatural semantical conditions.

Acknowledgments

The author is grateful to Jan Friso Groote, Albert R. Meyer, Alain Mayer and Larry Stockmeyer for fruitful discussions and comments.

References

[ABGS] C. Alvarez, J. Balcazar, J. Gabarro and M. Santa. Parallel Complexity in the design and analysis of concurrent systems. In *PARL 91*, volume 505 of *Lect. Notes in Computer Science*. Springer Verlag, 1991.

[Bl] B. Bloom. Ready Simulation, Bisimulation, and the Semantics of CCS-like languages. *Ph.D. Thesis*, Technical Report MIT/LCS/TR-491, 1990.

[3] An automaton is a labeled transition system with a set of accepting states

[BHR] S. Brookes, C. Hoare, and A. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.

[Fu] M. Furer. The complexity of inequivalence problem for regular expressions with intersection. In *Proc. 7th ICALP*, volume 85 of *Lect. Notes in Computer Science*. Springer Verlag, 1980.

[GM] J. F. Groote and F. Moller. Verification of Parallel System via Decomposition. *Preliminary Report, CWI 1991.*

[GV] J. F. Groote and F. Vaandrager. An Efficient Algorithm for Branching Bisimulation and Stuttering Equivalence. In *International Conference on Automata, Languages and Programming*, volume 443 of *Lect. Notes in Computer Science*. Springer Verlag, 90.

[He] M. Hennessy. *Algebraic Theory of Processes.* MIT Press, Cambridge, Massachusetts, 1988.

[Ho] C. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, Englewood Cliffs, 1985.

[HU] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Lamguages and Computations*, Addison-Wesley, Reading, MA, 1979.

[Hu] H. Hunt. The equivalence problem for regular expressions with intersection is not polynomial in tape, *Tech. Rep. TR 73-161*, Cornell University, 1973.

[MS] A. J. Mayer and L. J. Stockmeyer. World Problems - This Time with Interleaving. *Technical Report, IBM RJ8180, July, 1991.*

[Mi] R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.

[KS] P. C. Kanellakis and S. A. Smolka. CCS Expressions, Finite State Processes, and Three Problems of Equivalence. In *Information and Computation*, volume 86, 1990.

[PT] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987.

[Pa] D. Park. Concurrency and Automata on infinite sequences. volume 104 of *Lect. Notes in Computer Science.* Springer Verlag, 1981.

[Plo] G. Plotkin. A structured approach to operational semantics. FN 19 DAIMI, Aarhus Univ., 1981.

[SM] L. J. Stockmeyer and A. R. Meyer. Word Problems Requiring Exponential Time. In *Proc. 5th ACM Symp. on Theory of Computing, 1973.*

[Sto] L. J. Stockmeyer. Private communication, Jan 1992.