Parallel complexity in the design and analysis of concurrent systems

Carme Álvarez
José L. Balcázar
Joaquim Gabarró
Miklos Santha

Report LSI–90–38

Parallel complexity in the design and analysis of concurrent systems

C. Àlvarez* J.L. Balcázar* J. Gabarró* M. Santha+

Dep. de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Pau Gargallo 5, 08028 Barcelona
Spain

CNRS - LRI
Université Paris-Sud
91405 Orsay
France

Keywords: Petri nets; partially commutative monoids; CCS; PRAM algorithms; boolean circuits; P-completeness.

Abstract: We study the parallel complexity of three problems on concurrency: decision of firing sequences for Petri nets, trace equivalence for partially commutative monoids, and strong bisimilarity in finite transition systems. We show that the first two problems can be efficiently parallelized, allowing logarithmic time Parallel RAM algorithms and even constant time unbounded fan-in circuits with threshold gates. However, lower bounds imply that they cannot be solved in constant time by a PRAM algorithm. On the other hand, strong bisimilarity in finite labelled transition systems can be classified as P-complete; as a consequence, algorithms for automated analysis of finite state systems based on bisimulation seem to be inherently sequential in the following sense: the design of an efficient parallel algorithm to solve any of these problems will require an exceedingly hard algorithmic breakthrough.

1. Introduction

Given the intrinsic difficulty of designing large software systems, it is natural that software tools would be designed to help in performing this task. The possibility of formalizing both specifications and implementations in the same, or in a closely related, formal language yields the potential of automated analysis, allowing for early checking of correctness and provably correct prototypes.

The design of correct concurrent programs is even more difficult than in the sequential case, and their verification using formal systems may give rise to formidable computational problems. For instance, the study of the correctness and liveness properties of mutual exclusion algorithms for just two processes already requires resorting to computerized analysis [27]; if more processes are considered, the state space soon becomes intractable.

* Research supported by the ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (project ALCOM).

+ Research supported by the Programme MERCURE of the DCSTD of the Ministère Français des Affaires Etrangères and the DGICYT of the Ministerio de Educación y Ciencia de España. This research was performed while visiting the Dep. de Llenguatges i Sistemes Informàtics of the Universitat Politècnica de Catalunya.

One reason to develop concurrent programs stems from the fact that important advantages can be gained from the use of massive parallelism. In view of the large number of parallel algorithms discovered in recent years (see [16] and [10]), it might be hoped that one such application would be the study of concurrent systems, and that algorithms running on highly parallel machines could perform automated analysis of large concurrent programs substantially faster than sequential algorithms. Such a behaviour corresponds to a running time roughly logarithmic in the size of the state space (assumed finite); and being able to tackle problems of relevant size corresponds to algorithms that use a large but feasible number of processors (cf. the definition of the class NC below).

One of the first models issued to study concurrent systems was the Petri net model. A Petri net consists of two different kinds of objects: places and transitions. Places serve to model pre and post conditions and transitions model events. A transition needs to satisfy some conditions to be fired, and its firing changes the valuations on the places (see below for exact definitions). The net evolves firing transitions sequentially and the behaviour of the whole system is described by the set of all possible firing sequences.

For each fixed Petri net, we exhibit an NC algorithm to decide very efficiently in parallel whether a given sequence of transitions is a firing sequence. We also discuss some lower bounds on the parallel time necessary to solve this problem.

In a monoprocessor environment, a concurrent system is fully described by the set of all sequential evolutions. A possible evolution of the system is described by a finite word called trace. Let us consider with more attention the sequencing of two events x and y in a trace $w = \cdots xy \cdots$. Let $w' = \cdots yx \cdots$ be the trace obtained from w by commuting the order of the events x and y. We have two different possibilities:

- The events x, y are independent from each other. In this case the order of execution is irrelevant and x, y commute. Then the traces w and w' correspond to the same parallel behaviour.
- The execution of x modifies the environment of y. Then these events are in conflict. The trace w' represents a behaviour different from w.

A basic question is: given two traces, do they model the same concurrent behaviour? A way to deal with this approach is to consider partially commutative monoids. This framework has been fully developed by Mazurkiewicz [18]. A mathematical characterization of trace equivalence was found in [8], and it can be used to find a fast sequential algorithm. Here we prove the existence of an NC algorithm, and discuss also some lower bounds.

A capability that seems natural to expect from software tools for aiding the design of concurrent systems is to be able to decide some form of equivalence of finite state systems. Indeed, this problem plays a fundamental role in the study of concurrent systems, and has been widely studied both from theoretical and practical points of view. Milner specifies in [21] a complete set of axioms for proving equivalence of finite state agents. Kanellakis and Smolka consider in [15] efficient sequential algorithms to solve this problem. On the more practical side, the prototype named Concurrency Workbench, implemented in Standard ML, has been used by Walker [27] for undertaking the automated analysis of mutual exclusion algorithms via finite state systems, using the fact that the state space of all these algorithms is finite.

Until now, the analysis of concurrent systems by means of bisimulation techniques has been based on sequential algorithms. A natural question to ask is: do the automatic bisimulation techniques admit fast parallel algorithms?

In this paper we give a strong evidence that unfortunately the answer to the above question is negative. More precisely, we prove that deciding bisimulation in finite transition systems is a P-complete problem. P-complete problems have efficient sequential algorithms but it is widely believed that they do not admit fast parallel ones.

In fact, this concept plays a role analogous to the notion of NP-complete problems. These are problems that can be solved by an exponentially slow exhaustive search, and they inherently seem to require superpolynomial time algorithms. The NP-completeness of a problem implies that success in designing a polynomial time sequential algorithm for it is highly unlikely.

Analogously, P-complete problems are identified as inherently sequential problems: if there are any problems in P that do not admit efficient parallel algorithms, then all P-complete problems are among them. Conversely stated, if a parallel algorithm is found for a P-complete problem which uses a feasible (e.g. polynomial) amount of hardware and runs in polylogarithmic time, then all problems solvable in polynomial time have also such feasible and very fast parallel algorithms. However, strong research in the area during several years has failed to produce such an algorithm for any of the well studied P-complete problems. Thus, the design of a parallel algorithm with these characteristics for a P-complete problem would require a breakthrough in Algorithmics. Actually the conjecture of many researchers in the field is that such an algorithm does not exist at all. Surveys of P-complete problems have appeared in [22] and [13].

2. Preliminaries

2.1 Sequential and parallel complexity classes. For the formal study of the possible existence of parallel algorithms we will consider two main complexity classes: P and NC. We mention also some interesting subclasses of NC. The class P models problems with *efficient sequential algorithms*; the class NC models problems with *fast parallel algorithms*, using a feasible number of processors. Each of these classes has many characterizations that support this description.

- By definition, the class P contains the problems for which a polynomial time sequential algorithm exists. This can be formalized by considering an abstract model of sequential computation for which "time" is a well-defined notion. Polynomial time RAM algorithms (a model quite close to a real computer [1]), polynomial time Turing machines ([1], [4]), or even polynomial size uniform circuits (see below) are all suitable for this purpose, and give equivalent definitions of the class P.

- The class NC formalizes the concept of efficiently parallelizable problems: it contains those problems for which a parallel algorithm can be designed which runs in polylogarithmic parallel time and uses a feasible (i.e. polynomial) amount of hardware. There are many characterizations of this class. Consider for instance Parallel RAM (PRAM) machines, which are one of the basic abstract models of parallel computers [26]. NC can be

defined as the class of all the problems that can be solved in a PRAM within $O(\log^k n)$ time for constant k and using polynomially many processors.

For theoretical analysis sometimes *unbounded fan-in boolean circuits* are preferable [7]. A boolean circuit is a directed, acyclic, labelled graph in which the nodes of indegree zero are the inputs, the nodes of indegree 1 compute boolean negation, and the nodes of indegree 2 or more compute either boolean conjunction or disjunction of all their inputs, according to their respective label. The nodes of outdegree zero are the output nodes. The *size* of a circuit is the number of its nodes; the *depth* is the length of the longest path from an input to an output. The nodes in a circuit are called also *gates*. Binary inputs and outputs might be binary encodings of other objects assuming some simple coding scheme.

To use boolean circuits to solve problems, we have to select a different circuit for each input length; but such a selection might be very hard to compute. Here we will explicitly rule out those families of circuits for which this selection is indeed hard, and will restrict ourselves to uniform families. A family of circuits is *uniform* if basic facts about the connection of the gates can be answered in deterministic logarithmic time, or equivalently can be expressed in an extended version of first order logic (see [6] for precise definitions).

It is well known that in many aspects PRAMs and uniform unbounded fan-in circuits are equivalent [26], with bounds on number of processors corresponding to bounds on the size of the circuit, and bounds on the PRAM time corresponding to bounds on the depth of the circuit. Thus NC is formed by the problems solvable by polylogarithmic depth, polynomial size uniform circuits.

NC has some interesting subclasses. In particular, AC^0 contains the problems solvable by unbounded fan-in uniform circuits of constant depth and polynomial size, or equivalently solvable by a PRAM in constant time with a feasible (i.e. polynomial) number of processors; and AC^1 contains the problems solved by unbounded fan-in uniform circuits of logarithmic depth and polynomial size, which corresponds to logarithmic time in a PRAM with again a feasible number of processors. AC^0 contains some problems with long history, for instance the addition of two integer numbers.

Lying between AC^0 and AC^1 is the class TC^0, defined by uniform constant depth polynomial size circuits which are allowed to use threshold gates. This class can be motivated by the growing of a complexity theory of neural networks [23], and is important for tight analysis of the complexity of certain problems; it also contains very natural and interesting problems such as the multiplication of two integer numbers [7].

Since threshold gates can simulate AND and OR gates we have that $AC^0 \subset TC^0$, but these two classes do not coincide: Ajtai [2] and Furst, Saxe and Sipser [9] proved that the inclusion was strict. This was shown by proving that the majority problem, coded as the set $MAJ = \{w \in (0+1)^* \mid |w|_1 \geq |w|_0\}$, cannot be solved by a constant depth polynomial size circuit having only AND and OR gates. The proof does not require any uniformity condition on the AC^0 circuits.

To compare and classify problems in P we use the *constant depth reducibility* [7]. A function f is constant depth reducible to g, denoted here as $f \leq_{cd} g$, if there is a family of circuits which compute f with polynomial size, constant depth, and oracle gates for g. The cost and depth of an oracle gate is 1. It can be easily shown that AC^0, TC^0, AC^1,

and AC are closed under this reducibility; e.g., if $g \in TC^0$ and $f \leq_{cd} g$, then $f \in TC^0$.

A problem S is *P-complete* under $\leq_{cd}$ reductions if $S \in P$ and every problem in P is $\leq_{cd}$-reducible to S. It can be shown that this reducibility is transitive, and therefore to prove that a problem in P is P-complete, it is enough to prove that some other complete problem in P is reducible to it. There are several standard P-complete problems which are natural candidates for the reduction. One of these is the Circuit Value Problem CVP. The input to this problem is pair formed by a circuit and an input to the circuit. The problem consists of computing the output of the circuit on the given input. When suitable additional hypotheses are assumed on the given circuit, we obtain variants of this problem that still are P-complete. In order to prove our results we consider one of these variants: the evaluation problem for *monotone alternating circuits*. Figure 1 gives us an example of such a circuit. The following is known [13]:

Theorem 1: The Monotone Alternating Circuit Value Problem $MACVP$ is P-complete.

Input: An encoding of a monotone alternating circuit c with one output, together with boolean input values $x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n$.

Output: The value of c on these input values.

Let us end here our complexity-theoretic notions and go on to introduce the problems whose complexity will be classified. The notations introduced here will be necessary for later description of parallel algorithms.

2.2 Petri nets. The Petri net model was one of the first models introduced to describe concurrent processes with distributed control [25]. Formally a Petri net is a tuple $N = \langle P, T, F, M_0 \rangle$ where:

1. The set $P = \{p_1, \ldots, p_r\}$ is called the *set of places*. During the evolution of the net, a place p contains a number of tokens denoted as $M(p)$ and called its marking. Such a marking models some local aspect of the system with global state $M = (M(p_1), \ldots M(p_r))$.
2. The *set of transitions* is $T = \{t_1, \ldots, t_s\}$. Transitions model the events of N and every sequential behaviour is represented by a word $w \in T^*$.
3. The *flow function* $F : \{(P \times T) \cup (T \times P)\} \rightarrow \mathbb{N}$ connects between them places and transitions. The value of F fixes the precondition to be fulfilled in order to fire a transition t in a marking M. The *firing rule* is:

$$\forall p \in P : M(p) \geq F(p,t)$$

   Additionally, F gives us the new marking M' reached after the firing of t in M, denoted as $M[t\rangle M'$, and defined by:

$$\forall p \in P : M'(p) = M(p) - F(p,t) + F(t,p)$$

4. $M_0 : P \rightarrow \mathbb{N}$ is the initial marking.

We denote by $\Delta(p,t)$ the variation on the number of tokens in a place p when t is fired, $\Delta(p,t) = F(t,p) - F(p,t)$. Then $M'(p) = M(p) + \Delta(p,t)$. The firing rule can be extended

from transitions to words $w \in T^*$ as usual and the whole sequential behaviour of the net N is described by the set of *firing sequences* which is:

$$S_N(M_0) = \{w \in T^* \mid \exists M : M_0[w\rangle M\}$$

Our first main result in the next section will classify the problem for Petri nets defined as follows:

Problem 2: Fixed a Petri net $N = \langle P, T, F, M_0\rangle$, the membership problem for firing sequences on this net N, denoted as *N-PETRI-FIRING* is:

Input: $w \in T^*$

Question: $w \in S_N(M_0)$?

2.3 Partially commutative monoids. Another way to model concurrent systems is with concurrent alphabets and partially commutative monoids [18]. We call ***concurrent alphabet*** a pair $(\Sigma, \sim)$ where $\Sigma = \{x_1, \ldots x_s\}$ is a finite alphabet denoting the set of events and $\sim$ is a symmetric and irreflexive binary relation on Σ called the ***commutation relation.*** The complementary notion is also useful: the *conflict relation* is defined as $\Sigma \times \Sigma \setminus \sim$. To describe equivalent behaviours in Σ^* we introduce the congruence generated by the commutation relations (i.e. if x and y commute we consider the relation $xy \sim yx$) and we denote as $w \overset{*}{\sim} w'$ the equivalence given by this congruence. The quotient monoid $\Sigma^*/\sim$ is called *partially commutative monoid* and its elements are called *traces.* If w and w' are equivalent then they model two sequential evolutions corresponding to a unique parallel behaviour. To study this equivalence we need the projection function over a subset Δ of Σ denoted as $\Pi_\Delta : \Sigma^* \longrightarrow \Delta^*$. This function is defined as $\Pi_\Delta(x) = x$ if $x \in \Delta$ and $\Pi_\Delta(x) = \lambda$ otherwise. The trace equivalence $w \overset{*}{\sim} w'$ has been characterized in [8] in the following way:

1. for every event x in Σ we have $\Pi_{\{x\}}(w) = \Pi_{\{x\}}(w')$ and
2. for every pair (x, y) of different events in conflict we have $\Pi_{\{x,y\}}(w) = \Pi_{\{x,y\}}(w')$.

In our main results we will consider the following problem:

Problem 3: Fixed a concurrent alphabet $(\Sigma, \sim)$, the trace equivalence problem, denoted as $(\Sigma, \sim)$-*TRACE-EQUIVALENCE*, is

Input: A string $w\$w'$ where $w, w' \in \Sigma^*$ and $\$ \notin \Sigma$.

Question: It is true that $w \overset{*}{\sim} w'$?

2.4 Finite transition systems. Concurrent systems can be analyzed also by means of transition systems [17]. Recall that a *finite labelled transition system* (FLTS for short) is a triple $M = \langle Q, \Sigma, T\rangle$, where Q is a finite set of states (or processes), Σ is a finite alphabet of actions and $T \subseteq Q \times \Sigma \times Q$ is the set of transitions. A transition $(q, x, q') \in T$ has label x and is denoted by $q \xrightarrow{x} q'$. Given two states p and q, the idea of having the same behaviour is formalized by the notion of strong bisimulation [24] (see also [20]).

A relation $S \subseteq Q \times Q$ is a *strong bisimulation* if $(p, q) \in S$ implies, for all $x \in \Sigma$, the following bisimilarity conditions:

(i) whenever $p \xrightarrow{x} p'$, then for some q', $q \xrightarrow{x} q'$ and $(p', q') \in S$,

(ii) whenever $q \xrightarrow{x} q'$, then for some p', $p \xrightarrow{x} p'$ and $(p', q') \in S$.

The *strong bisimilarity* relation $\sim$ is defined as the union of all strong bisimulations, that is

$$\sim \; = \bigcup \{S \mid S \text{ is a strong bisimulation } \}$$

Notice that the strong bisimilarity relation is also a strong bisimulation.

Other relationships such as bisimulation and observational equivalence can be defined in similar ways, using "invisible actions" [20]. It is not difficult to see that the decisional problems for these notions are equivalent to the decision of strong bisimulations.

We will prove the P-completeness of the the following problem:

Problem 4: The problem *STRONG-BISIMILARITY* is

Input: An encoding of a finite transition system with two selected states p^* and q^*.

Question: Are p^* and q^* strongly bisimilar?

3. Main results

3.1 Petri net firing. Fixed a Petri net $N = \langle P, T, F, M_0 \rangle$ we would like to study the complexity of *N-PETRI-FIRING* problem. We start with an intuitive massivelly parallel algorithm able to solve this problem. After, we will consider some tight bounds.

Proposition 5: Given a Petri net N, the decision problem *N-PETRI-FIRING* belongs to NC.

Proof. Given a Petri net N and a sequence of transitions $w = x_1 \dots x_i \dots x_n$ it is easy to prove that w is a firing sequence iff the following holds:

- To fire the transition x_1 the following property has to be satisfied:

$$\forall p \in P : M_0(p) \geq F(p, x_1)$$

- To fire the transition x_i $(1 < i \leq n)$ it is necessary to fulfil two conditions. First the prefix $x_1 \dots x_{i-1}$ is a firing sequence. And second, all the places have to contain enough tokens to enable x_i. Both conditions can be expressed together as

$$\forall 1 < i \leq n \quad \forall p \in P : M_0(p) + \sum_{t \in T} \Delta(p, t) \cdot |x_1 \dots x_{i-1}|_t \geq F(p, x_i)$$

These conditions can be easily verified in parallel. To do this we associate a processor to every transition x_i of the input string. The processor i will operate fundamentally with transition x_i. The NC program solving this problem is given in the program "N-Petri-Firing". ∎

To obtain a tight upper bound we can express the *N-PETRI-FIRING* problem in terms of first order logic enlarged with majority quantifiers [3]. Considering Immerman's work [14],

for $1 \leq i \leq n$ **do in parallel**
  for $1 \leq j \leq s$ **do**
    (* by prefix sum techniques processor i compute $count_i[j]$ *)
    $count_i[j] := |x_1 \ldots x_i|_{t_j}$
  end for;
  for $1 \leq k \leq r$ **do**
    $delta_i[k] := count_i[1] \cdot \Delta(p_k, t_1) + \ldots + count_i[s] \cdot \Delta(p_k, t_s)$;
  end for;
  if $i = 1$
    then $enabled_i := \bigwedge_{1 \leq k \leq r} (M_0[k] \geq F(p_k, x_i))$
    else $enabled_i := \bigwedge_{1 \leq k \leq r} (M_0[k] + delta_{i-1}[k] \geq F(p_k, x_i))$
  end if;
end parallel for;
(* by recursive folding all the processors help to compute the result *)
$N\text{-}Petri\text{-}Firing := \bigwedge_{1 \leq i \leq n} enabled_i$

Program. N-Petri-Firing

this formalism can be transformed into parallel programs running over PRAM machines enlarged with threshold operations. In our case these programs have constant time. We also give a lower bound by showing that *N-PETRI-FIRING* problem is equivalent to the *MAJ* problem under constant depth reductions. Hence this problem cannot be solved in constant time by a standard PRAM with a polynomial number of processors. For a detailed proof of the following proposition see [3].

Proposition 6: The *N-PETRI-FIRING* problem belongs to TC^0. Moreover, a lower bound complexity is fixed by the following two assertions:

1. Fixed a Petri net N we have $N\text{-}PETRI\text{-}FIRING \leq_{cd} MAJ$.
2. There exists a Petri net N such that $MAJ \leq_{cd} N\text{-}PETRI\text{-}FIRING$

3.2 Trace equivalence. Fixed a concurrent alphabet $(\Sigma, \sim)$ we would like to study the complexity of the $(\Sigma, \sim)$-*TRACE-EQUIVALENCE* problem. As we have done above, first we will propose an intuitive massively parallel algorithm to solve this problem, and second

we will consider some tight bounds.

Proposition 7: Given a concurrent alphabet $(\Sigma, \sim)$, the decision problem $(\Sigma, \sim)$-*TRACE-EQUIVALENCE* belongs to *NC*.

Proof. Given w and w' it is easy to prove that $w \overset{*}{\sim} w'$ iff the two conditions given by [8] are satisfied. These conditions can be verified in parallel.

For the first condition, i.e., every letter x of the alphabet Σ appears in w and in w' the same number of times, we use masking and prefix sum techniques as we propose in the program "Equal-Length".

---

equal-length := TRUE;
for $x \in \Sigma$ **do**
  (* by masking and prefix sum techniques compute *)
  $l_1 := |\Pi_x(w)|$;
  $l_2 := |\Pi_x(w')|$;
  equal-length := *equal-length* $\wedge (l_1 = l_2)$;
end do

Program Equal-Length

And for the second one, the letters of every pair in conflict appear in w and in w' following the same order, is verified also using masking and prefix sum techniques and as many processors as $\max\{|w|, |w'|\}$. For every pair (x, y) in conflict, the processor i verifies that the i^{th} letter of $\Pi_{\{x,y\}}(w)$ is equal to $\Pi_{\{x,y\}}(w')$. The program "Equal-Conflicts" verifies this condition. ∎

We can obtain a tight upper bound on the complexity of $(\Sigma, \sim)$-*TRACE-EQUIVALENCE* by expressing it in terms of first order logic enlarged with majority quantifiers. We also have a lower bound of this problem. It can be seen that $(\Sigma, \sim)$-*TRACE-EQUIVALENCE* cannot be solved by a PRAM in constant time because it is equivalent to the MAJ problem under a constant depth reduction which increases the computation time only with a constant. For a detailed proof of the following proposition see [3].

Proposition 8: The $(\Sigma, \sim)$-*TRACE-EQUIVALENCE* problem belongs to TC^0. Moreover, a lower bound complexity is fixed by the following two assertions:

1. Fixed a concurrent alphabet $(\Sigma, \sim)$ we have

$$(\Sigma, \sim)\text{-}TRACE\text{-}EQUIVALENCE \leq MAJ$$

2. There exists a concurrent alphabet $(\Sigma, \sim)$ such that

$$MAJ \leq (\Sigma, \sim)\text{-}TRACE\text{-}EQUIVALENCE$$

equal-conflicts := TRUE;
for every pair (x,y) in conflict **do**
(* by masking and prefix sum techniques compute *)
$k_1 := |\Pi_{\{x,y\}}(w)|; k_2 := |\Pi_{\{x,y\}}(w')|;$
if $k_1 = k_2$
then
for $1 \leq i \leq k_1$ **do in parallel**
$u :=$ letter i^{th} of $\Pi_{\{x,y\}}(w); v :=$ letter i^{th} of $\Pi_{\{x,y\}}(w');$
$test_i := (u = v)$
end parallel for;
(* by prefix sum techniques *)
$equal\text{-}conflicts := equal\text{-}conflicts \wedge \bigwedge_{1 \leq i \leq k_1} test_i$
else
equal-conflicts := FALSE
end if
end for

Program Equal-Conflicts

3.3 Bisimulations. In contrast with these problems allowing very fast and feasible parallel algorithms we prove next that the *STRONG-BISIMILARITY* problem is P-complete

It is well known that strong bisimilarity in a finite labelled transition system (FLTS) is a polynomial time decidable property [20]. To see this, it suffices to construct $\sim$ as intersection of the sequence of relations $\equiv_0, \equiv_1, \ldots,$ which are defined by induction as follows:

(i) For every $(p,q) \in Q \times Q$, $p \equiv_0 q$,

(ii) $p \equiv_{i+1} q$ if for every $x \in \Sigma$,

whenever $p \xrightarrow{x} p'$, then for some q', $q \xrightarrow{x} q'$ and $p' \equiv_i q'$;

whenever $q \xrightarrow{x} q'$, then for some p', $p \xrightarrow{x} p'$ and $p' \equiv_i q'$.

It is easy to see that these relations can be constructed in polynomial time. This is because $\sim$ coincides with $\equiv_k$ where k is the number of states in the finite transition system. More efficient algorithms to solve this problem have been considered in [15]. The P completeness of the *STRONG-BISIMILARITY* problem will follow from the following lemma. A more detailed proof appears in [5].

Lemma 9: *MACVP* can be reduced to *STRONG-BISIMILARITY*.

Proof. We will transform an arbitrary instance of the circuit value problem for monotone alternating circuits *MACVP* into an instance of *STRONG-BISIMILARITY* in three steps.

• We define the k-alternating pattern A_k. Figure 2 shows A_4. This is a circuit of height k, where every level has two gates, one valuated to 0 and other valuated to 1. It is easy to check then that in A_k the following two conditions are satisfied:

- every OR gate has an input valuated to 0,
- every AND gate has an input valuated to 1.

• We couple the k-alternating pattern A_k with the circuit C to get a new circuit C'. Figure 3 shows the circuit C' constructed from the example of Figure 1. The circuit C' satisfies the following three properties:

- every OR gate has at least an input valuated to 0,
- every AND gate has at least an input valuated to 1,
- every gate of C' evaluates to the same value as the corresponding gate in A_k or C.

• We now transform the circuit C' into a FLTS M over a one letter alphabet. M contains a state corresponding to each gate of C'. These states are called ***ordinary states***. In addition M contains $n+1$ ***auxiliary states***, associated with the $n+1$ inputs of C' which evaluate to 1 (the n inputs of C of value 1, and the constant 1 input of A_k). We say that these auxiliary states are on level 0. Figure 4 shows M in our example.

By induction it can be shown that the circuit C evaluates to 1 with the given input values if and only if states p^* and q^* in M are strongly bisimilar. ∎

As a consequence of the precedent lemma we obtain our announced result.

Theorem 10: The *STRONG-BISIMILARITY* problem is P-complete.

Other results related to this one can be obtained by the same proof idea. The properties named Observation Equivalence and Observation Congruence are defined in [19]. We can state:

Theorem 11: The problem of deciding Observation Equivalence and the problem of deciding Observation Congruence of two states in a LFTS are both P-complete.

4. Extensions and open questions

We have presented a quite precise classification of three problems on concurrency. These were the decision of firing sequences for Petri nets, the trace equivalence for partially commutative monoids, and the strong bisimulation decision problem for finite transition systems. These classifications give us hints about the complexity of massively parallel algorithms to solve them: the first two have such algorithms but however the third one cannot have such an algorithm unless all problems in P do.

Now we want to complete the discussion by raising some questions. For the bisimulation problem, our version of the statement requires the system to be part of the input. This is necessary since, the system being finite, if we fix it then we obtain only a finite number of possible pairs of states, and therefore the problem can be solved in constant time.

On the contrary, in our first two problems the devices (i.e. Petri nets and concurrent alphabets) are independent of the input, and the proofs rely strongly of this fact. It is interesting to see what happens when the description of the device is added as a part of the input. For comparison, recall that context-free parsing can be done efficiently in parallel [16]; however when a coding of the grammar is added as a part of the input the complexity grows up substantially, becoming P-complete [11]. Let us briefly describe the properties of our first two problems assuming that the devices are part of the input. No proofs will be provided here.

• Let us consider first the problem of general trace equivalence for partially commutative monoids. Inputs are a concurrent alphabet $(\Sigma, \sim)$, and words w and w' in Σ^*; the problem is to decide whether w and w' are equivalent. Using more complex arguments, we can prove that this problem is in TC^0, and thus has the same complexity as the problem for fixed monoid; it therefore can be solved by fast parallel algorithms.

• In the same way we can consider a more general version of membership for Petri net firing sequences, where the net is a part of the input. Here a problem arises. Our way of computing the variation of tokens in each place due to a prefix of the trace relies heavily on the fact that the net is fixed, and therefore the number of places and the number of transitions are constants. Thus we can compute in constant time the quantities $delta_i[k]$ as indicated in the program N-Petri-Firing. But if the net is part of the input, these expressions are sums of nonconstant numbers, and we must resort to a multiple addition. It is well known that this problem belongs to a non-uniform version of TC^0 [7]. Thus we obtain that the problem belongs to non-uniform TC^0. It seems hard to prove that this problem is in uniform TC^0.

• Finally, let us present some additional considerations regarding the bisimulation problem. Since it is so relevant to the design of concurrent systems, our negative P-completeness result calls for new concepts of equivalence that might be of practical value, yet testable by fast parallel algorithms. On the other hand, from the standpoint of a developer of a concurrent system, another relevant issue is whether interaction with a software tool might be more efficient than completely automatic equivalence testing. It is well known from the study of NP problems that in many cases "verifying" is easier than "computing". This is also true in our case; indeed, the problem of whether a given relation is a bisimulation is in NC. This opens a possible way to partially overcome the P-completeness obstacle. The idea would be to design concurrent systems in an interactive way through a sequence of stepwise refinements, e.g. in the line of [12], in such a way that at every step the designer keeps direct intuition of how to transform the precedent bisimulation to obtain a new one. He then can guess the result and verify it. Perhaps only in some rare cases the designer will need to compute the whole bisimulation, and if this case is infrequent enough he would accept such a long computational process.

References

[1] Aho, A., Hopcroft, J., Ullman, J.: *The Design and Analysis of Computer Algorithms.* Addison-Wesley (1975).

[2] Ajtai, M.: Σ^1_1-formulae on finite structures. *Ann. Pure Appl. Logic* **24**, 1–48 (1983).

[3] Àlvarez, C., Gabarró, J.: The parallel complexity of two problems on concurrency. Report LSI–89–22, Universitat Politècnica de Catalunya. Submitted for publication.

[4] Balcázar, J.L., Díaz, J., Gabarró, J.: *Structural Complexity I.* Springer Verlag EATCS Monographs in Theoretical Computer Science, v. 11 (1988).

[5] Balcázar, J.L., Gabarró, J., Santha, M.: Deciding bisimilarity is P-complete. Report LSI–90–25, Universitat Politècnica de Catalunya. Submitted for publication.

[6] Barrington, D.M., Immerman, N., Straubing, H.: On uniformity within NC^1. In: *Proc. Third Structure in Complexity Theory Conference*, 47–59 (1988).

[7] Chandra, A.K., Stockmeyer, L., Vishkin, U.: Constant depth reducibility. *SIAM J. Comput.* **13**, 2, 423–439 (1984).

[8] Cori, R., Perrin, D.: Automates et commutations partielles. *RAIRO Inf. Theor.* **19**, 21–31 (1985).

[9] Furst, M., Saxe, J.B., Sipser, M.: Parity, circuits and the polynomial time hierarchy. *Math. Syst. Theory* **17**, 13–27 (1984).

[10] Gibbons, A., Rytter, W.: *Efficient Parallel Algorithms.* Cambridge University Press (1988).

[11] Goldschlager, L.: ϵ-Productions in context-free grammars. *Acta Informatica* **16**, 303–308 (1981).

[12] He Jifeng: Process Simulation and Refinement. *Formal Aspects of Computing* **1**, 229–241 (1989).

[13] Hoover, H.J., Ruzzo, W.L.: A Compendium of Problems Complete for P. Manuscript (1984).

[14] Immerman, N.: Expressibility and parallel complexity. *SIAM J. Comput.* **18**, 3, 625–638 (1989).

[15] Kanellakis, P.C., Smolka, S. A.: CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation* **86**, 202–241 (1990).

[16] Karp, R., Ramachandran, V.: A Survey of Parallel Algorithms for Shared Memory Machines. In: *Handbook of Theoretical Computer Science*, North-Holland (1990).

[17] Keller, R.M.: Formal Verification of Parallel Programs. *Comm. ACM*, **19**, 7, 371–384 (1976).

[18] Mazurkiewicz, A: Basic notions of trace theory. Springer Verlag Lecture Notes in Computer Science 354, 285–363 (1989).

[19] Milner, R.: *A Calculus of Communicating Systems.* Springer Verlag Lecture Notes in Computer Science 92 (1980).

[20] Milner, R.: *Communication and Concurrency.* Prentice Hall (1989).

[21] Milner, R.: A Complete Axiomatization for Observation Congruence of Finite-State Behaviours. *Information and Computation.*

[22] Miyano, S., Shiraishi, S., Shoudai, T.: A list of P-complete problems. Technical Report RIFIS-TR-CS-17, Kyushu University 33, 1989.

[23] Parberry, I.: A primer on the complexity theory of neural networks. In: *Formal techniques in artificial intelligence*, R.B. Banerji (editor), North-Holland (1990).

[24] Park, D.: Concurrency and Automata on Infinite Sequences. Springer Verlag Lecture Notes in Computer Science 104, 168–183 (1981).

[25] Peterson, J.L.: *Petri net theory and the modeling of systems.* Prentice-Hall (1981).

[26] Stockmeyer, L., Vishkin, U.: Simulation of parallel random access machines by circuits. *SIAM J. Comput.* **13**, 2, 409–422 (1984).

[27] Walker, D.J.: Automated Analysis of Mutual Exclusion Algorithms using CCS. *Formal Aspects of Computing*, 1, 273–292 (1989).

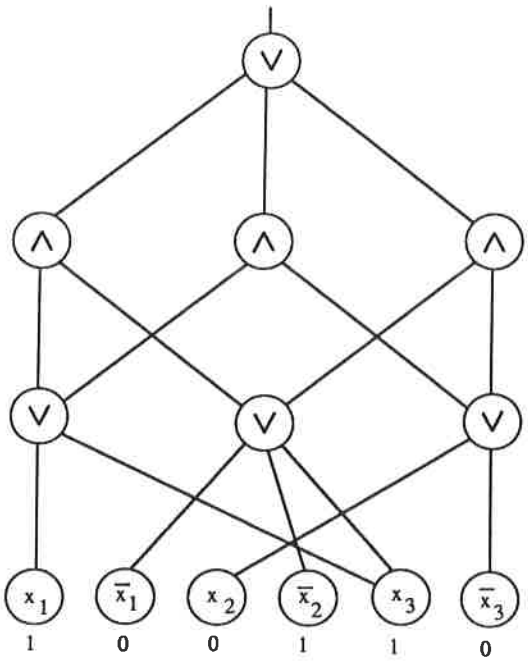

Fig. 1 A monotone alternating boolean circuit C

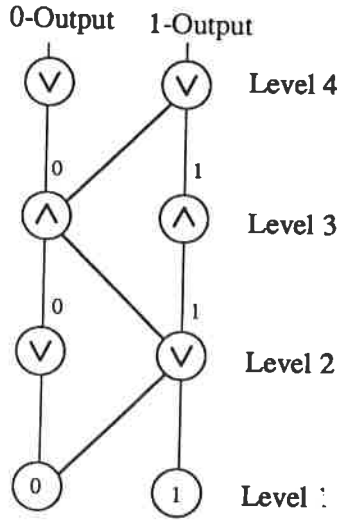

Fig. 2 The 4-alternating pattern A_4

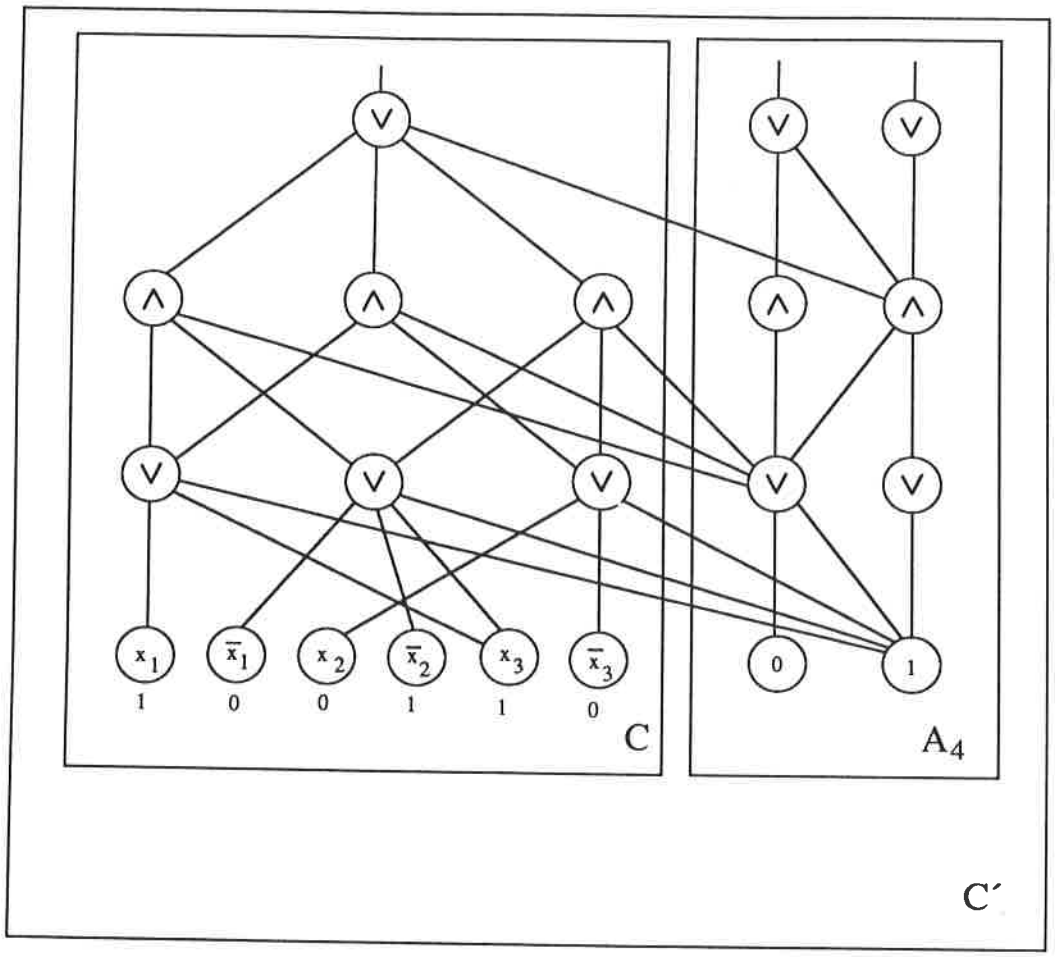

Fig. 3 The coupling of C and A_4 into C'

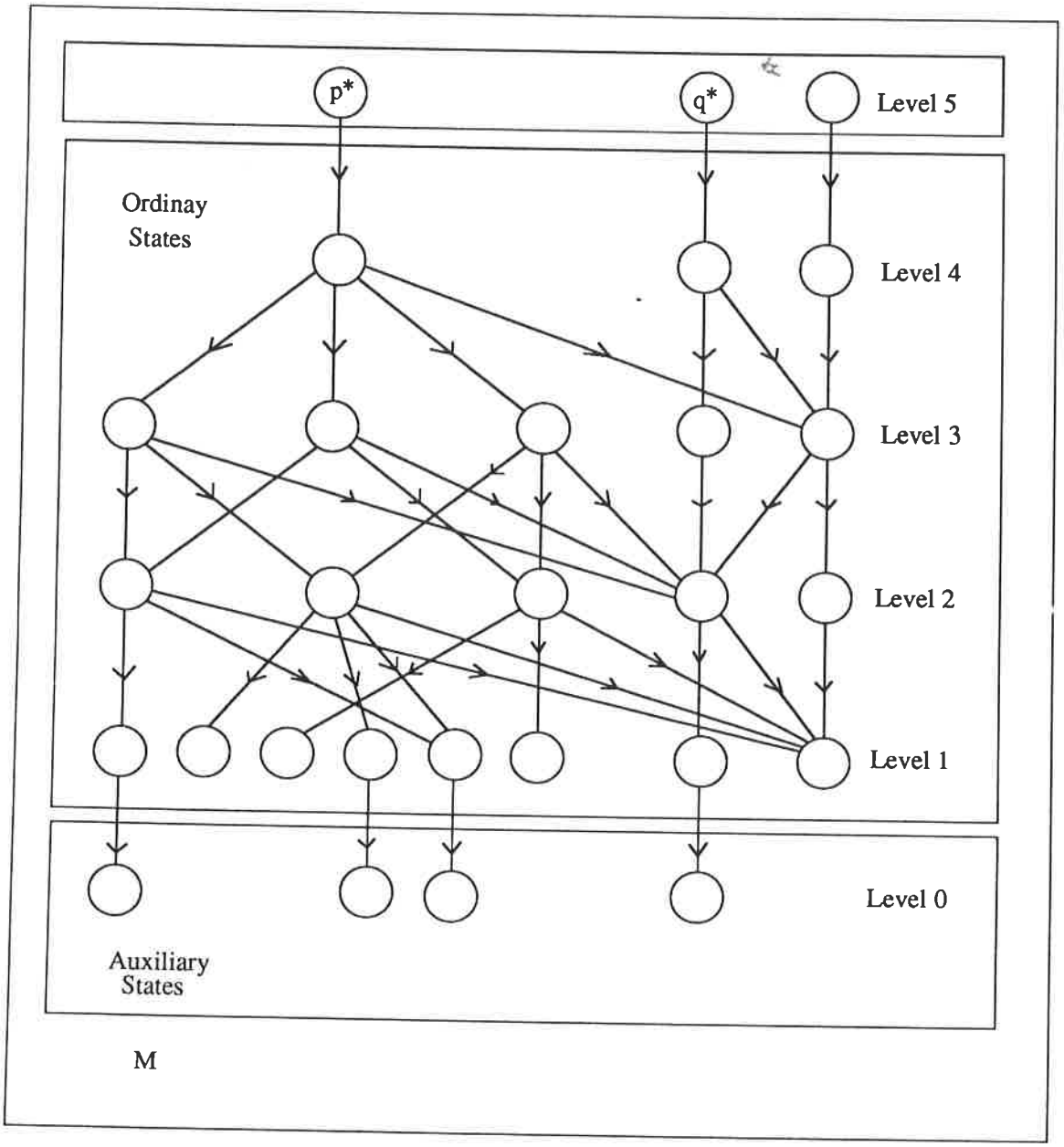

Fig. 4 The transition system M corresponding to C'