Formal Aspects of Computing (1992) 4: 638–648

Formal Aspects of Computing

Deciding Bisimilarity is *P*-Complete

José Balcázar,[a] Joaquim Gabarró[a] and Miklós Sántha[b]

[a] Dep. de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Barcelona, Spain, and
[b] CNRS, URA 410, Université Paris-Sud, LRI, Orsay, France

Keywords: CCS; Bisimilarity; *P*-completeness; Many–one *NC*-reduction

Abstract. In finite labelled transition systems the problems of deciding strong bisimilarity, observation equivalence and observation congruence are *P*-complete under many–one *NC*-reducibility. 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 *NC* algorithm to solve any of these problems will require an algorithmic breakthrough, which is exceedingly hard to achieve.

1. Introduction

Given the intrinsic difficulty of designing large software systems, it is natural that software tools would be designed to help to perform this task. The possibility of formalising 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, and their verification using formal systems may give rise to formidable computational problems. For instance, already the study of the correctness and liveness properties of mutual exclusion algorithms for just two processes obtains substantial help from computerised analysis [Wal89]; if even more processes are considered then the state space soon becomes intractable.

One reason to develop concurrent programs stems from the fact that important advantages can be gained from the use of massive parallelism. One hopes that such an application would be the study of concurrent systems, and that algorithms

Correspondence and offprint requests to: J. Gabarró, Dep. de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Pau Gargallo 5, 08028 Barcelona, Spain.

running on highly parallel machines could perform automated analysis of large concurrent programs substantially faster than sequential algorithms. A possible way of modelising this requirement is to try to attain a running time roughly logarithmic in the size of the state space of the concurrent program (assumed finite); moreover, to be able to tackle problems of relevant size, the algorithms must be restricted to a large but feasible number of processors (cf. the definition of the class NC below).

In particular, it seems natural to expect from such a software tool the capability to decide some form of equivalence of finite state systems. This problem plays a fundamental role in the study of concurrent systems, and has been widely studied both from a theoretical and practical point of view. Milner specifies in [Mil89b] a complete set of axioms for proving equivalence of finite state agents. Kanellakis and Smolka consider in [KaS90] efficient sequential algorithms to solve this problem. On the more practical side, the prototype named Concurrency Workbench [CPS89], implemented in Standard ML, has been used by Walker [Wal89] for undertaking the automated analysis of mutual exclusion algorithms. Finite state systems can be used for that purpose, since 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. However, in view of the advantages that might be obtained from their parallelisation, and of the large number of parallel algorithms discovered in recent years (for overviews see [KaR90] and [GiR88]), a natural question to ask about these bisimulation techniques is: do they admit solution by means of 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, and other related problems are P-complete. P-complete problems have efficient sequential algorithms but it is widely believed that they do not admit fast parallel ones. Thus our negative results raise in turn more general questions. What kind of properties of parallel and distributed systems can be decided via fast parallel algorithms? Also, how can we avoid using bisimulation techniques in the design of these systems?

A preliminary version of this paper was presented at the conference PARLE'91 as a part of the reference [ABG91].

2. Preliminaries

Concurrent systems can be analysed through transition systems [Kel76]. 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 formalised by the notion of strong-bisimulation [Par81]. The following definitions are borrowed from [Mil89a].

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:

1. Whenever $p \xrightarrow{x} p'$, then for some q', $q \xrightarrow{x} q'$ and $(p', q') \in S$.
2. 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.

To keep information about the internal behaviour of the system, we can enlarge the set of actions Σ with an invisible action τ. The new set of actions is then $Act = \Sigma + \tau$. Given $w \in Act^*$, we denote $\hat{w} \in \Sigma^*$ the word obtained by deleting all the occurrences of τ in w. Given $w = x_1 \ldots x_n \in Act^*$, we use the notation $q \stackrel{w}{\Longrightarrow} q'$ if there exists a path of transitions

$$q(\stackrel{\tau}{\rightarrow})^* \stackrel{x_1}{\rightarrow} (\stackrel{\tau}{\rightarrow})^* \cdots (\stackrel{\tau}{\rightarrow})^* \stackrel{x_n}{\rightarrow} (\stackrel{\tau}{\rightarrow})^* q'$$

from q to q'.

To consider equivalence up to internal behaviour, we define the notion of bisimulation. A relation $S \subseteq Q \times Q$ is a *bisimulation* if $(p,q) \in S$ implies, for all $u \in Act$, the following two conditions:

1. Whenever $p \stackrel{u}{\longrightarrow} p'$, then for some q', $q \stackrel{\hat{u}}{\Longrightarrow} q'$ and $(p',q') \in S$.
2. Whenever $q \stackrel{u}{\longrightarrow} q'$, then for some p', $p \stackrel{\hat{u}}{\Longrightarrow} p'$ and $(p',q') \in S$.

The *observation equivalence* or *bisimilarity* relation $\approx$ is defined as

$$\approx = \bigcup \{S \mid S \text{ is a bisimulation } \}$$

Finally, to deal with internal actions just at the start of the processes, we consider the notion of observation congruence. By definition, two states p and q are *observation congruent* (in notation $p = q$) if for all $u \in Act$, we have

1. Whenever $p \stackrel{u}{\longrightarrow} p'$, then for some q', $q \stackrel{u}{\Longrightarrow} q'$ and $p' \approx q'$.
2. Whenever $q \stackrel{u}{\longrightarrow} q'$, then for some p', $p \stackrel{u}{\Longrightarrow} p'$ and $p' \approx q'$.

For the formal study of the possible existence of parallel algorithms we will consider two complexity classes: P and NC. The first one models problems with *efficient sequential computation*; the second one models problems with *fast parallel computation*, using a feasible number of processors. Each of these classes has many characterisations that support this description.

By definition, the class P contains the problems for which a polynomial time sequential algorithm exists. This can be formalised 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 [AHU75]), polynomial time Turing machines ([AHU75], [BDG88]), or even polynomial size uniform circuits (see below) are all suitable for this purpose, and give equivalent definitions of the class P.

A basic computational model that we need for the definition of the class NC and also for the proof of the main results below is the *boolean circuit* model. A boolean circuit is a directed, acyclic, labelled graph of maximum indegree 2, in which the nodes of indegree zero are the inputs, and each node of indegree k computes the k-ary boolean function indicated in its 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

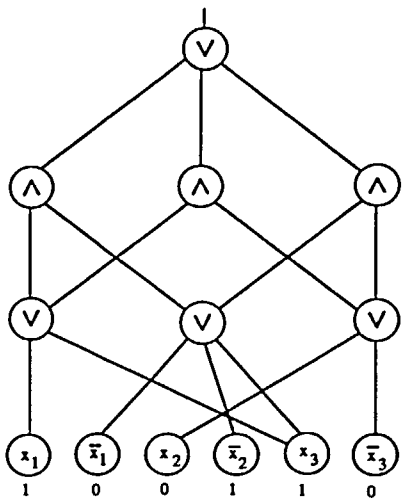

Fig. 1. A monotone alternating boolean circuit

in a circuit are called also *gates*. Occasionally we resort to gates with more than two inputs. If gates with more than two inputs exist in the circuits, then they are assumed to be shorthand forms for subcircuits of gates with two inputs, e.g. an AND gate of indegree 3 stands for two AND gates, the first computing the AND of two of the inputs and the second computing the AND of this result with the third input.

Various additional hypotheses can be assumed on a given circuit. The following restriction will be important for us: a *monotone alternating circuit* is divided into levels, so that the inputs to a gate at a given level are all outputs of gates from the previous level. The circuit contains only AND and OR gates. To make up for the absence of negation gates, it receives each input together with its negation. All gates in the same level are of the same type, and the levels alternate between AND and OR levels. Gates AND and OR with only one input behave like identity. Figure 1 shows an example of a monotone alternating circuit.

A boolean circuit computes a boolean function by substituting values for the inputs, evaluating all the nodes, and collecting values at the output nodes. Binary inputs and outputs might be binary encodings of other objects assuming some simple coding scheme. To use boolean circuits for computing functions on an infinite domain (e.g. on the set of all finite binary sequences), we have to select a different circuit for each input length. In principle, 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. By definition, a family of circuits is *uniform*, if it is possible to construct for each length n, the circuit corresponding to this input length using an algorithm that requires only a very small amount of resources (such as logarithmic memory space). Discussions of the different ways of defining uniformity appear in [BDG90] and [BIS90].

The class NC is defined as the class of all problems that can be solved by uniform circuits of polynomial size and polylogarithmic (i.e. $O(\log^k n)$, for some $k > 0$) depth [BDG90]. Intuitively, the depth of a circuit measures the parallel

time needed to compute the corresponding function and its size the necessary hardware, and this intuition is supported by existing characterisations of NC in terms of more natural computational models such as the parallel RAM. Thus the class NC is a plausible formalisation of the concept of efficiently parallelisable problem.

We need from complexity theory the notion of many–one NC-reducibility and the corresponding notion of completeness for P. By definition problem S is *many–one NC-reducible*, (*NC-reducible* for short) to problem T if there exists a uniform family of circuits $\{c_n\}$ of polynomial size and polylogarithmic depth such that for every positive integer n and for every $x \in \{0,1\}^n$, we have

$$x \in S \iff c_n(x) \in T$$

where $c_n(x) \in \{0,1\}^*$ denotes the output of the circuit c_n with input x [All89]. A problem S is *P-complete* under NC-reduction if $S \in P$ and every problem in P is NC-reducible to S. In a sense, P-complete problems could be identified as inherently sequential ones: an NC algorithm found for a P-complete problem would imply the existence of very fast parallel algorithms for every problem in P. However, 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 [HoR84] and [MSS89]. Our main results prove the P-completeness of several problems on LFTSs.

Since NC-reducibility is transitive, the usual way of proving the P-completeness of a problem in P is to reduce some known other complete problem in P to it. There are several standard P-complete problems which are natural candidates for the reduction. One of these is the Circuit Value Problem. The input to this problem is a 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 such a variant [HoR84]. The Monotone Alternating Circuit value problem is P-complete under many–one NC-reductions:

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.

3. Main Results

Our main goal is to prove that the Strong Bisimilarity problem is P-complete under many–one NC-reduction.

Input: an encoding of a finite transition system with two selected states p^* and q^*.
Output: decide if p^* and q^* are strongly bisimilar.

It is well known that deciding strong bisimilarity in a LFTS is a P problem [KaS90]. To see this, it suffices to construct the strong bisimilarity relation $\sim$ as intersection of the sequence of relations $\equiv_0, \equiv_1, \ldots$, which are defined by induction as follows (see [Mil80]):

1. For every $(p,q) \in Q \times Q$, $p \equiv_0 q$
2. $p \equiv_{i+1} q$ 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 $p' \equiv_i q'$.

Then it can be seen immediately that the relation $\sim$ can be constructed in polynomial time since it is just the relation $\equiv_k$ where k is the number of states in the finite transition system. More efficient algorithms to solve this problem have been considered in [PaT87] and [KaS90]. The P-completeness of the Strong Bisimilarity problem will follow from the following lemma which constitutes the main part of our result.

Lemma 1. The Circuit Value Problem for Monotone Alternating Circuits can be many-one NC-reduced to the Strong Bisimilarity Problem.

Proof. We will transform an instance of the Circuit Value Problem for Monotone Alternating Circuits into an instance of the Strong Bisimilarity Problem in three steps. In the first step we design an auxiliary circuit. In the second step we combine it with the original circuit C to get a new circuit C'. Finally, in the third step we transform C' into a finite labelled transition system.

Let us suppose that we are given a monotone alternating circuit C which has k levels. The input gates of the circuits are on the first level, and the output gate is on the kth level. This number can be found fast in parallel.

1. We define the k-alternating pattern A_k. This is a monotone alternating circuit of depth k, where every level has two gates. One of the two gates computes the constant 0, and the other the constant 1. At the input level (first level) the two gates are the constants 0 and 1. At an OR level, the gate with value 0 gets a unique wire from the gate of the previous level which has the value 0; the gate with value 1 gets a wire from both gates of the previous level. The wires entering an AND level are defined in a complementary way: the gate with value 1 gets a unique wire from the gate of the previous level which has the value 1; the gate with value 0 gets a wire from both gates of the previous level. It is easy to check then that in A_k the following two conditions are satisfied:

Every OR gate has an input with value 0.
Every AND gate has an input with value 1.

The gates at the kth level of A_k are respectively called the 0-output and the 1-output. We can construct A_k fast in parallel since it has a very regular structure. Figure 2 gives us A_4.

2. We couple the k-alternating pattern A_k with the circuit C to get a new circuit C'. The gates of C' are the gates of A_k and C, and all the wires of these two circuits are also wires of C'. In addition, to every gate of C we add a wire coming from the preceding level of the k-alternating pattern. If the gate is an OR, the wire comes from the gate with value 0. If the gate is an AND, the wire comes from the gate with value 1. The circuit C' satisfies the following three properties:

Every OR gate has at least an input with value 0.
Every AND gate has at least an input with value 1.
Every gate of C' evaluates equally to the corresponding gate in A_k or C.

Figure 3 shows C' in our example.

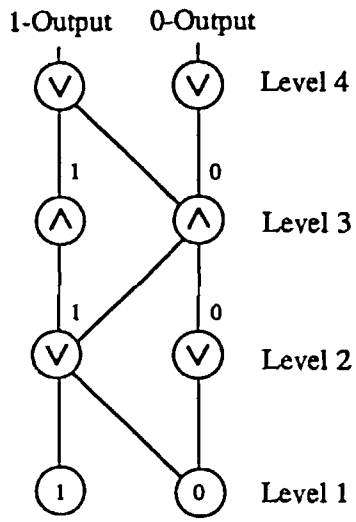

Fig. 2. The 4-alternating pattern A_4

3. We now transform the circuit C' into an FLTS M over a one-letter alphabet. M contains a state corresponding to each gate of C'. These states are called *ordinary states*. They are naturally distributed on k levels, corresponding to the k levels of C'. 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.

To each wire of C' corresponds a transition of M. The transition goes from the output of the wire to its input. In addition, there is a transition from each state on the first level corresponding to a gate which evaluates to 1, to its auxiliary state at level 0. All the transitions are labelled by the unique letter in the alphabet. Finally, let us specify two states p^* and q^* of M. State p^* is the one which corresponds in M to the output gate of C. State q^* is the one which corresponds to the 1-output gate of A_k. Figure 4 shows M in our example.

We are now ready to show 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. We will show the implication in both directions.

($\Rightarrow$): Let us suppose that C evaluates to 1. We define a binary relation S over the states of M. Let us say that an ordinary state of M is a 0-state, if it corresponds to a gate of C' which evaluates to 0. A 1-state is defined similarly. Let p and q be two states of of M. Then we define S such that $(p,q) \in S$ if one of the following conditions is satisfied:

p and q are both auxiliary states.

p and q are on the same level and they are both 0-states.

p and q are on the same level and they are both 1-states.

We will show that S is a strong bisimulation. By definition $(p^*,q^*) \in S$, this will imply that p^* and q^* are strongly bisimilar. Let p and q be states such that $(p,q) \in S$. We will show that the bisimilarity conditions are satisfied by the couple (p,q). There is nothing to show for the states on level 0. There is nothing to show

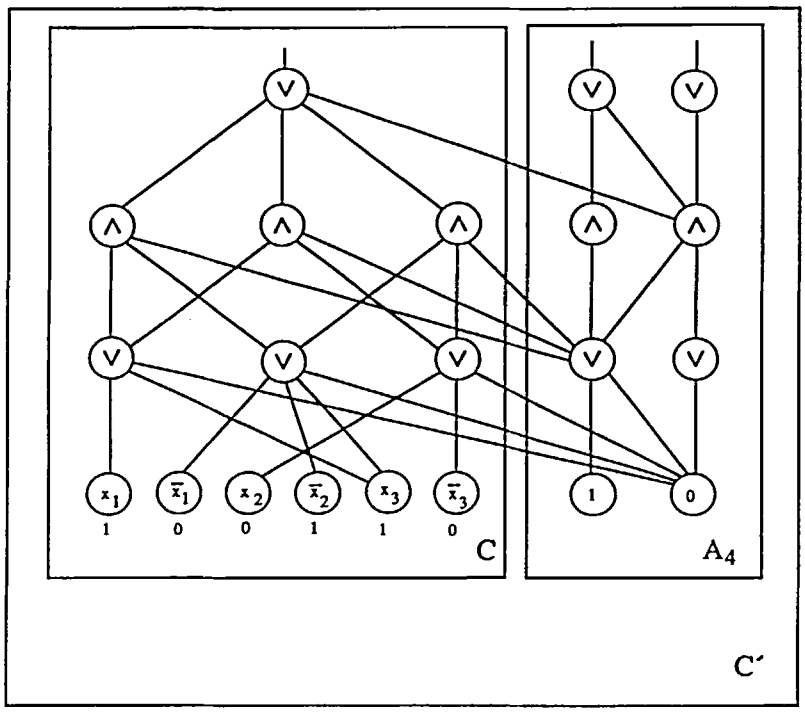

Fig. 3. The coupling of C and A_4 into C'

either for the 0-states of level 1, and the 1-states of level 1 satisfy the conditions since all the states of level 0 are in relation according to S.

Let us suppose that states p and q are from level $m > 1$. Without loss of generality we can suppose that the mth level corresponds to an OR level in C' (the argument is complementary for an AND level). The states p and q are either 0-states or 1-states. In the first case we claim that all the successors of both p and q are 0-states. This is indeed true since otherwise the corresponding OR gates in C' would not evaluate to 0. Therefore by the definition of the relation S, for every successor p' of p and every successor q' of q, we have $(p', q') \in S$.

Let us now suppose that p and q are 1-states. Then we claim that they both have at least one 1-state and at least one 0-state as successor. The claim about the 1-states follows from an argument complementary to our previous one. The claim about the 0-states follows from our construction: as we observed in C' every OR gate has at least one 0-entry. In fact, this is the point where we need the alternating pattern.

($\Leftarrow$): Let us suppose that C evaluates to 0. We will show that p^* and q^* cannot be bisimilar. This will follow from the following claim: for any input assignment, for every 0-state p and every 1-state q, for every bisimulation S, if p and q are on the same level, then $(p, q) \notin S$. The claim is proved by induction on the level of the states.

If they are on the first level then this follows from the fact that q has a transition while p has none. If they are on a level $m > 1$ then we can suppose that

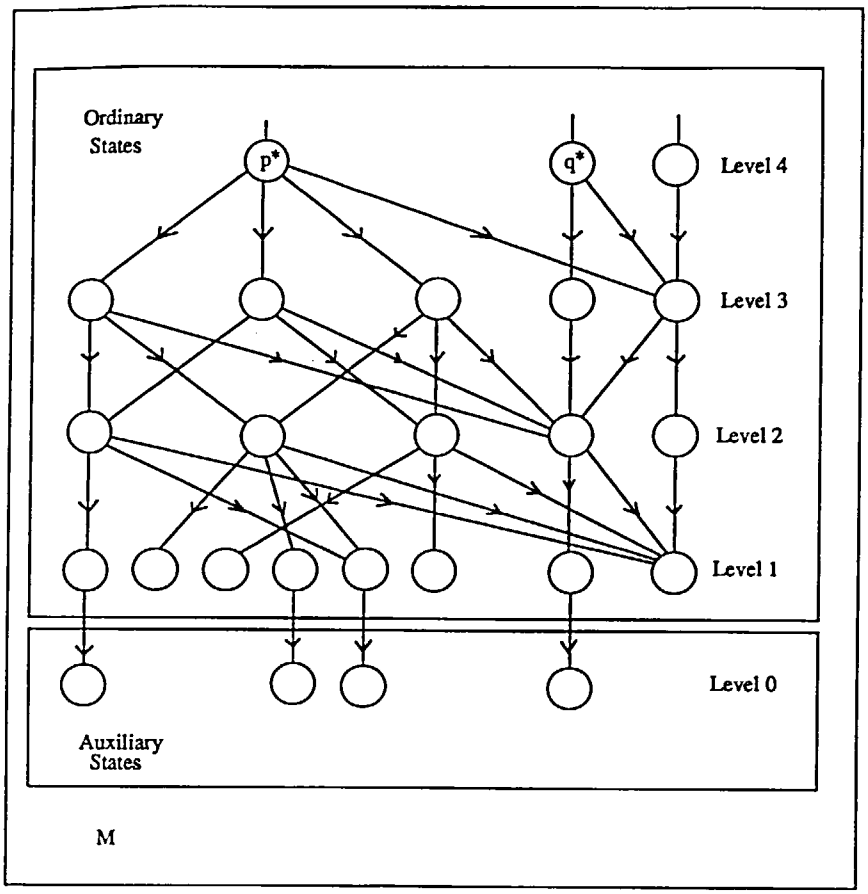

Fig. 4. The transition system M corresponding to C'

this is an OR level (as usual, the argument for an AND level is complementary). Obviously, q must have a 1-state successor q'. On the other hand, for any successor p' of p, we know that p' is a 0-state. By our inductive hypothesis, p' and q' are not bisimilar. Thus p and q are not bisimilar either. This completes the induction and the proof of the lemma. □

As a consequence of Lemma 1 and the discussion preceding it we obtain our main result.

Theorem 1. The Strong Bisimilarity problem is P-complete under NC-reduction.

Input: an encoding of a finite transition system and two selected states p and q.
Output: decide whether p and q are strongly bisimilar.

The problem of Strong Bisimilarity can trivially be NC-reduced to both Observation Equivalence and Observation Congruence. Moreover, these two problems can be solved in P by reducing them in polynomial time via a transitive closure algorithm to Strong Bisimilarity. As a corollary, we have the following two P-completeness results.

Theorem 2. The Observation Equivalence problem is P-complete under NC-reduction.

Input: an encoding of a finite transition system and two states p and q.
Output: decide whether p and q are observation equivalent.

Theorem 3. The Observation Congruence problem is P-complete under NC-reduction.

Input: an encoding of a finite transition system and two states p and q.
Output: decide whether p and q are observation congruent.

4. Conclusions

We have presented quite a precise classification of the problem of deciding strong bisimilarity in LFTSs in terms of its computational complexity, by showing that it is complete for the class P. We have also discussed the intuitive implications of this result. We now want to complete the discussion by raising two sorts of questions.

1. First comes the question of whether it could be possible to work with some useful concept similar to bisimulation, adequate to describe appropriately some sort of process equivalence, but having efficient parallel NC algorithms. Regarding this question we can consider two different aspects:

- Are there useful subclasses of bisimulations which are not P-complete? For instance, are there useful particular cases of LFTS for which deciding bisimulation is in NC? Natural candidates are the processes of in degree 1. For processes of in degree bounded by any constant larger than 1 our construction here can be easily adjusted to prove P-completeness.
- Are there some kind of approximations to bisimulations which are easier to compute? What good might they be?

2. Second, we must acknowledge some limitations of the notions we use. We have accepted here NC to capture the proper notion of efficient parallel computation. Therefore, the P-completeness of a problem reminds us strongly of the lack of efficient parallel algorithms to solve it. However, a significant number of reserchers feel unease with this approach. For large problems, n^k processors with large k may seem unreasonable, and on the contrary, speeding problems up to polylog time may seem excessively demanding; see [ViS86] for alternative definitions of complexity classes addressing these issues. It would be interesting to study the complexity of bisimilarity under this other approach.

Acknowledgement

J. L. Balcázar and J. Gabarró acknowledge support from the ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (project ALCOM).

M. Sántha acknowledges support from 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.

We are grateful to the referees for their careful reading and their interesting comments and suggestions.

References

[AHU75] Aho, A., Hopcroft, J. and Ullman, J.: *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1975.

[All89] Allender, E.: *P*-Uniform Circuit Complexity. *J. ACM*, **36**(4), 912–928 (1989).

[ABG91] Àlvarez, C., Balcázar, J.L., Gabarró, J. and Sántha, M.: Parallel Complexity in the Design and Analysis of Concurrent Systems. In: *Parallel Architectures and Languages Europe PARLE'91*, Lecture Notes in Computer Science 505 Springer-Verlag pp. 288–303, 1991.

[BDG88] Balcázar, J.L., Díaz, J. and Gabarró, J.: *Structural Complexity I.* Springer-Verlag, 1988.

[BDG90] Balcázar, J.L., Díaz, J. and Gabarró, J.: *Structural Complexity II.* Springer-Verlag, 1990.

[BIS90] Barrington, D.A.M., Immerman, N. and Straubing, H.: On Uniformity Within NC, *Journal of Computer and System Sciences*, **41**, 274–306 (1990).

[GiR88] Gibbons, A. and Rytter, W.: *Efficient Parallel Algorithms.* Cambridge University Press, 1988.

[CPS89] Cleaveland, R., Parrows, J. and Steffen B.: The Concurrency Workbench. In: *Automatic Verification Methods for Finite State Systems.* Lecture Notes in Computer Science 407, Springer-Verlag, pp. 24–37, 1989.

[HoR84] Hoover, H.J. and Ruzzo, W.L.: A Compendium of Problems Complete for *P*. Manuscript, 1984.

[KaS90] Kanellakis, P.C. and Smolka, S. A.: CCS Expressions, Finite State Processes, and three Problems of Equivalence. *Information and Computation*, **86**, 202–241 (1990).

[KaR90] Karp, R. and Ramachandran, V.: Parallel Algorithms for Shared-Memory Machines. In: *Handbook of Theoretical Computer Science*, Volume A. North-Holland, pp. 869–941, 1990.

[Kel76] Keller, R.M.: Formal Verification of Parallel Programs. *C. ACM*, **19**(7), 371–384 (1976).

[Mil80] Milner, R.: *A Calculus of Communicating Systems.* Lecture Notes in Computer Science 92, Springer-Verlag, 1980.

[Mil89a] Milner, R.: *Communication and Concurrency.* Prentice Hall, 1989.

[Mil89b] Milner, R.: A Complete Axiomatization for Observation Congruence of Finite-State Behaviours. *Information and Computation*, **81**, 227–247 (1989).

[MSS89] Miyano, S., Shiraishi, S. and Shoudai, T.: A List of *P*-complete Problems. Technical Report RIFIS-TR-CS-17, Kyushu University 33, 1989.

[PaT87] Paige, R. and Tarjan, R.E.: Three Partition Refinement Algorithms. *SIAM Journal on Computing*, **16**(6), 973–989 (1987).

[Par81] Park, D.: Concurrency and Automata on Infinite Sequences, Lecture Notes in Computer Science 104. Springer-Verlag pp. 168–183, 1981.

[ViS86] Vitter and J.S., Simons, R.A.: New Classes for Parallel Complexity: a Study of Unification and Other Complete Problems for *P*. *IEEE Transactions on Computers*, **C-35**(5), 403–418 (1986).

[Wal89] Walker, D.J.: Automated Analysis of Mutual Exclusion Algorithms Using CCS. *Formal Aspects of Computing*, **1**, 273–292 (1989).

Received July 1990
Accepted in revised form November 1991 by R. Milner