INFORMATION AND COMPUTATION 139, 111–129 (1997)
ARTICLE NO. IC972661

Complexity of Equivalence Problems for Concurrent Systems of Finite Agents

Alexander Rabinovich

Department of Computer Science, The Sackler School of Exact Sciences, Tel Aviv University, Tel Aviv, Israel 69978
E-mail: rabino@math.tau.ac.il

A concurrent system of synchronous communicating agents is assembled from simpler sequential agents by parallel composition and hiding. For example, *hide* $a_1, \ldots a_l$ *in* $(p_1 \parallel p_2 \cdots \parallel p_n)$ describes the system of communicating agents $p_1, \ldots p_n$ in which the communication events $a_1, \ldots a_l$ are hidden. Consider descriptions of two systems p and q of synchronously communicating finite state agents. Assume that one wants to check whether $p \sim q$ for one of the commonly used equivalence $\sim$. We show that this question is PSPACE hard for all equivalences that 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. © 1997 Academic Press

1. INTRODUCTION

1.1 Equivalences

There is a variety of semantics for concurrency that reflects 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 (maybe with infinite number of states). Usually a behavior equivalence $\sim$ (an implementation preorder $\preccurlyeq$) is introduced on labeled transition systems. The question whether system p behaves like (implements) system q is mathematically reformulated as a question whether $p \sim q$ ($p \preccurlyeq q$). In the literature on concurrency many such equivalences were proposed (see [RvG]). 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 that 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.

0890-5401/97 $25.00
Copyright © 1997 by Academic Press
All rights of reproduction in any form reserved.

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], and observational congruence [Mi] are only a few among many well investigated equivalences [RvG]. 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.

Kanellakis and Smolka examined in [KS] the computational complexity of checking different equivalences between finite state automata. The classical result of Meyer and Stockmeyer [MeS] 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.

1.3. State Explosion

Concurrent CCS-like languages, in addition to sequential combinators like prefixing, choice, and iteration use two new fundamental combinators: parallel

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

composition ($\|$) and hiding. Roughly speaking, parallel composition describes communication between components of a system and hiding describes what communication events will become invisible and what 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* $(p_1 \| p_2 \cdots \| p_n)$ describes the network of synchronous communicating agents $p_1, \ldots p_n$ in which the communication events $a_1, \ldots a_l$ 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 $p_1 \| p_2 \cdots \| p_k$ 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* $(p_1 \| p_2 \cdots \| p_n)$ and *hide* $b_1, \ldots b_m$ *in* $(q_1 \| q_2 \cdots \| q_k)$ of two synchronously communicating systems p and 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 the descriptions of p, q in terms of their components p_i, q_j be used in order to obtain an efficient algorithm? Groote and Moller [GM] considered the problem of checking bisimulation equivalence between two systems $p = p_1 \| p_2 \cdots \| p_n$ and $q = q_1 \| q_2 \cdots \| q_n$ 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.

Such a network p can be described as *hide* $a_1, \ldots a_l$ *in* $(p_1 \| p_2 \cdots \| p_n)$ 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 equivalences even in the 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.

It is clear that a lower bound for checking an equivalence $\sim$ is also a lower bound for checking any implementation preorder that generates $\sim$.

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	

FIG. 1. Summary of our complexity results

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

The rest of this paper is organized as follows: Section 2 provides basic definitions. In Section 3 lower bounds are proved for a net over acyclic automata. In Section 4 PSPACE hardness of checking any reasonable equivalence for nets of automata is given. In Section 5 we show EXPSPACE completeness of verifying trace equivalence. Section 6 gives a conclusion and further results.

The main results of the paper were announced in [Ra].

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 latter 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 introduces net expressions—a language for description of nets.

2.1. Labeled Transition Systems

DEFINITION. 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: $\rightarrow$ 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 the automata theory. Usually it is extracted from the transition diagram that 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 that runs in parallel with T. (2) In our definition, unlike in the definition of automaton, the set of accepting states was not mentioned. Implicitly, all states are accepting states. It is a technical decision that simplifies our presentation.

2.2. Trace and Strong Bisimulation Equivalences

A finite alternating sequence $q_0, a_1, q_1, a_1, ..., a_n, q_n \cdots$ 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 that 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 Park [Pa] and plays a very important role in concurrency [Mi].

DEFINITION 3 (Strong Bisimulation [Pa]). Let $T^1 = \langle Q^1, q_0^1, A, \rightarrow_1 \rangle$ and $T^2 = \langle Q^2, q_0{}^2, A, \rightarrow_2 \rangle$ 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 $q \xrightarrow{\alpha}_1 q'$, then $p \xrightarrow{\alpha}_2 p'$, for some p' for which $q'Rp'$.
- Whenever qRp and $p \xrightarrow{\alpha}_2 p'$, then $q \xrightarrow{\alpha}_1 q'$, for some q' for which $q'Rp'$.

(Here $\alpha \in A \cup \{\tau\}$)

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 $\sim_1 \leqslant \sim_2$) 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 $\sim_1 \leqslant \sim \leqslant \sim_2$.

2.3. Operations on Labeled Transition Systems

We define here synchronization (parallel composition) and hiding operations à la Hoare [Ho] which are more convenient for our purposes. Yet, 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 that are able to express net operations.

Synchronization (infix notations—$\|$). The synchronization T of two labeled transition systems $T^1 = \langle Q^1, q_0^1, A^1, \rightarrow_1 \rangle$ and $T^2 = \langle Q^2, q_0^2, A^2, \rightarrow_2 \rangle$ is defined as follows:

- States: $Q = Q^1 \times Q^2$.
- The initial state is (q_0^1, q_0^2).
- Communication alphabet $A = A^1 \cup A^2$.
- Transitions are given by the following inference rules:

1. if a is a communication 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 a communication 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' is a communication in A and $a' \neq a$ then

$$\frac{q_1 \xrightarrow{a'}_1 q_2}{q_1 \xrightarrow{a'} q_2}$$

2. τ transitions:

$$\frac{q_1 a \xrightarrow{a}_1 q_2}{q_1 \xrightarrow{\tau} q_2} \qquad \frac{q_1 \xrightarrow{\tau} q_2}{q_1 \xrightarrow{\tau} q_2}$$

Let $A = \{a_1, a_2, ..., a_n\}$ 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. Note also that synchronization is commutative and associative, hence $p_1 \| p_2 \cdots \| p_n$ is well defined.

2.4. Net Expressions

A system of concurrently communicating agents is assembled from simple systems by parallel composition and hiding. Below, net expressions are introduced; they describe systems of concurrently communicating agents.

Let $\{C_T\}$ be a set of constant indexed by labeled transition systems.
The set of net expressions is defined by:

$$EXP := C_T \mid EXP \parallel EXP \mid \textit{ hide } a_1 \cdots a_n \textit{ in } EXP.$$

Semantics $[\![\ ]\!]$ assigns to any net expression a labeled transition system. It is defined inductively in a standard way:

- $[\![C_T]\!] = T$
- $[\![E_1 \parallel E_2]\!] = [\![E_1]\!] \parallel [\![E_2]\!]$
- $[\![\textit{hide } a_1 \cdots a_n \textit{ in } EXP]\!] = \textit{hide } a_1 \cdots a_n \textit{ in } [\![EXP]\!]$.

We use notation $[\![E]\!]_{trace}$ for the set of traces of the labeled transition system $[\![E]\!]$; we say that expressions are bisimilar (trace equivalent) if their transition systems are bisimilar (trace equivalent). The size of the net expressions is defined inductively:

- $size(C_T) =$ number of transitions + number of states + the size of the alphabet of T.
- $size(E_1 \parallel E_2) = size(E_1) + size(E_2) + 1$.
- $size(\textit{hide } a_1 \cdots a_n \textit{ in } EXP) = size(EXP) + n$.

A normal form net expression is an expression of the form $\textit{hide } a_1 \cdots a_n \textit{ in } (C_{T_1} \parallel C_{T_2} \parallel \cdots \parallel C_{T_m})$.
One can easily show

FACT 1. *There exists an algorithm that for every net expression E finds a normal form net expression E' such that E is bisimulation equivalent to E' and the size of E is equal to the size of E'.*

Proof. This fact follows from the commutativity of $\parallel$ and the following two laws:

1. $\textit{hide } a \textit{ in } T \sim_{bis} \textit{hide } b \textit{ in } T'$, where b does not appear in the alphabet of a labeled transition system T and T' is obtained from T by renaming all occurrences of a to b.

2. $(\textit{hide } a \textit{ in } T_1) \parallel T_2 \sim_{bis} \textit{hide } a \textit{ in } (T_1 \parallel T_2)$, provided a is not in the alphabet of T_2. ∎

3. LOWER BOUNDS FOR NETS OVER ACYCLIC SYSTEMS

Let $p_1 \cdots p_n, q_1 \cdots q_m$ be finite labeled transition systems and $\sim$ be an equivalence that lies between strong bisimulation and trace equivalences.

THEOREM 2. *The problem whether $\textit{hide } a_1, \ldots a_k \textit{ in } (p_1 \parallel p_2 \cdots p_n) \sim \textit{hide } b_1, \ldots b_m \textit{ in } (q_1 \parallel q_2 \cdots q_r)$ is NP hard and co-NP hard, even in the case when p_i and q_j are acyclic transition systems.*

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 $\bigvee_i c_i$, where each c_i is a conjunction of three literals $c_i = \bigwedge_{j=1}^{3} l_{i,j}$. The problem whether $\forall x_1 \cdots x_n\ A$ is true is known as 3-DNF tautology problem, and it is co-NP complete.

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

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

We will use structured alphabet—labels are of the form $\langle ch, v \rangle$.

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. 2. Assume that c_3 is $x_1 \wedge x_2 \wedge \neg x_4$; C_3 will be as in Fig. 3.

Disjunction is simulated by D in Fig. 4.

Let us sketch some ideas underlying constructions of X_1, C_3, and D.

X_j is a tree with two branches. The upper (lower) branch corresponds to the choice of value *True* (*False*) for x_j. For every clause c_i in which x_j occurs, X_j sends the value contributed by x_j to this clause along port $m_{j,i}$; if x_j occurs positively (negatively) in C_i then X_j contains action $\langle m_{j,i}, T \rangle$ ($\langle m_{j,i}, F \rangle$) on its upper branch and action $\langle m_{j,i}, F \rangle$ ($\langle m_{j,i}, T \rangle$) on its lower branch.

C_i reads the values contributed by the variables to c_i. If all variables occuring in c_i contribute value *True* then C_i outputs $\langle d_i, T \rangle$, otherwise it outputs $\langle d_i, F \rangle$. Recall that in general, c_i contains three occurrences of distinct variables x_l, x_m, x_r ($l < m < r$). The transition diagram for C_i is obtained from the transition diagram for C_3 in Fig. 2 by replacing index 3 by i and replacing indexes 1, 2, and 4 by l, m, and r.

D reads the values contributed by all c_i. If at least one clause contributes *True*, then D outputs $\langle Result, T \rangle$, otherwise it outputs $\langle Result, F \rangle$.

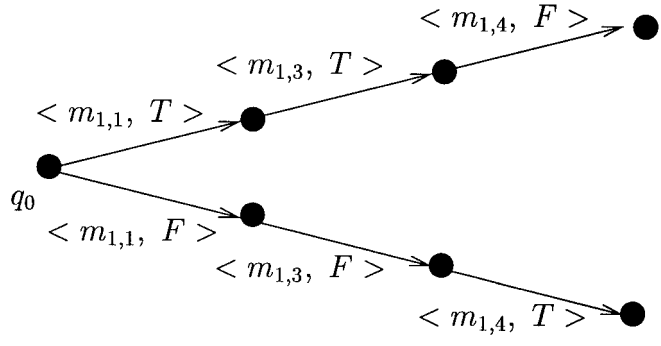

FIG. 2. X_1 is over alphabet $\{m_{1,1}, m_{1,3}, m_{1,4}\} \times \{T, F\}$.

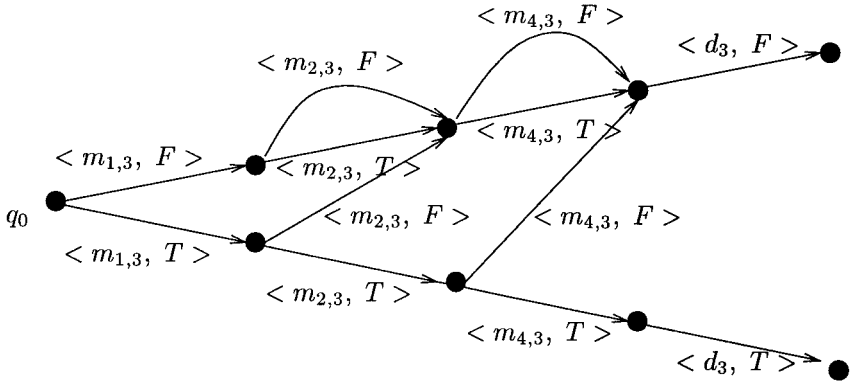

FIG. 3. C_3 is over alphabet $\{m_{1,3}, m_{2,3}, m_{4,3}, d_3\} \times \{T, F\}$.

Let M be the alphabet of $X_1 \parallel \cdots \parallel X_n \parallel C_1 \parallel C_2 \cdots \parallel C_k \parallel D$ and B is defined as $M - \{\langle Result, T\rangle, \langle Result, F\rangle\}$. Consider the labeled transition system $Sys = hide\ B\ in\ X_1 \parallel \cdots \parallel X_n \parallel C_1 \parallel C_2 \cdots \parallel C_k \parallel D$. Its description is polynomial in the size of simulated formula $\forall x_1 \cdots x_n.(c_1 \vee c_2 \vee \cdots \vee c_k)$. It is easy to observe that:

1. *Sys* is an acyclic labeled transition system.

2. All its maximal paths have the same length l that 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 $\langle Result, T\rangle$ or $\langle Result, F\rangle$.

4. All last actions in the maximal paths are $\langle Result, T\rangle$ iff the formula is a tautology; in this case the *Sys* is bisimulation equivalent to the system $\tau^l\langle Result, T\rangle$ (see Fig. 5). Hence, if the formula is a tautology, then $Sys \sim \tau^l\langle Result, T\rangle$ for any equivalence $\sim$ which is coarser than bisimulation equivalence.

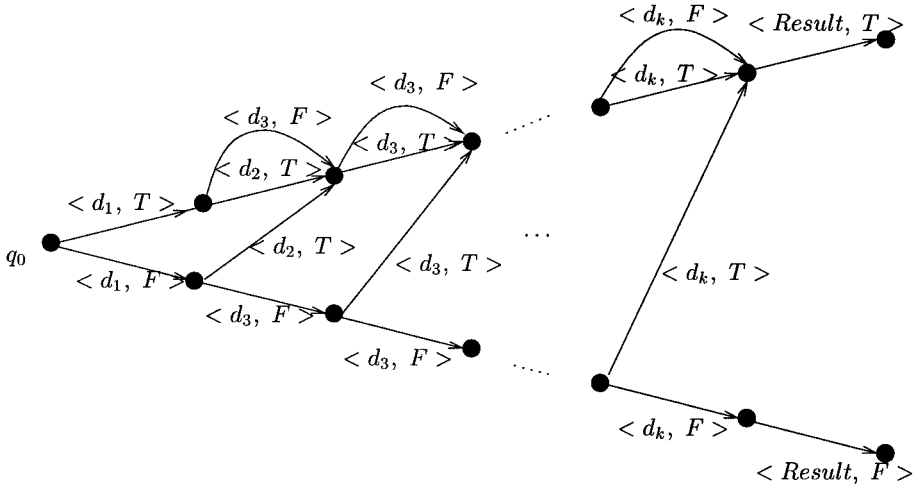

FIG. 4. D is over alphabet $\{d_1, d_2, \ldots d_k, Result\} \times \{T, F\}$.

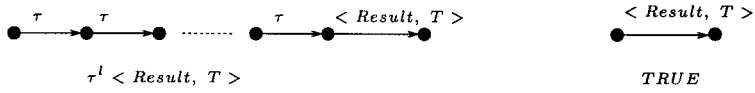

FIG. 5. $\tau^l\langle Result, T\rangle$ and *TRUE* are over alphabet $\{Result\} \times \{T, F\}$

5. *Sys* has a maximal path with the last action labeled by $\langle Result, F\rangle$ iff the formula is not a tautology; in this case *Sys* is not trace equivalent to $\tau^l\langle Result, T\rangle$. Hence, if the formula is not a tautology, then $Sys \not\sim \tau^l\langle Result, T\rangle$ for any equivalence $\sim$ that refines trace equivalence.

6. From 4 and 5, it follows that for any equivalence $\sim$ between trace and bisimulation equivalences $Sys \sim \tau^l\langle Result, T\rangle$ 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* is the LTS over the alphabet $\{\langle Result, T\rangle, \langle Result, F\rangle\}$ given in Fig. 5. They are equivalent iff the simulated formula is a tautology.

4. PSPACE LOWER BOUND

Let $p_1 \cdots p_n, q_1 \cdots q_r$ be finite labeled transition systems and $\sim$ be an equivalence which lies between strong bisimulation and trace equivalences.

THEOREM 3. *The problem whether hide* $a_1, \ldots a_n$ *in* $(p_1 \| p_2 \cdots \| p_k)$ *is* $\sim$*-equivalent to hide* $b_1, \ldots b_m$ *in* $(q_1 \| q_2 \cdots \| q_r)$ *is PSPACE-hard.*

Proof. We will do a generic reduction from a polynomial space bounded Turing machine. For each deterministic Turing machine M that is space bounded by a polynomial $pol(n)$ we give a polynomial algorithm that takes as an input a string x and produces two expressions in in the form *hide* $a_1, \ldots a_n$ *in* $(p_1 \| p_2 \cdots \| p_k)$ which are equivalent iff M accepts x.

These expressions are obtained from an expression *SIM* which simulates the behavior of M on x. Hence, the central idea is to construct a short description of a system *SIM* which simulates the behavior of M on x.

Now we are going to describe *SIM*. Without loss of generality we can assume that M is a deterministic Turing machine with only one accepting state and one rejecting state; if a word is accepted (rejected), then M stops in the accepting (rejecting) state; M has a move on any tape symbol from any state, except the accepting and rejecting states (there is no move from these two states).

In our proof, for x of length n the *SIM* will have a form:

$$CONTROL_n \| CELL_1 \| CELL_2 \| \cdots \| CELL_{pol(n)}.$$

$CELL_i$ represents the contents of ith tape cell in the computation of M on x. Cells are communicating with control; there is no direct communication between cells.

Let Q, T be the sets of states and tape symbols of M. $CELL_i$ is defined as follows:

STATES. $T \cup (Q \times T) \cup \{wait\}$.

INITIAL STATE. The initial state for $CELL_i$ depends on i and it is

- (q_0, a_1), where q_0 is the initial state of M and a_1 is the first symbol of string x for $CELL_1$.
- The ith letter of x for $1 < i \leqslant |x|$.
- Blank symbols for $i > |x|$.

ALPHABET. $(\{i\} \times Q \times T) \cup (\{\bar{i}\} \times (Q \cup T))$.

TRANSITION RELATION. It is defined as follows:

- From state (q, a) it can move to state *wait* via the communication $\langle i, q, a \rangle$.
- From state a it can move to state (q, a) via the communication $\langle \bar{i}, q \rangle$.
- From state *wait* it can move to state a via the communication $\langle \bar{i}, a \rangle$.

Intuitively, if the head of M is over cell i then $CELL_i$ transmits to $CONTROL_n$ the message $\langle i, q, a \rangle$ which describes where, over what symbol and in what state is M. When $CELL_i$ receives a message $\langle \bar{i}, q \rangle$ (or $\langle \bar{i}, a \rangle$), the message indicates that M is in state q and puts the head over ith cell (or M writes a in this cell).

Let r be the number of M commands. $CONTROL_n$ is defined as follows:

STATES. *START*, *Accept*, *Reject*, *FINAL*, $\{0, 1\} \times \{1, ..., pol(n)\} \times \{1, ..., r\}$.

INITIAL STATE. *START*.

ALPHABET. $\{1, ... pol(n)\} \times Q \times T \cup (\{\bar{1}, ...\overline{pol(n)}\} \times (Q \cup T)) \cup \{accept, reject\}$; i.e., the alphabet is the union of the alphabets of the $CELL_i$ $(i = 1, ... pol(n))$ and two new actions *accept* and *reject*.

TRANSITIONS. $CONTROL_n$ has $3 \times p(n)$ transition for every command of M and a number of transitions between *START*, *Accept*, *Reject*, and *FINAL*. Below the transition relation is described:

- If $qa \to q'b$ *left* is kth command of M then for every i there are transitions:
  1. From *START* to $\langle 0, i, k \rangle$ via the communication $\langle i, q, a \rangle$.
  2. From $\langle 0, i, k \rangle$ to $\langle 1, i, k \rangle$ via the communication $\langle \bar{i}, b \rangle$.
  3. From $\langle 1, i, k \rangle$ to *START* via the communication $\langle \overline{i-1}, q' \rangle$.
- The case when the head is moving to the right is defined similarly. Just replace $i-1$ by $i+1$.
- If q is the accepting (the rejecting) state of M then for every tape symbol a, and every i there is a transition from *START* to *Accept* (to *Reject*) labeled by $\langle i, q, a \rangle$.
- From *Accept* to *FINAL* there is a transition labeled by *accept*.
- From *Reject* to *FINAL* there is a transition labeled by *reject*.

Remark. From the first clause in the definition of the transition relation of $CONTROL_n$ one can see that the kth command of M is simulated by three transitions. $CONTROL_n$ moves from state *START* to the state $\langle 0, i, k \rangle$ in which it

remembers that the head is over the ith cell and it simulates the kth command. Then it moves to the state $\langle 1, i, k\rangle$ and changes the contents of the ith cell to b. Finally, it returns to the state $START$ and puts the head with the M's state q' over cell $i-1$.

Let us explain in what sense the system SIM defined as $CONTROL_n \parallel CELL_1 \parallel CELL_2 \parallel \cdots \parallel CELL_{pol(n)}$ simulates the behavior of the Turing machine M on input x.

An instantaneous description (id) of M for an input of length n is a sequence $\langle q, \ , i, a_1, \ldots a_{pol(n)}\rangle$. Here, q is a state of M, $a_1, \ldots a_{pol(n)}$ are tape symbols that represent the current contents of the first $pol(n)$ tape cells and i is the location of the head. Recall that the space complexity of M is bounded by pol, therefore, $1 < i < pol(n)$.

To the above instantaneous description corresponds the state:

$$\langle START, a_1, \ldots a_{i-1}, (q, a_i), a_{i+1}, \ldots, a_{pol(n)}\rangle \text{ of } SIM.$$

Recall that M is deterministic and for every non-final state q and for every tape symbol a there is a unique command of M of the form $qa \to q'bD$, where q' is a state of M, b is a tape symbol and $D \in \{left, right\}$ is the direction of a move.

Let $id = \langle q, i, a_1, \ldots a_{pol(n)}\rangle$ be an instantaneous description. Assume that q is not a final state and that $qa_i \to q'bleft$ is kth command of M. Then in one step M can move from id to the instantaneous description $id' = \langle q', i-1, a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_{pol(n)}\rangle$. This step of M will be simulated by the following three transitions of SIM:

1. From state $\langle START, a_1, \ldots a_{i-1}, (q, a_i), a_{i+1}, \ldots, a_{pol(n)}\rangle$ to state $\langle (0, i, k), a_1, \ldots a_{i-1}, wait, \ , a_{i+1}, \ldots, a_{pol(n)}\rangle$ via the communication $\langle i, q, a_i\rangle$.

2. From state $\langle (0, i, k), a_1, \ldots a_{i-1}, wait, a_{i+1}, \ldots, a_{pol(n)}\rangle$ to state $\langle (1, i, k), a_1, \ldots a_{i-1}, b, a_{i+1}, \ldots, a_{pol(n)}\rangle$ via the communication $\langle \bar{i}, b\rangle$.

3. From state $\langle (1, i, k), a_1, \ldots a_{i-1}, b, a_{i+1}, \ldots, a_{pol(n)}\rangle$ to state $\langle START, a_1, \ldots a_{i-2}, (q', a_{i-1}), b, a_{i+1}, \ldots, a_{pol(n)}\rangle$ via the communication $\langle \overline{i-1}, q'\rangle$.

The case when the head moves to the right is simulated in a similar way.

Recall that M has no move from the final states that are the accepting and the rejecting states of M.

SIM has the following transitions for the final states of M:

1. From state $\langle START, a_1, \ldots a_{i-1}, (q, a_i), a_{i+1}, \ldots, a_{pol(n)}\rangle$ it moves to state $\langle Accept, a_1, \ldots a_{i-1}, wait, a_{i+1}, \ldots, a_{pol(n)}\rangle$ via the communication $\langle i, \ , q, a\rangle$ if q is the accepting state of M.

2. From state $\langle START, a_1, \ldots a_{i-1}, (q, a_i), a_{i+1}, \ldots, a_{pol(n)}\rangle$ it moves to state $\langle Reject, a_1, \ldots a_{i-1}, wait, a_{i+1}, \ldots, a_{pol(n)}\rangle$ via the communication $\langle i, \ , q, a\rangle$ if q is the rejecting state of M.

3. From state $\langle Accept, \ldots\rangle$ it moves to state $\langle FINAL, \ldots\rangle$ via the communication $accept$.

4. From state $\langle Reject, \ldots\rangle$ it moves to state $\langle FINAL, \ldots\rangle$ via the communication $reject$.

It is easy to see that we have provided above all the transitions of *SIM* that are reachable from its initial state. Recall that M is deterministic and it always reaches the rejecting or the accepting state. Hence, from the above description of simulation of M by *SIM* it follows that the reachable part of *SIM* is a path. Moreover, if M makes s moves on the input x the length of this path is $3 \times s + 2$ and the last action of the path is *accept* (*reject*) iff M accepts (rejects) x.

This is summarized by

OBSERVATION 1. 1. *SIM* is a path; its last action is labeled either by *accept* or by *reject* and no other action is labeled by *accept* or *reject*.

2. The last action of *SIM* is labeled by *accept* iff M accepts x.

3. The last action of *SIM* is labeled by *reject* iff M rejects x.

Now, let *ACCEPT* be the automaton over the alphabet $\{accept, reject\}$ with two states and only one transition from the initial state to the second state; this transition is labeled by *accept*.

From Observation 1 follows

OBSERVATION 2. For any equivalence $\sim$ between trace and bisimulation equivalences

$$(CONTROL_n \parallel CELL_1 \parallel CELL_2 \parallel \cdots \parallel CELL_{pol(n)} \parallel ACCEPT)$$
$$\sim (CONTROL_n \parallel CELL_1 \parallel CELL_2 \parallel \cdots \parallel CELL_{pol(n)})$$

if and only if M accepts x.

The reader can easily check that the size of $CELL_i$ is independent of n, that the size of $CONTROL_n$ is $O(pol(n))$, and that our reduction is polynomial. ∎

Remarks (Some strengthenings of Theorem 3). 1. Let $T = \langle Q, q_0, A, \rightarrow \rangle$ be a labeled transition system. Let Q' be the subset of Q reachable from the initial state q_0 and let $\rightarrow'$ be the restriction of $\rightarrow$ on Q'. Let $Reach(T)$ be the labeled transition system $\langle Q', q_0, A, \rightarrow' \rangle$. We say that an equivalence $\sim$ is reasonable if (A) $\sim$ refines trace equivalence and (B) $T \sim Reach(T)$ for any T. Our proof shows that Theorem 3 holds for any reasonable equivalence.

2. A slight modification of the reduction used in our proof shows that the problem whether *hide* $a_1, \ldots a_k$ *in* $(p_1 \parallel p_2 \cdots p_n) \sim q$ is PSPACE hard for any equivalence between trace and bisimulation equivalences.

Indeed, let *LOOP* be the labeled transition system over the alphabet $\{accept, reject\}$ with only one state s and with the only one transition $s \xrightarrow{\tau} s$. Let $CONTROL'_n$ be obtained from $CONTROL_n$ by deleting the transition between the states *Accept* and *FINAL* and adding τ-transition $Accept \xrightarrow{\tau} Accept$. Let *lab* be the alphabet of $CONTROL_n$, except for the actions *accept* and *reject*.

The PSPACE lower bound follows from the observation that

$$hide\ lab\ in\ (CONTROL'_n \parallel CELL_1 \parallel CELL_2 \parallel \cdots \parallel CELL_{pol(n)}) \sim LOOP$$

if and only if M accepts x.

3. Our proof of Theorem 3 also shows that the problem whether $p_1 \parallel p_2 \cdots \parallel p_k$ is $\sim$-equivalent to $q_1 \parallel q_2 \cdots \parallel q_r$ is PSPACE-hard for any equivalence $\sim$ between trace and bisimulation equivalences. Indeed, just note that hiding is not used in Observation 2.

4. We say that an equivalence $\sim$ has *τ-abstraction property* if $c\tau p \sim cp$ for every action c and every labeled transition system p. In particular, $\tau p \sim \tau^l p$ for every $l > 0$.

All equivalences studied in the literature that ignore some of the internal behavior have the above τ-abstraction property. Among such equivalences is observational congruence [Mi] and all equivalences that take into account either fairness or divergence.

A slight modification of the reduction used in our proof shows

COROLLARY 4. *For any p and any equivalence $\sim$ that has τ-abstraction property and lies between trace and bisimulation equivalences, the problem whether hide $a_1, \ldots a_n$ in $(p_1 \parallel p_2 \cdots \parallel p_k)$ is $\sim$-equivalent to τp is PSPACE hard.*

Proof. Indeed, let p' be such that p' and p are not $\sim$-equivalent. Let $CONTROL'_n$ be obtained from $CONTROL_n$ by (1) deleting the transition *accept* between the states *Accept* and *FINAL* and adding a τ transition from the state *Accept* to p; (2) deleting the transition *reject* between the states *Reject* and *FINAL* and adding a τ transition from the state *Reject* to p'.

The PSPACE lower bound follows from τ-abstraction property and from the observation that *hide lab in* $(CONTROL'_n \parallel CELL_1 \parallel CELL_2 \parallel \cdots \parallel CELL_{pol(n)})$ is bisimulation equivalent to $\tau^l p$ (respectively, $\tau^l p'$) for some $l > 0$ iff M accepts (respectively, rejects) x. ∎

A weaker version of Corollary 4 was noted in [SHRS].

By a generalization of the proof of Corollary 4 one can also show

COROLLARY 5. *For any p and any equivalence $\sim$ between trace and bisimulation equivalences that has τ-abstraction property and is a congruence with respect to the plus operation of CCS the problem whether hide $a_1, \ldots a_n$ in $(p_1 \parallel p_2 \cdots \parallel p_k)$ is $\sim$-equivalent to p is PSPACE hard.*

Remark (PSPACE lower bound for CCS parallel composition and restriction). In Milner's Calculus of Communicating Systems [Mi], every communication action c has the complement action $\bar{c}$. An action c may synchronize only with its complement action and in this case the invisible action τ is produced.

A CCS expression $(p_1 \mid p_2 \cdots p_k) \backslash \{a_1, \ldots a_l\}$ describes a network of communicating agents $p_1, \ldots, p_k$ in which the communication actions $a_1, \ldots a_l$ and their complements are hidden. Here, $\mid$ is CCS parallel composition operator and $\backslash\{a_1, \ldots a_l\}$ is CCS restriction operator.

Below we describe what modification should be made in the proof of Theorem 3 in order to show that

The problem whether $(p_1 \mid p_2 \cdots \mid p_k)\backslash\{a_1, \ldots a_l\}$ is $\sim$-equivalent to $(q_1 \mid q_2 \cdots \mid q_r)\backslash\{b_1, \ldots b_m\}$ is PSPACE-hard for any equivalence $\sim$ between trace and bisimulation equivalences.

Let $CELL_i$ and $CONTROL_n$ be defined as in the proof of Theorem 3. Let *lab* be the alphabet of $CONTROL_n$, except for the actions *accept* and *reject*. Let $CONTROL'_n$ be obtained from $CONTROL_n$ by renaming every communication in *lab* by its complement communication (we do not rename the actions *accept* and *reject*). Let SIM' be $(CONTROL'_n \mid CELL_1 \mid \ldots \mid CELL_{pol(n)})$

Note that the reachable part of $SIM'\backslash lab$ is a path in which the last action is labeled by *accept* or *reject* and all other actions are labeled by τ.

Moreover, the last action is labeled by *accept* if and only if M accepts x. Therefore, for any equivalence $\sim$ between trace and bisimulation equivalences $(SIM'\backslash lab) \sim SIM'\backslash(lab \cup \{reject\})$ if and only if M accepts x.

This completes the proof.

5. EXPSPACE COMPLETENESS OF TRACE EQUIVALENCE

THEOREM 6. *The problem whether hide* $a_1, \ldots a_k$ *in* $(p_1 \parallel p_2 \cdots p_n)$ *is trace equivalent to hide* $b_1, \ldots b_m$ *in* $(q_1 \parallel q_2 \cdots q_r)$ *is EXPSPACE-complete.*

Proof. Meyer and Stockmeyer [MeS] have shown the EXPSPACE completeness of deciding whether the language of a regular expression with squaring over alphabet Σ is equal to Σ^*.

We will provide a polynomial time reduction from the above problem to the problem of trace equivalence of net expressions. Our proof gives $DSPACE(c^{n/\log n})$ lower bound. The upper bound of $DSPACE(O(d^n))$ is obtained in a manner similar to the proof of Theorem 13.14 in [HU].

Regular expressions with squaring may use the usual operations union, concatenation and Kleene's star as well as squaring operation $R^2 = RR$.

NOTATIONS. We use the notation $Lan(R)$ for the language defined by an expression R. For a language L we denote by $Prefix(L)$ the language which contains all the prefixes of the strings in L.

THEOREM 7 [MeS]. *Let* Σ *be an alphabet of size* >1. *There is a constant* $c>1$ *such that no deterministic Turing machine with space bound* c^n *can check whether the language of a regular expression with squaring over alphabet* Σ *is equal to* Σ^*.

Let Σ be an alphabet and s, e be two actions not in Σ. For any regular expression with squaring R over Σ, we will construct a net expression $R_{s,e}$ such that the following properties are satisfied:

Properties of the construction. 1. $size(R_{s,e}) = O(size(R))$.

2. $[\![R_{s,e}]\!]_{traces} = Prefix((\Sigma^* \, sLan(R) \, e)^*)$.

Hence $Lan(R) = \Sigma^*$ iff $[\![R_{s,e}]\!] =_{traces} [\![C_{\Sigma^*}]\!]$, where C_{Σ^*} is the transition diagram in Fig. 6.

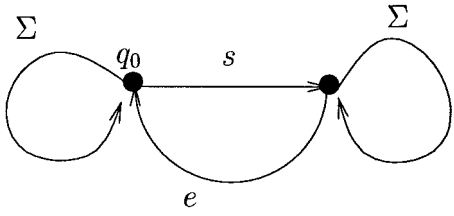

FIG. 6. C_{Σ^*}.

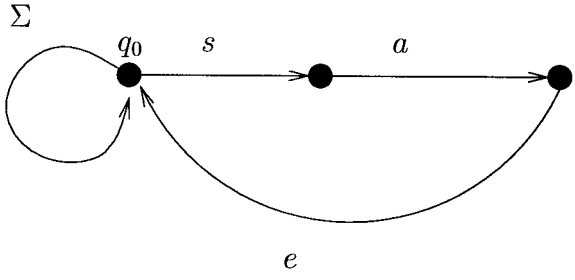

FIG. 7. *Atomic*(a, s, e).

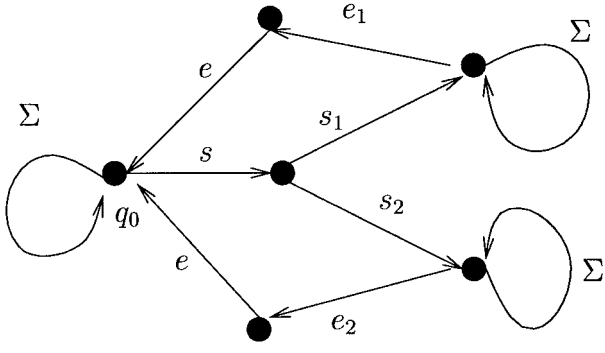

FIG. 8. *SUM*$(s, e, s_1, e_1, s_2, e_2)$.

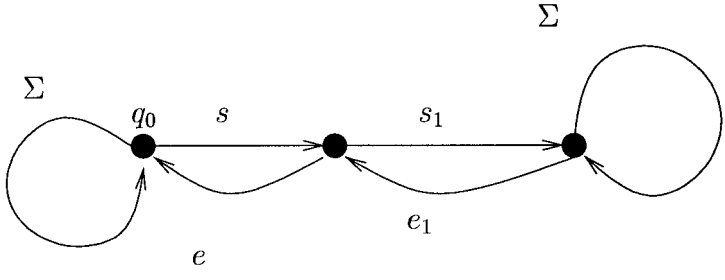

FIG. 9. *ITER*(s, e, s_1, e_1).

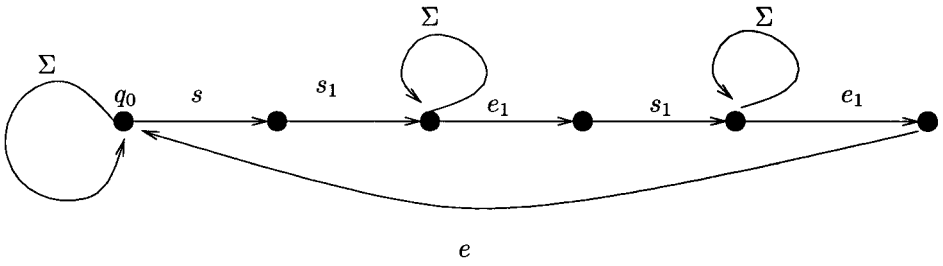

FIG. 10. $SQUARE(s, e, s_1, e_1)$

Theorem 6 will follow from Meyer–Stockmeyer's EXPSPACE-completeness theorem, Fact 1 (see Section 2.4) and the above properties. In the remainder of the section we provide a construction which satisfies properties 1 and 2. The construction is defined inductively by the the structure of regular expressions with squaring

1. $a_{s,e} \triangleq Atomic(a, s, e)$, where $Atomic(a, s, e)$ denotes the labeled transition system pictured in Fig. 7.

2. $(R_1 \cup R_2)_{s,e} \triangleq$ *hide* s_1, e_1, s_2, e_2 *in* $(SUM \parallel (R_1)_{s_1,e_1} \parallel (R_2)_{s_2,e_2})$, where s_1, e_1, s_2, e_2 are fresh actions and SUM denotes the labeled transition system in Fig. 8.

3. $(R_1; R_2)_{s,e} \triangleq =$ *hide* i *in* $((R_1)_{s,i} \parallel (R_2)_{i,e})$, where s_1, i and e_1 are fresh action.

4. $(R^*)_{s,e} \triangleq$ *hide* s_1, e_1 *in* $(ITER \parallel R_{s_1,e_1})$, where s_1 and e_1 are fresh action names and $ITER$ is the transition diagram in Fig. 9.

5. $(R^2)_{s,e} \triangleq =$ *hide* s_1, e_1 *in* $(SQUARE \parallel R_{s_1,e_1})$, where s_1 and e_1 are fresh action names and $SQUARE$ is the transition diagram in Fig. 10.

The reader can easily check that our construction satisfies the properties stated above. And this completes the proof of Theorem 6. ▌

Our proof of Theorem 6 also shows

COROLLARY 8. *The problem whether hide* $a_1, \ldots a_k$ *in* $(p_1 \parallel p_2 \cdots p_n)$ *is trace equivalent to* q *is EXPSPACE-complete.*

Let l_1 and l_2 be the lengths of the binary descriptions of R and of the normal form of $R_{s,e}$ respectively. Since the size of $R_{s,e}$ is $O(size(R))$ and $R_{s,e}$ has at most $O(size(R))$ fresh actions it follows that $l_2 = O(l_1 \log l_1)$. Hence, by Theorem 7, it follows:

COROLLARY 9. *There is a constant* $c > 1$ *such that no deterministic Turing machine with space bound* $c^{n/\log n}$ *can check whether hide* $a_1, \ldots a_k$ *in* $(p_1 \parallel p_2 \cdots p_n)$ *is trace equivalent to* q.

6. CONCLUSION AND FURTHER RESULTS

We demonstrated that the problem of equivalence of synchronously communicating systems of finite agents is PSPACE-hard for any equivalence between bisimulation and trace equivalences. As it was mentioned in the Introduction, in order to

check bisimulation equivalence between $p = hide\ a_1, \ldots a_l\ in\ (p_1 \parallel p_2 \cdots \parallel p_n)$ and $q = hide\ b_1, \ldots b_m\ in\ (q_1 \parallel q_2 \cdots \parallel q_k)$, one can first construct p and q and then apply the algorithm given by Paige and Tarjan [PT]. This procedure requires exponential time. Our conjecture is:

Conjecture. EXPTIME-lower bound can be proved for all equivalences between bisimulation and trace equivalences.

Larry Stockmeyer [Sto] proved EXPTIME lower bound for checking bisimulation and weak bisimulation equivalences.

We proved EXPSPACE-completeness for the problem of checking trace equivalence between synchronous systems of finite agents. Let us mention other equivalences for which we can prove the EXPSPACE lower bound. Bisimulation equivalence was originally defined as the limit of the sequence $\sim_0, \sim_1, \ldots \sim_k \cdots$ of successively finer equivalences (see [Mi] p. 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.[2] 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.

The results of the paper hold 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 succinctly. Indeed, in our proofs of lower bounds we used expressions of the form $hide\ a_1 \cdots a_n\ in\ (p_1, \ldots p_k)$ with the property that every communication action c occurs in at most two among $p_1, \ldots, p_k$. Any expression E that appears in the proofs of our theorems can be translated in a polynomial time to a bisimulation equivalent expression E' of the form $(q_1 \mid \ldots q_k) \backslash \{b_1, \ldots b_l\}$, where $\mid$ and $\backslash$ are CCS parallel composition and restriction operators. Moreover, the size of E' is linear in the size of E. (We provided such a translation for the expressions that appear in the proof of Theorem 3; see the last remark in Section 4. Appropriate translations for Theorems 2 and 4 are even simpler.) Hence, all our theorems hold for CCS parallel composition and restriction.

In this paper we considered only interleaving semantics and ignored partial order or "true" concurrency semantics. In [Ja] the complexity of equivalence of 1-safe labeled Petri nets under a variety of true concurrency equivalences was investigated.

ACKNOWLEDGMENTS

The author is grateful to Jan Groote, Alain Mayer, Albert R. Meyer, Larry Stockmeyer, and Boris Trakhtenbrot for fruitful discussions and comments. Many thanks are given to the anonymous referee for his helpful suggestions.

Received May 17, 1995; final manuscript received August 28, 1996

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

REFERENCES

[ABGS] Alvarez, C., Balcazar, J., Gabarro, J., and Santa, M. (1992), Parallel Complexity in the design and analysis of concurrent systems, *in* "Formal Aspects of Computing," Vol. 4, No. 6A.

[Bl] Bloom, B. (1990), "Ready Simulation, Bisimulation, and the Semantics of CCS-Like Languages," Ph.D. Thesis, Technical Report MIT/LCS/TR-491.

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

[RvG] van Glabbeek, R. (1993), The linear time -branching time spectrum, *in* "CONCUR 93," Lect. Notes in Computer Science, Vol. 715, pp. 60–81, Springer Verlag, Berlin.

[GM] Groote, J. F., and Moller, F. (1992), Verification of parallel system via decomposition, *in* "Proc. of Third Conference on Concurrency Theory," Lect. Notes in Computer Science, Vol. 630, pp. 62–76, Springer Verlag, Berlin.

[GV] Groote, J. F., and Vaandrager, F. (1990), An efficient algorithm for branching bisimulation and stuttering equivalence, *in* "International Conference on Automata, Languages and Programming," Lect. Notes in Computer Science, Vol. 443, Springer Verlag, Berlin.

[He] Hennessy, M. (1988), "Algebraic Theory of Processes," MIT Press, Cambridge, MA.

[Ho] Hoare, C., (1985), "Communicating Sequential Processes," Prentice-Hall, Englewood Cliffs, NJ.

[HU] Hopcroft, J., and Ullman, J. (1979), "Introduction to Automata Theory, Languages and Computations," Addison–Wesley, Reading, MA.

[Ja] Jategaonkar, L. (1993), "Observing "True" Concurrency," Ph.D. Thesis, Technical Report MIT/LCS/TR-618.

[MS] Mayer, A. J., and Stockmeyer, L. J. (1994), The complexity of world problems, this time with interleaving, *Inform. and Comput.* **115**, 293–311.

[MeS] Meyer, A. R., and Stockmeyer, L. J. (1972), The equivalence problem for regular expressions with squaring requires exponential space, *in* "Proc. 13th IEEE Symp. on Switching and Automata Theory," pp. 125–129.

[Mi] Milner, R. (1989), "Communication and Concurrency," Prentice-Hall, Englewood Cliffs, NJ.

[KS] Kanellakis, P. C., and Smolka, S. A. (1990), CCS expressions, finite state processes, and three problems of equivalence, *Inform. and Comput.* **86**.

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

[Pa] Park, D., (1981), "Concurrency and Automata on Infinite Sequences," Lect. Notes in Computer Science, Vol. 104, Springer Verlag, Berlin.

[Plo] Plotkin, G. (1981), "A Structured Approach to Operational Semantics," FN 19 DAIMI, Aarhus Univ.

[Ra] Rabinovich, A. (1992), Checking equivalences between concurrent systems of finite agents, *in* "International Conference on Automata, Languages and Programming," Lect. Notes in Computer Science, Vol. 623, pp. 696–707, Springer Verlag, Berlin.

[SHRS] Shulka, S. K., Hunt, H. B., Rosenkratz, D. J., and Stearn, R. E. (1996), "On the Complexity of Relational Problems for Finite State Processes," Lect. Notes in Computer Science, Vol. 1099, pp. 466–477, Springer-Verlag, Berlin.

[SM] Stockmeyer, L. J., and Meyer, A. R. (1973), Word problems requiring exponential time, *in* "Proc. 5th ACM Symp. on Theory of Computing," pp. 1–9.

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