1400008549
còpia 1

Deciding Bisimilarity is P-complete

José L. Balcázar
Joaquim Gabarró
Miklos Santha

Report LSI–90–25

FACULTAT D'INFORMÀTICA
BIBLIOTECA
R. 7982 30 JUL. 1990

Keywords: CCS; bisimilarity; P-completeness; many-one NC-reductions.

Abstract. On 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 efficient parallel algorithm to solve any of these problems will require an exceedingly hard algorithmic breakthrough.

Resum. Per a sistemes de transició etiquetats i finits, els problemes de decidir la bisimilaritat forta, l'equivalència d'observació i la congruència d'observació són P-complets sota les NC-reductibilitat. Com a conseqüència, els algoritmes fonamentats en la bisimulació per l'anàlisi automàtica dels sistemes finits semblen ser inherentment sequencials en el següent sentit: el disseny d'un algoritme paral·lel eficient per a resoldre aquests problemes comportaria la solució d'importants problemes oberts.

Deciding Bisimilarity is P-complete

J.L. Balcázar* J. Gabarró*
Dep. de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Pau Gargallo 5, 08028 Barcelona
Spain

M. Santha+
CNRS, Laboratoire de Recherche en Informatique
Université Paris-Sud
91405 Orsay
France

Keywords: CCS; bisimilarity; P-completeness; many-one NC-reductions.

Abstract. On 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 efficient parallel algorithm to solve any of these problems will require an exceedingly hard algorithmic breakthrough.

* 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.

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 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, 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 two processes already requires resorting to computerized analysis [Wal89]; if more processes are considered, 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. 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).

In particular, a capability that seems natural to expect from such a software tool is to be able 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 [Mil90b] 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, implemented in Standard ML, has been used by Walker [Wal89] 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. However, in view of the advantages that might be obtained from their parallelization, and of the large number of parallel algorithms discovered in recent years (for overviews see [KaR88] 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 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. 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?

2. Preliminaries

Concurrent systems can be analyzed 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 formalized 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:

(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.

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 $\hat{w} = x_1 \ldots x_n$, we use the notation $q \stackrel{\hat{w}}{\Longrightarrow} q'$ if there exists a path of transitions

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

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:

(i) whenever $p \xrightarrow{u} p'$, then for some q', $q \stackrel{\hat{u}}{\Longrightarrow} q'$ and $(p', q') \in S$,

(ii) whenever $q \xrightarrow{u} q'$, 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 equivalence. By definition, two states p and q are *observation equivalent* (in notation $p = q$) if for all $u \in Act$, we have:

(i) whenever $p \xrightarrow{u} p'$, then for some q', $q \stackrel{u}{\Longrightarrow} q'$ and $p' \approx q'$,

(ii) whenever $q \xrightarrow{u} q'$, 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 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 [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, the nodes of indegree 1 compute boolean negation, and the nodes of indegree 2 compute either boolean conjunction or disjunction, 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*. Occasionally we resort to gates with more or less than two inputs. If gates with more than two inputs exist in the circuits, then they are assumed to be shorthands 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. The gates AND and OR with only one input behave like identity.

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 immediately preceeding level. The circuit does not contain negation gates; instead 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. Figure 1 gives us 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). A discussion of the different ways of defining uniformity appears in [BDG90].

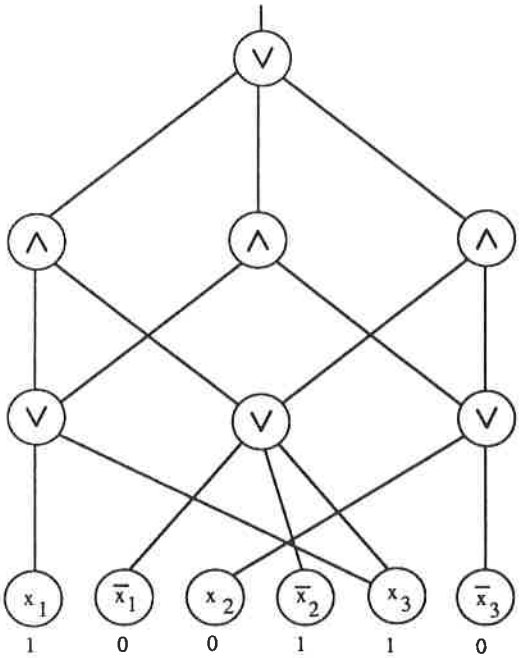

Fig. 1 An alternating boolean circuit C

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. Thus 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 some feasible (i.e. polynomial) amount of hardware. The well-foundedness of the class NC is reinforced by the fact that it turns out to be the same as the class of problems solvable by a Parallel RAM (PRAM) with a polynomial number of processors in polylogarithmic running time. There are many similar characterizations of NC in terms of other models of parallel computation. Polynomially many processors is in a sense reasonable amount, when massively parallel machines are being designed, and polylogarithmic parallel running time is fast even for large size instances. In fact, one of the reasons of the huge interest today in parallel computers is that they enable us to find solutions for such instances.

The last complexity-theoretic notions we need are *reducibility* and *completeness*. Reducibility is a useful notion for comparing the difficulty of different problems. Intuitively, a problem S can be reduced to a problem T if the existence of an algorithm of a certain kind that solves T implies the existence of a similar algorithm to solve S. This can be formalized in a variety of ways. Since the computational problems studied here all have decisional form (i.e. the problems have a yes/no answer), we concentrate only in what is called in complexity theory *many-one NC reducibility* (in short NC reducibility), considered for

example in [All89].

The idea is that S is NC-reducible to T, denoted here by $S \leq_m^{NC} T$, if every instance of S can be transformed into an instance of T for which the answer is the same, in polylogarithmic parallel time using a polynomial number of processors. We formalize this in terms of uniform circuits. We say that $S \leq_m^{NC} T$ 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. The interest of this definition lies in the fact that if S is NC-reducible to T and $T \in NC$, then $S \in NC$. Thus an efficient parallel algorithm for S can be obtained by composing the NC reduction with an NC algorithm for T.

A problem S is *P-complete* under NC reduction if $S \in P$ and every problem in P is NC-reducible to S. This concept plays here an analogous role 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 problems in P that do not admit efficient parallel algorithms, then all P-complete problems are among them. Conversely stated, if an NC algorithm is found for a P-complete problem, then $P = NC$, and thus all problems solvable in polynomial time have also 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 an NC algorithm 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. A survey of P-complete problems has appeared as [MSS89] or [HoR84]. Our main results prove the P-completeness of several problems on LFTS's.

It can be shown that NC 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. 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 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 [Mil80]. 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 because $\sim = \equiv_k$ where k is the number of states in the finite transition system. More efficient algorithms to solve this problem have been considered in [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.

(i) We define the k-alternating pattern A_k. This is a circuit of height k, where every level has two gates, one valuated to 0 and other valuated to 1. At the input level (first level) the two gates are the constants 0 and 1. At an OR level, the gate which evaluates to 0 gets a unique wire from the gate of the previous level which had the value 0; the gate which evaluates to 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 which evaluates to 1 gets a unique wire from the gate of the previous level which had the value 1; the gate which

evaluates to 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 valuated to 0,
- every AND gate has an input valuated to 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. Fig 2 gives us A_4.

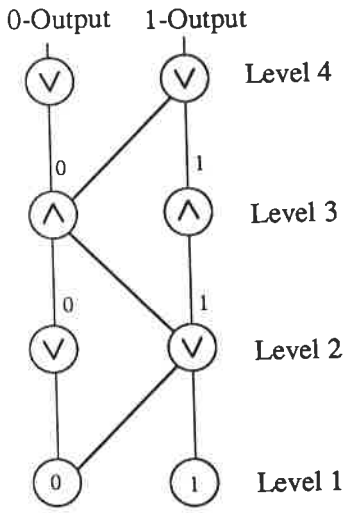

Fig. 2 The 4-alternating pattern A_4

(ii) 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 precedent level of the k-alternating pattern. If the gate is an OR, the wire comes from the gate valuated to 0. If the gate is an AND, the wire comes from the gate valuated to 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.

Fig 3 shows C' in our example.

(iii) 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.* 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

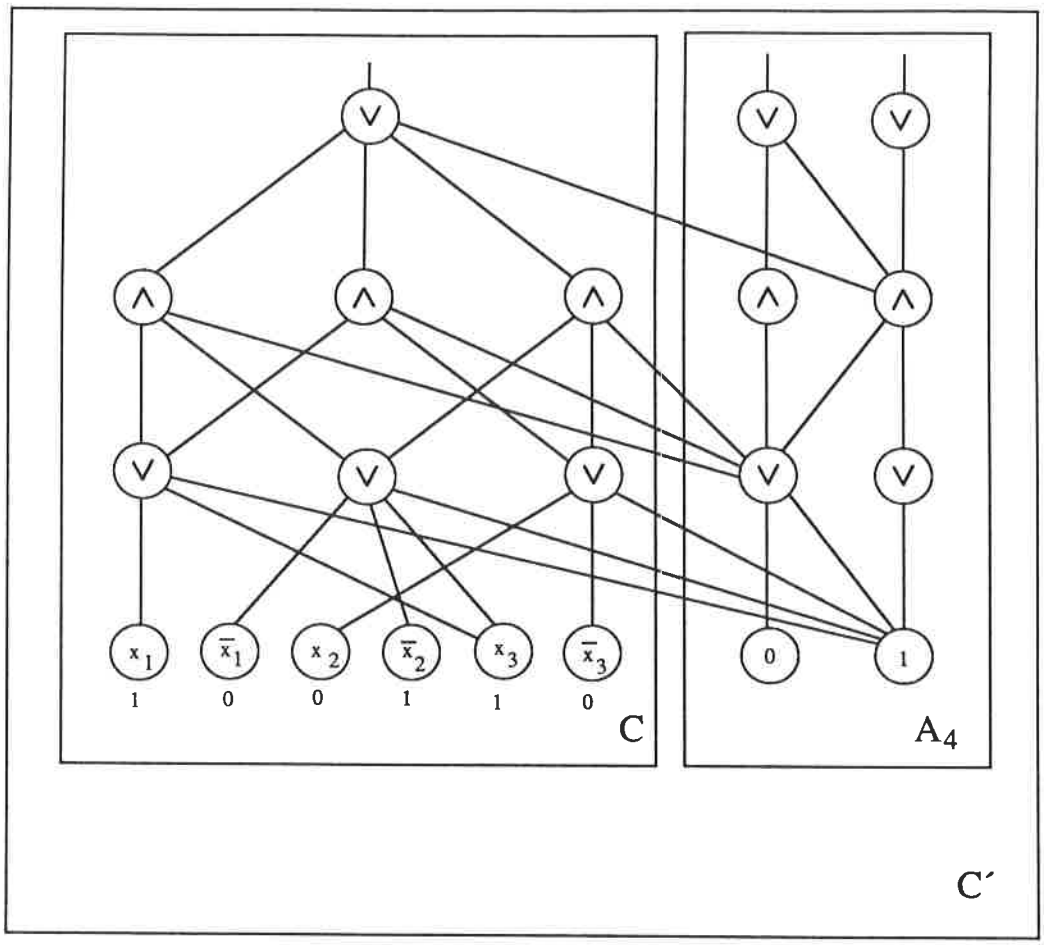

Fig. 3 The coupling of C and A_4 into C'

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 labeled 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. Fig 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

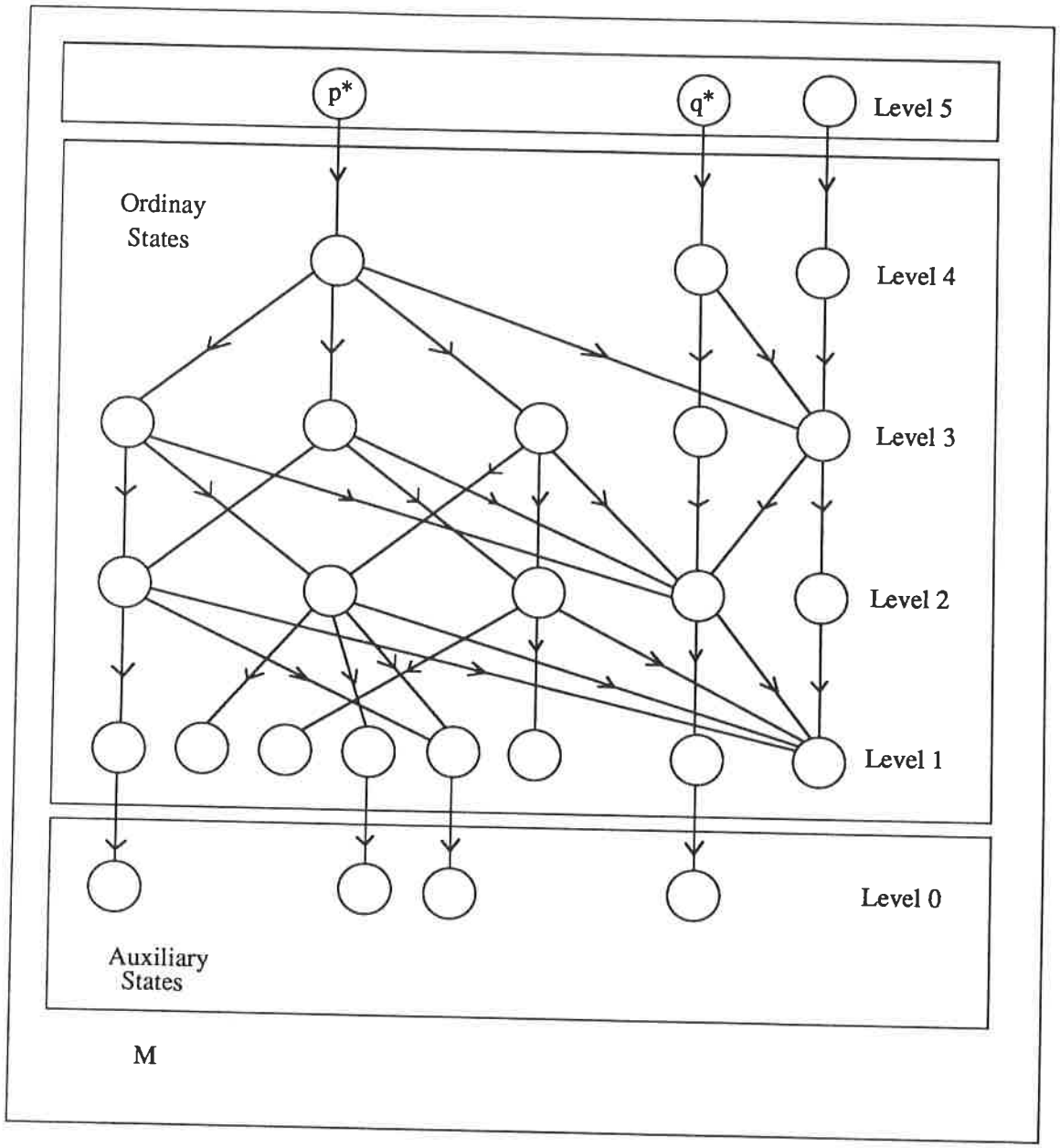

Fig. 4 The transition system M corresponding to C'

both directions.

Direction 1: 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. By definition, $(p,q) \in S$, 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. As 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 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 a complementary argument 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.

Part 2: Let us suppose that C evaluates to 0. We will show that p^* and q^* can not 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 this is an AND level (as usual, the argument for an OR level is complementary). Obviously, p must have a 0-state successor p'. On the other hand for any successor q' of q, we know that q' is a 1-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 the precedent lemma and the discussion preceeding it we obtain our main result.

Theorem 2: The Strong Bisimilarity problem is P-complete under NC-reductions

- *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 3: 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 4: 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 a quite precise classification of the problem of deciding strong bisimilarity in LFTS's 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 some questions of two sorts. First, a comparison with previous complexity-theoretic results classifying similar equivalence problems for other computation models calls for new concepts of equivalence that might be of practical value, yet testable by fast parallel algorithms; second, from the standpoint of a developer of a concurrent system, whether interaction with a software tool might be more efficient than completely automatic equivalence testing.

(1) Many notions of equivalence for various kinds of systems have been considered, each with different complexity-theoretic properties. Let us review some of them.

- Perhaps one of the better known concepts is language equivalence for nondeterministic finite automata. Unfortunately this equivalence is expensive to test, since it is complete for the, in principle, much larger complexity class $PSPACE$ [Sto74].
- In [JoP89] it is proved that deciding observation equivalence in programs that may read, store, and write data, but cannot perform any other computation, is NP-hard.
- As shown here, observation equivalence for finite transition systems is P-complete.
- At the end of the chain we have that state equivalence for deterministic finite automata can be decided in NC [KRe89].

Thus a natural open question along this line is to find a useful notion of equivalence for finite labelled transition systems decidable in NC. This might be obtained through a careful analysis of the properties of concurrent systems, trying to find out which parts of the analysis are inherently sequential and which parts can be done in parallel.

(2) It is well known from the study of NP problems that in many cases "to verify" is easier than "to compute". 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 [HeJ89], 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

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

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

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

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

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

[HeJ89] He Jifeng: Process Simulation and Refinement, *Formal Aspects of Computing*, 1, 229–241 (1989)

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

[JoP 89] Jonsson, B. Parrow, J.: Deciding Bisimulation Equivalences for a Class of Non-Finite State Programs, Springer Verlag Lecture Notes in Computer Science 349, pp 421–433, 1989.

[KaR88] Karp, R., Ramachandran, V.: A Survey of Parallel Algorithms for Shared Memory Machines, Technical Report UCB/CSD 88/408, 1988. To appear in *Handbook of Theoretical Computer Science*, to be published by North-Holland.

[KRe89] Kanellakis, P.C., Revesz, P.Z.: On the Relationship of Congruence Closure and Unification. *J. Symbolic Computation* 7, 427–444, 1989.

[KaS90] Kanellakis, P.C., Smolka, S. A.: CCS Expressions, Finite State Processes, and three Problems of Equivalence, *Information and Computation*, 86, 202–241, (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*, Springer Verlag Lecture Notes in Computer Science 92, 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.*

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

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

[Sto74] Stockmeyer, L.: The Complexity of Decision Problems in Automata Theory and Logic, MAC TR–133, Project MAC, MIT, Cambridge, Mass 1974.

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