Deciding Bisimilarity over Finite Labelled Transition Systems is P-complete

Luca Aceto Anna Ingolfsdottir

October 23, 2008

In what follows we consider *finite* labelled transition systems, that is, labelled transition systems with a finite set of states and finitely many transitions. We recall that, as remarked in Section 3.6 of the book, for a labelled transition system with n states and m transitions, strong bisimilarity between any two given states is decidable in deterministic polynomial time—more precisely in $O(nm)$ time (Kanellakis and Smolka, 1990). This result by Kanellakis and Smolka was subsequently improved upon by Paige and Tarjan who devised an algorithm that runs in $O(m \log n)$ time (Paige and Tarjan, 1987). This is in strong contrast with the complexity of deciding language equivalence, where the problem is known to be PSPACE-complete (Hunt, Rosenkrantz and Szymanski, 1976)—that is, one of the hardest decision problems that can be solved using a polynomial amount of space.

The problem of checking observational equivalence (weak bisimilarity) over finite labelled transition systems can be reduced to that of checking strong bisimilarity using a technique called *saturation*. Intuitively, saturation amounts to

1. first pre-computing the weak transition relation (see Definition 3.3 in the book), and then

2. constructing a new pair of finite processes whose original transitions are replaced with the weak transitions.

The question whether two states are weakly bisimilar now amounts to checking strong bisimilarity over the saturated systems. Since the computation of the weak transition relation can be carried out in polynomial time (and you should convince yourself of this claim), the problem of checking for weak bisimilarity can also be decided in polynomial time. The same holds true for the problem of checking observational congruence (Milner, 1989). (See Exercise 3.36 in the book for the definition of observational congruence.)

Efficient algorithms are also available for deciding a variation on the notion of weak bisimilarity called *branching bisimilarity* (Glabbeek and Weijland, 1996) over finite labelled transition systems. (Branching bisimilarity is the subject of Exercise 3.35 in the book.) Notably, Groote and Vaandrager have developed in (Groote and Vaandrager, 1990) an algorithm for checking branching bisimilarity that has time complexity $O(m \log m + mn)$ and space complexity $O(m + n)$.

These algorithmic results indicate that strong, weak and branching bisimilarity can be decided over finite labelled transition systems faster than many other equivalences.

This is good news for researchers and practitioners who develop and/or use software tools that implement various forms of bisimilarity checking. However, the size of the labelled transition systems on which the above-mentioned algorithms are run in practice is often huge. It is therefore natural to ask oneself whether one can devise efficient *parallel* algorithms for bisimilarity checking. Such algorithms might exploit the parallelism that is becoming increasingly available on our computers to speed up checking whether two processes are bisimilar or not. It would be very satisfying, as well as practically useful, if algorithms for checking bisimilarity could run on a highly parallel machine with a running time that is proportional to the logarithm of the size of the state space of the input labelled transition system using a feasible number of processors. But is an algorithm meeting the above desiderata likely to exist?

A formal, negative answer to the above question has been given by Balcázar, Gabarró and Santha, who showed in (Balcázar, Gabarró and Santha, 1992) that deciding strong bisimilarity between finite labelled transition systems is P-complete—this means that it is one of the 'hardest problems' in the class P of problems solvable in deterministic polynomial time. P-complete problems are of interest because they appear to lack highly parallel solutions. So showing that an algorithmic problem is P-complete is interpreted as strong evidence that the existence of an efficient parallel solution for it is unlikely. See, for instance, the book (Greenlaw, Hoover and Ruzzo, 1995) for much more information on P-completeness.

In the remainder of this note, we shall present the main ideas behind the proof of the above mentioned result, which we reiterate below in the form of a theorem.

Theorem 1 [Balcázar, Gabarró and Santha 1992] The problem of deciding strong bisimilarity between two states in a finite labelled transition system is P-complete.

This theorem can be proved by offering a reduction from a problem that is known to be P-complete to the problem of checking strong bisimilarity in a finite labelled transition system. The type of reduction used in the argument is technically called an *NC-reduction*. We will not define NC-reductions formally; suffice it to say here that, roughly speaking, an NC-reduction is a reduction that can be computed using 'uniform' boolean circuits that have polynomial size and polylogarithmic depth in the size of the input. (Intuitively, the depth of a circuit indicates the amount of time that is needed to compute the reduction and its size describes the amount of hardware resources used in the computation.)

A *boolean circuit* is a directed, acyclic, node-labelled graph of largest indegree two. (In what follows, we occasionally use nodes with indegree larger than two. These nodes should be interpreted as short-hands for circuits made up of nodes with indegree at most two.) The nodes with indegree zero are the inputs to the circuit and each node of indegree n computes the n-ary boolean function that labels it. The nodes of outdegree zero are the outputs of the circuit. The *size* of a circuit is the number of nodes in it, and its *depth* is the length of a longest path from an input to an output. A boolean circuit computing the boolean function

$$(\neg x_1) \wedge (x_1 \vee (\neg x_2))$$

is depicted in Figure 1.

A *monotone alternating circuit* is a circuit whose nodes are divided into levels, so that the inputs to a node at a given level are the outputs of nodes at the previous level.

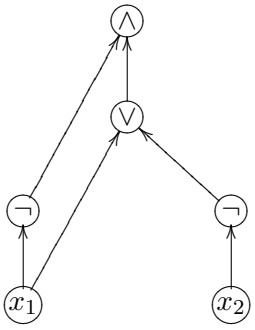

Figure 1: An example boolean circuit

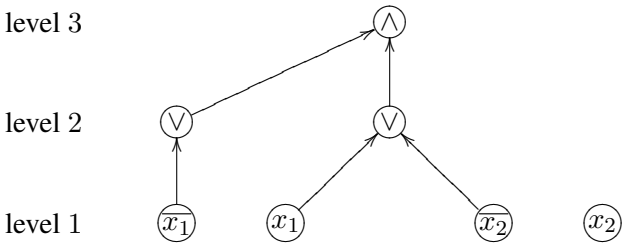

Figure 2: A monotone alternating circuit

The circuit contains only nodes labelled with $\vee$ and $\wedge$; all nodes at the same level in the circuit are of the same kind, and the two kinds of nodes alternate. Without loss of generality, we may assume that the second level of such a circuit consists of $\vee$-labelled nodes. To overcome the absence of $\neg$-labelled nodes, the circuit receives each input together with its negation. Finally, nodes with indegree one that are labelled with $\vee$ and $\wedge$ behave like the identity function.

A monotone alternating circuit computing the boolean function

$$(\neg x_1) \wedge (x_1 \vee (\neg x_2))$$

is depicted on Figure 2. In that figure, nodes labelled $\overline{x_1}$ and $\overline{x_2}$ stand for the negation of the inputs x_1 and x_2, respectively.

It is well-known that the so-called *Circuit Value Problem for Monotone Alternating Circuits*—that is, the problem of computing the value of a monotone, alternating circuit with one output given a collection of values for its inputs—is P-complete (Greenlaw et al., 1995). In order to prove Theorem 1 it therefore suffices to show that the Circuit Value Problem for Monotone Alternating Circuits reduces to the problem of checking bisimilarity between two distinguished states in a finite labelled transition system that is constructed from the circuit.

Rather than giving the construction in general, we exemplify it on the circuit depicted on Figure 2. From that monotone alternating circuit, we construct a finite labelled transition system in three steps.

1. We observe that the circuit on Figure 2 has three levels. We therefore build a so-called *alternating pattern* A_3 as depicted on Figure 3.

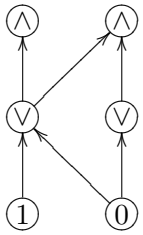

Figure 3: The alternating pattern A_3

Note that the boolean value of the leftmost nodes of A_3 is 1, whereas that of the rightmost nodes is 0.

The general procedure for constructing the alternating pattern A_m for each $m \geq 1$ is as follows.

> For each $m \geq 1$, the monotone, alternating circuit A_m is the circuit with m levels, each consisting of two nodes (one with boolean value 0 and the other with value 1), that is defined as follows.
>
> - A_1 consists of two nodes, labelled with the boolean constants 0 and 1.
> - For $m > 1$, the $\vee$-labelled nodes in A_m whose value is 0 are the targets of an edge from the single node at the previous level that has value 0; on the other hand, the $\vee$-labelled nodes in A_m whose value is 1 are the targets of an edge from both nodes at the previous level.
> - For $m > 1$, the $\wedge$-labelled nodes in A_m whose value is 1 are the targets of an edge from the single node at the previous level that has value 1; on the other hand, the $\wedge$-labelled nodes in A_m whose value is 0 are the targets of an edge from both nodes at the previous level.

We invite you to check that A_3 is indeed generated by this procedure.

2. The pattern A_3 is combined with the circuit on Figure 2 as follows.

   > Each node at level ℓ, with $1 < \ell \leq 3$, in the original circuit receives an incoming edge from a node at the previous level in the alternating pattern A_3. If the node is labelled $\vee$, then the edge comes from the node in A_3 at level $\ell - 1$ whose value is 0. Conversely, if the node is labelled $\wedge$, then the edge comes from the node in A_3 at level $\ell - 1$ whose value is 1.

   The resulting boolean circuit is depicted on Figure 4.

   Note that the above construction does not change the truth value of any of the nodes in the original circuit (regardless of the value of its inputs) and in the alternating pattern. Moreover, regardless of the value of the inputs of the original monotone alternating circuit,

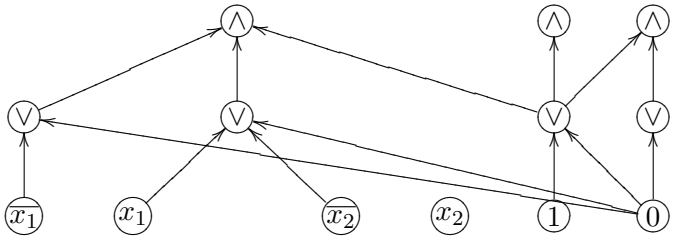

Figure 4: The combined circuit

- every $\vee$-labelled node in the resulting boolean circuit has at least one input with value 0 and
- every $\wedge$-labelled node has at least one input with value 1.

3. Finally, we transform the boolean circuit on Figure 4 into a finite, acyclic labelled transition system over a one-letter alphabet. The states in the labelled transition system are

   - the nodes in the circuit on Figure 4 and
   - three auxiliary states at level 0 associated with the three inputs of the combined circuit that evaluate to 1 with the given assignment of truth values to the inputs of the circuit on Figure 2.

   The transitions in the labelled transition system are defined as follows.

   - There is a transition for each edge in the circuit. However, the transition goes from the node that is the target of the edge to the node that is the source of the edge.
   - There is a transition from each of the three nodes on the first level that evaluates to 1 to its accompanying auxiliary state at level 0.

   Finally, we specify two distinguished states p and q in the labelled transition system that we have constructed. State p is the one that corresponds to the output node of the circuit on Figure 2. State q instead is the one that corresponds to the output node of A_3 that evaluates to 1.

The labelled transition system that results from this construction when $x_1 = x_2 = 0$, with states p and q marked, is depicted on Figure 5, where we have omitted the label from the transitions since they all carry the same label.

As an exercise, we invite you to check that the two states p and q are strongly bisimilar by exhibiting a strong bisimulation containing the pair (p, q). Indeed, the original boolean circuit evaluates to 1 when $x_1 = x_2 = 0$, and the construction that we have exemplified above guarantees that p and q are bisimilar in this case.

While proving that p and q are strongly bisimilar, you will see that the reason for adding the edges from the nodes in the circuit to the alternating pattern is that this guarantees that each $\vee$-labelled node has a transition to a node whose value is 0. For instance, the leftmost $\vee$-labelled node on Figure 5 must be bisimilar to the $\vee$-labelled node reachable from q. In order for this to hold, the leftmost $\vee$-labelled node must

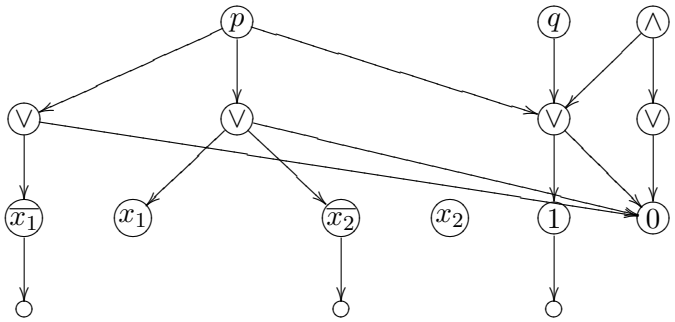

Figure 5: The labelled transition system when $x_1 = x_2 = 0$

have a transition reaching a node at level 1 whose value is 0. This transition would not exist without the outgoing edge leading to the alternating pattern. You should find it easy to construct an example circuit showing the need of having a transition to a node whose value is 1 from each $\wedge$-labelled node. (Do so!)

Exercise 1 *Construct the labelled transition system that results from the above construction when $x_1 = x_2 = 1$ and show that p and q are* not *bisimilar in that case.* ♦

In general, it can be shown that, given any assignment of boolean values to the inputs of the monotone alternating circuit on Figure 2, states p and q in the labelled transition system constructed following the above-mentioned procedure are strongly bisimilar if, and only if, the circuit evaluates to 1 when fed with those input values. Indeed, as you might have already found out after working through the suggested exercises, if the original circuit evaluates to 1 for the given input values, then each $\vee$-labelled state stemming from a node in the original circuit that is reachable from p must also evaluate to 1. Therefore, for each such state, at least one of the input states to which it is connected must have value 1. That input state hence exhibits a transition to an auxiliary state like the 1-labelled node in the alternating pattern. This is enough to ensure that p and q are bisimilar.

Conversely, if the original circuit evaluates to 0 for a given assignment of values to its inputs, then at least one the two leftmost $\vee$-labelled states reachable from p must have value 0. That state is not bisimilar to the single $\vee$-labelled state reachable from q since it cannot perform two transitions in a row.

The above argument can be generalized to any instance of the Circuit Value Problem for Monotone Alternating Circuits, establishing Theorem 1.

Exercise 2 (For the keenest) *Prove Theorem 1 in general.*

Hint: Show, first of all, that the relation $\mathcal{R}$ defined below is a strong bisimulation over the labelled transition system generated by the reduction we have sketched above.

$s \mathcal{R} t$ if, and only if, one of the following conditions holds:

- *s and t are both auxiliary states,*
- *s and t are states at the same level that evaluate to 0 or*

- *s and t are states at the same level that evaluate to* 1.

Next, use this fact to conclude that if the original circuit evaluates to 1 *for the given input values, then* p *and* q *are bisimilar.*

To complete the proof, it suffices to show that if s *and* t *are states at the same level such that* s *evaluates to* 0 *and* t *evaluates to* 1*, then* s *and* t *are not bisimilar. (Why?)* ♦

Since the construction we have sketched above produces finite, acyclic labelled transition systems over one label, Theorem 1 can be strengthened as follows.

Theorem 2 The problem of deciding strong bisimilarity between two states in a finite, acyclic labelled transition system over a one-letter alphabet is P-complete.

Moreover, since the single letter used in constructing the labelled transition system can be chosen to be different from τ, and observation equivalence, observational congruence, branching bisimilarity and rooted branching bisimilarity coincide with strong bisimilarity over τ-free labelled transition systems, we have the following result.

Theorem 3 The problem of deciding observation equivalence, observational congruence, branching bisimilarity and rooted branching bisimilarity between two states in a finite, acyclic labelled transition system over a one-letter alphabet is P-complete.

Acknowledgement We thank Arnar Birgisson for his careful proof reading and his suggestions that led to several improvements in the note.

References

Balcázar, J. L., Gabarró, J. and Santha, M. (1992). Deciding bisimilarity is P-complete, *Journal of Formal Aspects of Computing Science* **4**(6A): 638–648.

Glabbeek, R. van and Weijland, W. (1996). Branching time and abstraction in bisimulation semantics, *Journal of the ACM* **43**(3): 555–600.

Greenlaw, R., Hoover, H. J. and Ruzzo, W. R. (1995). *Limits to Parallel Computation: P-Completeness Theory*, Oxford University Press.

Groote, J.F. and Vaandrager, F. (1990). An efficient algorithm for branching bisimulation and stuttering equivalence, *Proceedings 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, Vol. 443 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 626–638.

Hunt, H. B., Rosenkrantz, D. J. and Szymanski, T. G. (1976). On the equivalence, containment, and covering problems for the regular and context-free languages, *Journal of Computer and System Sciences* **12**: 222–268.

Kanellakis, P. C. and Smolka, S. A. (1990). CCS expressions, finite state processes, and three problems of equivalence, *Information and Computation* **86**(1): 43–68.

Milner, R. (1989). *Communication and Concurrency*, Prentice-Hall International, Englewood Cliffs.

Paige, R. and Tarjan, R. E. (1987). Three partition refinement algorithms, *SIAM Journal of Computing* **16**(6): 973–989.