Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT
Theoretical Computer Science 311 (2004) 221–256

Theoretical Computer Science

www.elsevier.com/locate/tcs

An efficient algorithm for computing bisimulation equivalence

Agostino Dovier*, Carla Piazza, Alberto Policriti

Dip. di Matematica e Informatica, Univ. di Udine, Via Delle Scienze 206, 33100 Udine, Italy

Received 16 July 2002; received in revised form 22 May 2003; accepted 2 June 2003
Communicated by T.A. Henzinger

Abstract

We propose an efficient algorithmic solution to the problem of determining a *Bisimulation* Relation on a finite structure working both on the explicit and on the implicit (symbolic) representation. As far as the explicit case is concerned, starting from a set-theoretic point of view we propose an algorithm that optimizes the solution to the *Relational Coarsest Partition Problem* given by Paige and Tarjan (SIAM J. Comput. 16(6) (1987) 973); its use in model-checking packages is also discussed and tested. For well-structured graphs our algorithm reaches a linear worst-case behaviour. The proposed algorithm is then re-elaborated to produce a *symbolic* version.
© 2003 Elsevier B.V. All rights reserved.

Keywords: Bisimulation; Non-well-founded sets; Rank-based methods; Verification; OBDDs

1. Introduction

It is difficult to accurately list all the fields in which, in one form or another, the notion of *bisimulation* was introduced and now plays a central rôle: Modal Logic, Concurrency Theory, Set Theory, Formal Verification, etc. In Modal Logic the notion was introduced by van Benthem [5] as an equivalence principle between Kripke structures. In Concurrency Theory it was introduced by Milner and Park for testing observational equivalence of the Calculus of Communicating Systems (CCS). In particular, in [41]

* Corresponding author.
E-mail addresses: dovier@dimi.uniud.it (A. Dovier), piazza@dimi.uniud.it (C. Piazza), policriti@dimi.uniud.it (A. Policriti).

0304-3975/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0304-3975(03)00361-X

a previous notion of automata simulation by Milner is refined in the context of omega regular languages for concurrency, while in [38] (weak and strong) bisimulation is proposed. In Set Theory, it was introduced by Forti and Honsell [24] as a natural principle replacing extensionality in the context of non-well-founded sets (also known as hypersets). As far as Formal Verification is concerned (cf. [13]), several existing verification tools make use of bisimulation in order to minimize the states' space of systems' description [8,15,21,43]. The reduction of the number of states is important both in compositional and in non-compositional Model Checking. Bisimulation serves also as a means of checking equivalence between transition systems. In the context of security many non-interference properties are based on checking bisimulation between systems [23].

The bisimulation problem is equivalent to determine the *coarsest partition* of a set, *stable* with respect to a given relation. Hopcroft [33] presents an algorithm for the minimization of the number of states in a given finite state automaton: the problem is equivalent to that of determining the coarsest partition of a set *stable* with respect to a finite set of functions. A variant of this problem is studied in [39], where it is shown how to solve it in linear time in case there is only one function. Paige and Tarjan [40] solved the problem in the general case in which the stability requirement is relative to a relation E (on a set N) with an algorithm whose complexity is $O(|E| \log |N|)$.

The main feature of the linear solution to the single-function coarsest partition problem (cf. [39]) is the use of a *positive* strategy in the search for the coarsest partition: the starting partition is the partition with singleton classes and the output is built via a sequence of steps in which two or more classes are merged. Instead, Hopcroft's solution to the (more difficult) many-functions coarsest partition problem is based on a (somehow more natural) *negative* strategy: the starting partition is the input partition and each step consists of the split of all those classes for which the stability constraint is not satisfied. The interesting feature of Hopcroft's algorithm lies in its use of a clever ordering (the so-called "process the smallest half" ordering) for processing classes that must be used in a split step. Starting from an adaptation of Hopcroft's idea to the relational coarsest partition problem, Paige and Tarjan succeeded in obtaining their fast solution [40].

In this paper we present a procedure that integrates positive and negative strategies to obtain the algorithmic solution to the bisimulation problem and hence to the relational coarsest partition problem. The strategy we develop is driven by the set-theoretic notion of *rank* of a set. The algorithm we propose uses [39,40] as subroutines and terminates in linear time in many cases, for example when the input problem corresponds to a bisimulation problem on acyclic graphs (well-founded sets). The algorithm operates in linear time in other cases as well and, in any case, it runs at a complexity less than or equal to that of [40]. Moreover, the partition imposed by the rank allows to process the input without storing the entire structure in memory at the same time. This allows (potentially) to deal with larger graphs than those treatable using a Paige and Tarjan-like approach.

When memory requirements become more stringent, the data-structure called *Ordered Binary Decision Diagrams* (*OBDDs*) are commonly used [10,37]. An OBDD allows an implicit (symbolic) representation of a graph $\langle N, E\rangle$ based on the encoding of the

characteristic function associated to the relation E. Rather than *explicitly* manipulate single states, this data-structure allows to deal with subsets of the set of states. In the final part of this paper we present a procedure that allows to compute the rank of a graph working on its symbolic representation. This procedure avoids the construction of the strongly connected components, used in the definition of rank. On this ground, we develop a symbolic rank-based bisimulation algorithm.

The paper is organized as follows: in Section 2 we recall the main related results. In Section 3 we introduce the set-theoretic formulation of the bisimulation problem. The subsequent Section 4 contains the algorithm for the well-founded case. Section 5 presents the basic idea of our proposed algorithm, while its optimizations are explained in Section 6. In Section 7 we show how our results and methods can be adapted to the *multi-relational* coarsest partition problem (i.e., bisimulation on labeled graphs) and in Section 8 we discuss some testing results. In Section 9 we develop the symbolic version of our rank-based algorithm. Finally, some conclusions are drawn in Section 10.

Some results reported in this paper have been preliminarily presented in [19,20].

2. Related works

The first significant result related to the algorithmic solution of the bisimulation problem is in [33], where Hopcroft presents an algorithm for the minimization of the number of states in a given finite state automaton. The problem is equivalent to that of determining the coarsest partition of a set *stable* with respect to a finite set of functions. A variant of this problem is studied in [39], where it is shown how to solve it in linear time in case of a single-function. Finally, Paige and Tarjan [40] solved the problem for the general case (which is the same as computing bisimulation equivalence) in which the stability requirement is relative to a relation E (on a set N) with an algorithm whose complexity is $O(|E| \log |N|)$. Kannellakis and Smolka [34] notice that the algorithm by Paige and Tarjan [40] can be used to determine the maximum bisimulation over a graph $G = \langle N, E \rangle$. Bouajjani et al. [7] propose an algorithm for the relational coarsest partition problem tailored for the context of the so-called on-the-fly Model Checking. Precisely, at each iteration the algorithm stabilizes only the reachable blocks with respect to all blocks. Lee and Yannakakis [35] improve this method by using only reachable blocks to stabilize the reachable blocks.

In the *symbolic* case (cf. Section 9) a popular bisimulation algorithm is the one in [9] by Bouali and de Simone. This algorithm implements the naïve negative strategy optimizing the Boolean operations involved: first, the set of reachable nodes R is computed through a symbolic visit of the graph, then, starting from $R \times R$ all the pairs $\langle u, v \rangle$ for which it is possible to prove that u is not bisimilar to v are removed. In [9] experimental results about the performances of the algorithm are presented, while there is no through discussion of its complexity in terms of basic symbolic operations.

Fisler and Vardi [22] analyze the complexity of the symbolic versions of the algorithms of Paige and Tarjan [40], Bouajjani et al. [7], and Lee and Yannakakis [35]. In particular, they determine the number of basic symbolic operations involved in each iteration of the three algorithms and conclude, through experimental results, that an

optimized version of the algorithm in [40], which splits only reachable blocks, performs better than the other two algorithms, since it gains from the *right* choice of the splitters.

In [27] Hennessy and Lin use the adjective *symbolic* in a different setting. They reexamine bisimulation equivalence for value-passing process calculi and generalize the standard notion of labeled transition graph using *symbolic actions*. In their setting, they compute bisimulations on infinite graphs which have a finite symbolic representation. Always in the context of process languages we mention the work by Hirshfeld et al. [29] on normed basic parallel processes, which are a particular class of infinite-state processes. The authors present an algorithm to decide bisimulation on normed basic parallel processes avoiding to compute all the states generated by the parallel compositions.

In [16] Cleaveland and Sokolsky show how bisimulation can be used to calculate other semantic relations like weak bisimulation and branching bisimulation.

Several tools developed to analyze systems implement one or more bisimulation algorithms. In general, in the case of explicit-state representation, the underlying algorithm used is the one proposed by Kanellakis and Smolka [39], while Bouali and de Simone's algorithm [9] is used in the case of symbolic representation. The verification environment XEVE [8] provides bisimulation tools which can be used for both minimization and equivalence test. The Concurrency Workbench (CWB) [14] tests bisimulation using techniques based on the Kanellakis and Smolka algorithm. The Compositional Security Checker (CoSec) [23] exploits the bisimulation algorithm implemented in CWB in order to test information flow security properties. In the Concurrency Workbench of the New Century (CWB-NC) [15] the underling bisimulation algorithm is the one by Paige and Tarjan. CÆSAR/ALDÉBARAN Development Package (CADP) [21] supports both explicit bisimulation based on Paige and Tarjan' algorithm and symbolic OBDD-based algorithms.

As for the criticism on the use of bisimulation algorithms in formal verification, Fisler and Vardi observe in [22] that "bisimulation minimization does not appear to be viable in the context of invariance verification", but in the context of compositional verification it "makes certain problems tractable that would not be so without minimization" [2,42].

3. The problem: a set-theoretic perspective

One of the main features of intuitive (naïve) Set Theory is the well-foundedness of membership. As a consequence, standard axiomatic set theories include the *foundation* axiom that forces the membership relation not to form cycles or infinite descending chains. In the 1980s the necessity to consider theories that do not assume this strong constraint (re-)emerged in many communities; hence various proposals for (axiomatic) non-well-founded set (hyperset) theories (and universes) were developed. Probably the first one was [24] by Forti and Honsell. Following Barwise and Moss (cf. [3]) we can say that the book [1] by Aczel can be considered as the definitive reference on non-well-founded sets. More recent references on this topic are [4,12].

Sets can be seen as nothing but *accessible pointed graphs* (cf. Definition 3.1). Edges represent membership, namely $\langle m, n\rangle$ (also denoted as $m \to n$) means that m has n as an element, and the nodes in the graph denote all the sets which contribute in the construction of the represented set.

Definition 3.1. An accessible pointed graph (apg) $\langle G, v\rangle$ is a directed graph $G = \langle N, E\rangle$ together with a distinguished node $v \in N$ (the point) such that all the nodes in N are reachable from v.

The resulting set-theoretic semantics for apg's, developed in [1], is based on the natural notion of *picture* of an apg. In the picturing process, each node of an apg is uniquely associated to a set. We say that an apg *represents* the set associated to its point.

The extensionality axiom—saying that two objects are equal if and only if they contain exactly the same elements—is the standard criterion for establishing equality between sets. If extensionality is assumed it is immediate to see that, for example, different acyclic graphs can represent the same set. However, extensionality leads to a cyclic argument (no wonder!) whenever one tries to apply it as a test to establish whether two cyclic graphs represent the same non-well-founded set (*hyperset*). To this end a condition (*bisimulation*) on apg's can be stated in accordance with extensionality: two apg's are bisimilar if and only if they are representations of the same set.

Definition 3.2. Given two graphs $G_1 = \langle N_1, E_1\rangle$ and $G_2 = \langle N_2, E_2\rangle$, a bisimulation between G_1 and G_2 is a relation $b \subseteq N_1 \times N_2$ such that
(1) $u_1 b\, u_2 \wedge \langle u_1, v_1\rangle \in E_1 \Rightarrow \exists v_2 \in N_2 (v_1 b\, v_2 \wedge \langle u_2, v_2\rangle \in E_2)$
(2) $u_1 b\, u_2 \wedge \langle u_2, v_2\rangle \in E_2 \Rightarrow \exists v_1 \in N_1 (v_1 b\, v_2 \wedge \langle u_1, v_1\rangle \in E_1)$.
Two apg's $\langle G_1, v_1\rangle$ and $\langle G_2, v_2\rangle$ are bisimilar if and only if there exists a bisimulation b between G_1 and G_2 such that $v_1 b\, v_2$.

We can now say that two hypersets are equal if their representations are bisimilar. For example the apg $\langle\langle\{v\}, \emptyset\rangle, v\rangle$ represents the empty set $\emptyset$. The hyperset Ω, i.e. the unique hyperset which satisfies the equation $x = \{x\}$ (see [1]), can be represented using the apg $\langle\langle\{v\}, \{\langle v, v\rangle\}\rangle, v\rangle$. Any graph such that each node has at least one outgoing edge can be shown to be a representation of Ω. It is clear that for each set there exists a collection of apg's which are all its representations. It is always the notion of bisimulation which allows us to find a *minimum* representation (there are no two nodes representing the same hyperset). Given an apg $\langle G, v\rangle$ that represents a set S, to find the minimum representation for S it is sufficient to consider the maximum bisimulation $\equiv$ between G and G.

Proposition 3.3. *Given an* apg $\langle G, v\rangle$ *the maximum bisimulation* $\equiv$ *between* G *and* G *always exists it is unique, and is an equivalence relation over the set of nodes of* G.

Proof. Since the number of relations on the nodes of a graph is finite, the result follows by the fact that the union of two bisimulation is a bisimulation itself. Thus,

$\equiv$ is the union of all the bisimulations on a graph. It is immediate to check that it is an equivalence relation. □

Thus, the minimum representation of a set S denoted by an apg $\langle G, v\rangle$ that represents it, is the apg $\langle G/\equiv, [v]_{\equiv}\rangle$ where $G/\equiv$ is the graph obtained from G by collapsing all equivalent nodes into a single one, and $[v]_{\equiv}$ is the node where v has been mapped. This graph is usually called *bisimulation contraction of* G.

An equivalent way to present the problem is to define the concept of bisimulation as follows.

Definition 3.4. Given a graph $G = \langle N, E\rangle$, a bisimulation on G is a relation $b \subseteq N \times N$ such that

(1) $u_1 b\, u_2 \wedge \langle u_1, v_1\rangle \in E \Rightarrow \exists v_2 (v_1 b\, v_2 \wedge \langle u_2, v_2\rangle \in E)$
(2) $u_1 b\, u_2 \wedge \langle u_2, v_2\rangle \in E \Rightarrow \exists v_1 (v_1 b\, v_2 \wedge \langle u_1, v_1\rangle \in E)$.

A bisimulation on G is nothing but a bisimulation between G and G. Thus it is not ambiguous to define $\equiv$ as the maximum bisimulation on G.

The problem of recognizing if two graphs are bisimilar and the problem of determining the maximum bisimulation on a graph are equivalent.

Proposition 3.5. *Two disjoint* apg's $A_1 = \langle\langle N_1, E_1\rangle, v_1\rangle$ *and* $A_2 = \langle\langle N_2, E_2\rangle, v_2\rangle$ *are bisimilar if and only if* $v_1 \equiv v_2$, *where* $\equiv$ *is the maximal bisimulation on* $A_3 = \langle\langle N_1 \cup N_2 \cup \{\mu\}, E_1 \cup E_2 \cup \{\langle \mu, v_1\rangle, \langle \mu, v_2\rangle\}\rangle, \mu\rangle$, *with* μ *a new node.*

Proof. The ($\leftarrow$) direction holds by definition. Now, assume that A_1 and A_2 are bisimilar. Let b be the union of all the bisimulations between them. By definition, it holds that $v_1 b v_2$. It is immediate to check that $b' = b \cup \{(\mu, \mu)\}$ is a bisimulation on A_3. $v_1\, b'\, v_2$ implies $v_1 \equiv v_2$, since $\equiv$ is the maximum one. □

The problem faced in this paper is that of finding the minimum graph bisimilar to a given graph, that is, the *bisimulation contraction* of a graph. To solve it, we use another characterization of the same problem based on the notion of *stability*:

Definition 3.6. Let E be a relation on the set N, E^{-1} its inverse relation, and P a partition of N. P is said to be *stable* with respect to E iff for each pair B_1, B_2 of blocks of P, either $B_1 \subseteq E^{-1}(B_2)$ or $B_1 \cap E^{-1}(B_2) = \emptyset$.

We say that a partition P *refines* a partition Q if each block (i.e., class) of P is contained in a block of Q. The above definition suggest a natural operation on partitions. A class B of P *splits* a class C of P if C is replaced in P by $C_1 = C \cap E^{-1}(B)$ and $C_2 = C \backslash C_1$; if C_1 or C_2 is empty, it is not added in P. A partition P is *split* by the class B if each class C of P is split by B. The split operation produces a refinement of a partition P; if P is stable with respect to E, the split operation returns P.

Definition 3.7. Let E be a relation on the set N and Q a partition of N. The coarsest stable partition problem is the problem of finding the coarsest partition P refining Q that is stable with respect to E. If Q is not given, we assume it is the trivial partition $\{N\}$.

We will first prove that this problem, that emerged in automata minimization, is equivalent to the problem of finding the bisimulation contraction of a graph.

Proposition 3.8. *Let $G = \langle N, E\rangle$ be a graph.*

(i) *Let P be a partition of its nodes stable with respect to E. Then b_P defined as*:

$$u\, b_P\, v \text{ iff } \exists B \in P(u \in B \wedge v \in B)$$

is a bisimulation on G.

(ii) *Let b be a bisimulation on G which is also an equivalence relation. Then $P_b = \{[u]_b : u \in N\}$, where $[u]_b = \{v \in N : ubv\}$, is a partition stable with respect to E.*

Proof.

(i) We prove that b_P is a bisimulation on G. Let us prove property (1) of Definition 3.4. Let u_1, u_2 be such that $u_1\, b_P\, u_2$. This means that there is a class $B_1 \in P$ such that $u_1 \in B_1$ and $u_2 \in B_1$. Assume there is $v_1 \in N$ such that $\langle u_1, v_1\rangle \in E$. Let B_2 be the class such that $v_1 \in B_2$. Since P is stable with respect to E and $E^{-1}(B_2)$ is not empty (it contains u_1), this means that $B_1 \subseteq E^{-1}(B_2)$. Thus, $u_2 \in E^{-1}(B_2)$, and this implies that there is $v_2 \in B_2$ such that $\langle v_1, v_2\rangle \in E$. By definition of b_P, $v_1\, b\, v_2$.
The proof of (2) of Definition 3.4 is similar.

(ii) By contradiction, assume that P_b is not stable with respect to E. This means that there are blocks B_1 and B_2 and two nodes u, v such that

$$u \in B_1 \backslash E^{-1}(B_2) \wedge v \in B_1 \cap E^{-1}(B_2)$$

This implies that there is a node $v' \in B_2$ such that $\langle v, v'\rangle \in E$ but no node u' bisimilar to v' (i.e., in B_2) can be reached by an edge from u. Thus, $u \not b\, v$. □

Corollary 3.9. *Let $G = \langle N, E\rangle$ be a graph. Computing the maximum bisimulation $\equiv$ on G or finding the coarsest stable partition of N stable with respect to E are equivalent problems.*

Proof. Immediate from Propositions 3.3 and 3.8. □

Given a set N, k relations $E_1, \ldots, E_k$ on N, and a partition P of N, the *multi-relational coarsest partition* problem consists of finding the coarsest refinement of P which is stable with respect to $E_1, \ldots, E_k$. As noted in [34], the algorithm of [40] that determines the coarsest partition of a set N stable with respect to k relations solves exactly the problem of *testing if two states of an observable Finite States Process* (*FSP*) are strongly equivalent. Our bisimulation problem is a particular case of *observable FSPs strong equivalence* problem ($k = 1$). In Section 7 we show how the case of bisimulation over a labeled graph (*multi-relational* case) can be linearly reduced

to our bisimulation problem. This means that the problem of finding the bisimulation contraction of a graph is equivalent to the multi-relational coarsest partition problem.

4. The well-founded case

We start by considering the case of acyclic graphs (well-founded sets). As done for the minimization of Deterministic Finite Automata, it is possible to determine the coarsest partition P stable with respect to E via the computation of a suitable greatest fixpoint. A "negative" (and blind with respect to the relation) strategy is applicable: start with the coarsest partition $P=\{N\}$, choose a class $B\in P$ (the splitter) and split all the classes using B whenever P is *not* stable. The complexity of the algorithm, based on a negative strategy, presented in [40] for this problem is $O(|E|\log|N|)$. The main "ingredient" in that algorithm is the use of Hopcroft's "process the smallest half" ordering in the choice of the block to be used to split.

We can take advantage of the set-theoretic point of view of the problem in order to develop a selection strategy for the splitters depending on the relation E. In particular, making use of the ordering induced by the notion of *rank* we start from a partition which is a refinement of the coarsest one; then we choose the splitters using the ordering induced by the rank. These two ingredients allow to obtain a linear-time algorithm. Those who are familiar with k-layered DFAs [32] can read our algorithm for the well-founded case as a generalization of the minimization algorithm for k-layered DFA. In the well-founded case we admit that a node at the ith layer may reach a node at the jth layer with $j>i$.

Definition 4.1. Let $G=\langle N,E\rangle$ be a directed acyclic graph. The *rank* of a node n is recursively defined as follows:

$$\begin{cases} rank(n) = 0 & \text{if } n \text{ is a leaf} \\ rank(n) = 1+\max\{rank(m):\ \langle n,m\rangle\in E\} & \text{otherwise} \end{cases}$$

The notion of *rank* determines a partition which is coarser than the maximum bisimulation.

Proposition 4.2. *Let u and v be nodes of an acyclic graph G. If $u\equiv v$, then $rank(u)=rank(v)$.*

Proof. By induction on $rank(u)$. □

The function *rank* can easily be computed in time $O(|N|+|E|)$ by one depth-first visit of the graph.

The converse of the above proposition, of course, is not true:

Example 4.3. Consider the graph in Fig. 1. The nodes u and v have both rank 2, but they are not bisimilar. The node u represents the set $\{\{\emptyset\}\}$, while the node v represents $\{\{\emptyset\},\emptyset\}$.

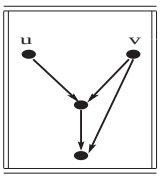


Fig. 1. Non-bisimilar nodes with the same rank.

Let P be a partition of N such that for each block B in P it holds that $u, v \in B$ implies $rank(u) = rank(v)$; then every refinement of P fulfills the same property. Hence, we can assign to a block B the rank of its elements.

Algorithm 1 (Well-founded case).

(1) **for** $n \in N$ **do compute** $rank(n)$; — compute the rank
(2) $\rho := \max\{rank(n) : n \in N\}$;
(3) **for** $i = 0, \ldots, \rho$ **do** $B_i := \{n \in N : rank(n) = i\}$;
(4) $P := \{B_i : i = 0, \ldots, \rho\}$; — P is the partition to be refined
(5) **for** $i = 0, \ldots, \rho$ **do**
  (a) $D_i := \{X \in P : X \subseteq B_i\}$; — determine the blocks currently at rank i
  (b) **for** $X \in D_i$ **do**
    $G := \mathbf{collapse}(G, X)$; — collapse nodes at rank i
  (c) **for** $n \in N \cap B_i$ **do** — refine blocks at higher ranks
    for $C \in P$ **and** $C \subseteq B_{i+1} \cup \cdots \cup B_\rho$ **do**
      $P := (P \setminus \{C\}) \cup \{\{m \in C : \langle m, n\rangle \in E\}, \{m \in C : \langle m, n\rangle \notin E\}\}$;

Collapsing nodes $a_1, \ldots, a_k$, as in step (5(b)), consists in eliminating all nodes but a_1 and replacing all edges incident to $a_2, \ldots, a_k$ by edges incident to a_1. Despite the nesting of **for**-loops the following holds.

Proposition 4.4. *The algorithm for the well-founded case correctly computes the bisimulation contraction of its input acyclic graph* $G = \langle N, E\rangle$ *and can be implemented so as to run in linear time* $O(|N| + |E|)$.

Proof. Proposition 4.2 ensures that the initial partition is correct. Now, we prove by induction on the i, that $rank(u) = rank(v) = i$ and $u \not\equiv v$ implies that u and v enter in different classes at the ith iteration.

For $rank(u) = rank(v) = 0$ the property holds trivially.

Ler $rank(u) = rank(v) = i > 0$. Observe that, by definition of bisimulation, if $u \not\equiv v$ then we are in one of the following two cases:

(1) there is $u' \in N$ such that $\langle u, u'\rangle \in E$ and u' is not bisimilar to any node reached by v, or, symmetrically
(2) there is $v' \in N$ such that $\langle v, v'\rangle \in E$ and is not bisimilar to any node reached by u.

By inductive hypothesis, non-bisimilar nodes at the lowest ranks have been already split and hence action (5c) will split nodes u and v.

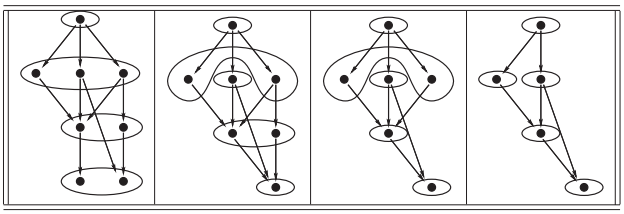


Fig. 2. Minimization process.

The complexity estimate is a consequence of the following facts:

- the *rank* can be computed by a depth-first visit;
- each node n is considered at most once in (5(b));
- the incoming edges are considered only once;
- for an efficient implementation of the collapsing procedure and of step (5c), it is sufficient to maintain the counter image $E^{-1}(n)$ of each node n. Notice that collapsing the nodes $n_1,\ldots,n_k$ into a single node can be done in time $O(E^{-1}(\{n_1,\ldots,n_k\}))$, and that the collapsing phase is done once per level;
- some suitable list manipulation is needed for storing partitions. □

An example of computation of the above algorithm can be seen in Fig. 2. In all the examples we present, the computation steps proceed from left to right.

5. Basic idea for the general case

The presence of cycles causes the usual notion of rank (cf. Definition 4.1) to be no longer adequate: an extension of this notion must be defined and such extension will be built on the notion of strongly connected component.

Definition 5.1. Given a graph $G=\langle N,E\rangle$, let $G^{\mathrm{scc}}=\langle N^{\mathrm{scc}},E^{\mathrm{scc}}\rangle$ be the graph obtained as follows:

$$N^{\mathrm{scc}}=\{c: c \text{ is a strongly connected component in } G\},$$
$$E^{\mathrm{scc}}=\{\langle c_1,c_2\rangle : c_1\neq c_2 \text{ and } (\exists n_1\in c_1)(\exists n_2\in c_2)(\langle n_1,n_2\rangle\in E)\}.$$

Given a node $n\in N$, we refer to the node of G^{scc} associated to the strongly connected component of n as $c(n)$.

Observe that G^{scc} is acyclic and if G is acyclic then G^{scc} is (isomorphic to) G itself.

We need to distinguish between the well-founded part and the non-well-founded part of a graph G.

Definition 5.2. Let $G=\langle N,E\rangle$ and $n\in N$. $G(n)=\langle N(n),E\restriction N(n)\rangle$ is the subgraph of G of the nodes reachable from n, where $E\restriction N(n)=E\cap(N(n)\times N(n))$. $WF(G)$, the well-founded part of G, is $WF(G)=\{n\in N : G(n) \text{ is acyclic}\}$.

Observe that $\langle G(n),n\rangle$ is an apg; $n\in WF(G)$ if and only if it denotes a well-founded set.

Definition 5.3. Let $G=\langle N,E\rangle$. The rank of a node n of G is defined as

$$\begin{aligned} rank(n) &= 0 \quad \text{if } n \text{ is a leaf in } G\\ rank(n) &= -\infty \quad \text{if } c(n) \text{ is a leaf in } G^{scc} \text{ and } n \text{ is not a leaf in } G\\ rank(n) &= \max(\{1+rank(m) : \langle c(n),c(m)\rangle\in E^{scc}, m\in WF(G)\}\cup\\ &\quad \{rank(m) : \langle c(n),c(m)\rangle\in E^{scc}, m\notin WF(G)\}) \quad \text{otherwise}\end{aligned}$$

$rank(n)$ is well-defined, since G^{scc} is always acyclic. Observe that the definition is consistent with Definition 4.1 for acyclic graphs. As a matter of fact, if G is acyclic then $G=G^{scc}$.

Nodes that are mapped into leaves of G^{scc} are either bisimilar to $\emptyset$ or to the hyperset Ω (cf. Section 3). For a non-well-founded node different from Ω the rank is 1 plus the maximum rank of a well-founded node reachable from it (i.e., a well-founded set in its transitive closure).

We have used the graph G^{scc} to explicitly provide a formal definition of the notion of rank. G^{scc} can be computed using Tarjan's classical algorithm (see [47]) essentially based on two depth-first visits of the graph. Once G^{scc} is known, it is immediate to compute the *rank* with one visit of G^{scc} and a further visit of G. All these tasks can be performed in time $O(|N|+|E|)$.

In order to speed up the process, we can swap the ordering of the visits of Tarjan's algorithm. More precisely, start with a depth-first visit of the graph $\langle N,E^{-1}\rangle$. Then use the finishing times $f[n]$ associated to the nodes by this visit to order (for decreasing values of $f[n]$) the nodes to perform the depth-first visit of $\langle N,E\rangle$. During this second visit it is easy to associate to each node n a Boolean value $WF[n]$ stating whether n is a well-founded node or not. Moreover, it is also possible to compute directly the *rank* of n. The correctness of this procedure follows from the fact that $rank(n)$ depends only on the nodes m such that $f[m]>f[n']$ for n' node in the same strongly connected component as n.

To sum up, the *rank* function can be computed by only two visits of the graph.

Proposition 5.4. *Let m and n be nodes of a graph G:*
(i) *$m\equiv\Omega$ if and only if $rank(m)=-\infty$;*
(ii) *$m\equiv n$ implies $rank(m)=rank(n)$.*

Proof. (i) Immediate from the characterization of Ω of [1]: a graph G rooted in m represents Ω if and only if each node has at least one outgoing edge.

(ii) If m and n are well-founded nodes, the result follows from Proposition 4.2.

If one node is well-founded and the other one is not, it is immediate to see that they cannot be bisimilar.

Assume now that both m and n are not well-founded. If $rank(m) = -\infty$ then the result follows from (i). Otherwise, since m is not well-founded, $rank(m) = h$ for some $h > 0$. Let a be a well-founded node reachable from m of rank $h-1$. Since $m \equiv n$, there exists a node b reachable from n such that $a \equiv b$. Since a and b are well-founded $rank(b) = rank(a)$ and this implies $rank(n) \geqslant rank(m)$. Symmetrically, starting from n, we can conclude that $rank(m) \geqslant rank(n)$ from which the thesis follows. □

The converse of Proposition 5.4(ii) is not true (Example 4.3 provides a counterexample also for this proposition). Moreover, observe that the rank of $c(n)$ in G^{scc} (that can be computed using Definition 4.1) is not necessarily equal to the rank of n in G.

Given a graph $G = \langle N, E\rangle$ with $\rho = \max\{rank(n) : n \in N\}$, we call the sets $B_{-\infty}, B_0, \ldots, B_\rho$, where $B_i = \{n \in N : rank(n) = i\}$, the *rank components* of G.

We can use the linear-time Algorithm 1 for the well-founded case in order to process the nodes in $WF(G)$ in the general case. Hence, we can assume that the input graph for the general case does not contain two different bisimilar well-founded nodes.

Algorithm 2 (General case).

(1) **for** $n \in N$ **do compute** $rank(n)$; — compute the rank
(2) $\rho := \max\{rank(n) : n \in N\}$;
(3) **for** $i = -\infty, 0, \ldots, \rho$ **do** $B_i := \{n \in N : rank(n) = i\}$;
(4) $P := \{B_i : i = -\infty, 0, \ldots, \rho\}$; — P is the partition to be refined
(5) $G :=$ **collapse**$(G, B_{-\infty})$; — collapse all the nodes of rank $-\infty$
(6) **for** $n \in N \cap B_{-\infty}$ **do** — refine blocks at higher ranks
  for $C \in P$ **and** $C \neq B_{-\infty}$ **do**
    $P := (P \setminus \{C\}) \cup \{\{m \in C : \langle m, n\rangle \in E\}, \{m \in C : \langle m, n\rangle \notin E\}\}$;
(7) **for** $i = 0, \ldots, \rho$ **do**
  (a) $D_i := \{X \in P : X \subseteq B_i\}$; — determine the blocks currently at rank i
    $G_i := \langle B_i, E \restriction B_i\rangle$; — isolate the subgraph of rank i
    $D_i :=$ **Paige–Tarjan**(G_i, D_i); — process rank i
  (b) **for** $X \in D_i$ **do**
    $G :=$ **collapse**(G, X); — collapse nodes at rank i
  (c) **for** $n \in N \cap B_i$ **do** — refine blocks at higher ranks
    for $C \in P$ **and** $C \subseteq B_{i+1} \cup \cdots \cup B_\rho$ **do**
      $P := (P \setminus \{C\}) \cup \{\{m \in C : \langle m, n\rangle \in E\}, \{m \in C : \langle m, n\rangle \notin E\}\}$;

In steps (1)–(4) we determine the function *rank* and we initialize a variable P representing the computed partition using it. The collapse operation (steps (5) and (7(b))) is as in the well-founded case. Splits of higher rank blocks is done in steps (6) and (7(c)). Step (7) is the core of the algorithm, where optimizations will take place. For each value i of the rank, we call the procedure of [40] on $G_i = \langle B_i, E \restriction B_i\rangle$, with a cost $O(|E \restriction B_i| \log |B_i|)$ and we update the partition P on nodes at ranks greater than i. From these observations:

Proposition 5.5. *The algorithm for the general case on input $G = \langle N, E\rangle$ correctly computes the bisimulation contraction of G, and can be implemented so as to run with a worst case complexity of* $O(|E| \log |N|)$.

Proof. From Proposition 5.4 we have that if $m \equiv n$, then m and n belong to the same block in the initial partition.

If $rank(m) = rank(n) = -\infty$, then from Proposition 5.4 we know that they are both bisimilar to Ω. Step (5) takes care of their collapse.

For the remaining cases, by induction on $rank(m)$, we prove that if $rank(m) = i$, then $m \equiv n$ if and only if at the ith iteration, after step (7(a)) there exists $X \in D_i$ such that $m, n \in X$.

If $rank(m) = 0$, then it must be the case that m is a leaf of G. At the beginning all the leaves of G are in B_0. After step (6) all the leaves of G still belong to the unique block B_0. This implies that the procedure **Paige–Tarjan** applied to G_0 does not split the block B_0. Hence, step (7(b)) collapses all the nodes at rank 0. This is equivalent to our thesis, since it is true that all the leaves of G are bisimilar (they all represent the empty set).

If $rank(m) = i > 0$ and we assume $m \equiv n$, then for all $m' \in m$ there exists $n' \in n$ such that $m' \equiv n'$ and vice-versa. Hence, by inductive hypothesis, they belong to the same block X of D_i at the beginning of the ith iteration of step (7). The correctness of the procedure **Paige–Tarjan** (cf. [40]) ensures that they still belong to the same block at the ith iteration after step (7(a)). The opposite direction is similar.

As far as complexity is concerned, we have already seen that step (1) can be executed in time $O(|N| + |E|)$. The complexity of steps (3) and (4) is clearly linear. Step (5) is performed as in the well-founded case, and thus, linear.

The complexity of step (6) is the same as the complexity of step (7c).

As for the complexity of step (7) at the ith loop, it corresponds to the complexity of the procedure **Paige–Tarjan** which is $O(|E \upharpoonright B_i| \log |B_i|)$. This is due to the fact that all the remaining sub-steps can be implemented at a cost which is linear in the size of G_i (cf. [33]); in other words (for some $c_1, c_2 \in \mathbb{N}$), the global cost is no worse than:

$$c_1(|N| + |E|) + \sum_{i=1}^{\rho} c_2(|E \upharpoonright B_i| \log |B_i|) = O(|E| \log |N|). \qquad \square \tag{1}$$

The complexity of the method sketched above is asymptotically equivalent to that of Paige and Tarjan. However, as for Algorithm 1, we take advantage of a refined initial partition and of a strategy to select blocks for splitting at higher ranks. In a specific rank, the negative strategy of Paige and Tarjan' algorithm is applied to the rank components which, in general, are much smaller than the global graph. In particular, for families of graphs such that ρ is $\Theta(|N|)$ and the size of the each rank component is bounded by a constant c the global cost becomes linear (cf. (1)).

6. Optimizations in the general case

In this section we develop some optimizations for our algorithm, that allow, in some cases, to reach a linear running time even for cyclic graphs. The optimizations described in Sections 6.1–6.4 are called *topological optimizations*. In Section 6.5 we explore the use of a new notion of rank.

6.1. No critical nodes

This optimization makes use of the **Paige–Tarjan–Bonic** procedure [39]. Such a procedure can be used in some cases to solve the coarsest partition problem in linear time adopting a "positive" strategy. Its integration with our algorithm produces a global strategy that can therefore be considered as a mix of positive and negative strategies.

Definition 6.1. A node n belonging to a rank component $B_i \subseteq N$ is said to be a *critical node* if $|\{m \in B_i : \langle n, m\rangle \in E\}| > 1$.

Critical nodes are called *multiple nodes* in [20]. Whenever B_i has no critical nodes, we can replace the call to **Paige–Tarjan** in step (7(a)) with a call to **Paige–Tarjan–Bonic**. This allows us to obtain a *linear time* performance at rank i (in (1) the term $c_2(|E \restriction B_i| \log |B_i|)$ can be replaced by $c_3(|E \restriction B_i| + |B_i|)$ for some $c_3 \in \mathbb{N}$).

Proposition 6.2. *The optimized algorithm for the general case on input G correctly computes the bisimulation contraction of G. If $G = \langle N, E\rangle$ is a graph with no critical nodes, then its worst case complexity is* $O(|N| + |E|)$.

Proof. Correctness follows from the correctness of the algorithm for the general case (see Proposition 5.5) and the correctness of the procedure in [39].

The only (minor) problem lies in the fact that in our case at any given rank we are actually working with partial functions. In fact if G has no critical nodes, then the graph G_i of the nodes at rank i can be seen as the graph of a partial function f on B_i: $f(n) = m$ iff $\langle n, m\rangle \in E$. The original statement of the single-function coarsest partition problem assumes, instead, the input functions to be total. However, it is not difficult to see how to produce the graph of a total function equivalent for our purposes. For example, add a new node with a self-loop acting as a *sink* for those nodes where the input function is not defined.

The complexity result follows from the complexity analysis performed in Proposition 5.5 and from the linear time complexity of **Paige–Tarjan–Bonic** [39]. □

In Fig. 3 we show an example of a graph on which the above optimization can be performed and the overall algorithm turns out to be linear.

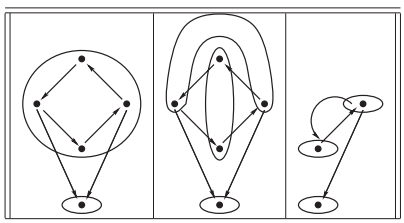


Fig. 3. Example of the first kind of topological optimization.

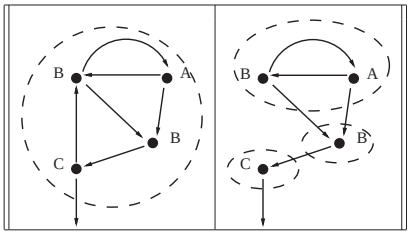


Fig. 4. Example of the second kind of topological optimization.

6.2. No bisimilar nodes on the same rank I

The crucial consideration behind the second optimization we propose is the following: the outgoing edges of a node u allow one to establish to which other nodes (of the same rank component) it is bisimilar. If we have some means to know that u is not bisimilar to any other nodes of its rank component, we can simply delete all edges outgoing from u. The deletion of a set of edges splits a rank component (i.e., we can re-compute the rank) and makes it possible to recursively apply our algorithm on a simpler case. The typical case in which the above idea can be applied occurs when, at a given iteration i, there exists a block X among the blocks of rank i which is a singleton set $\{n\}$: then all the outgoing edges from node n can be safely deleted. In next section we show the usefulness of this optimization in cases coming from formal verification.

In some cases the application at each step of this optimization allows the algorithm to run in linear time, as shown in Fig. 4. To improve the readability of the picture we have used the labels A, B, and C to distinguish the nodes which belong to the same block at the previous iteration, while the circles are used to denote the ranks (before and after edges' removal).

6.3. No bisimilar nodes on the same rank II

This optimization is a generalization of the previous one. Even in case there are no singletons in set D_i of blocks of rank i, some sort of optimization based on similar ideas is possible. For example, if X in D_i is of the form $\{u_1,\ldots,u_h\}$, let us define $succ(u_j)=\{Y\colon Y\in D_i \text{ and } \exists v\in Y\colon \langle u_j,v\rangle\in G_i\}$. It is immediate to see that if two nodes u_j and u_k of X are such that $succ(u_j)\neq succ(u_k)$, then they cannot be equivalent. In particular if there exists a node u_j such that $succ(u_j)$ is different from all the other sets $succ(u_k)$, then such node is not bisimilar to any node of G_i and we can delete its outgoing edges. The set-theoretic meaning of this deletion is that, for a given $u_j\in X$, all the candidates for bisimilarity have elements witnessing the difference. In Fig. 5 we present a case in which we apply this optimization.

The difference between this optimization and the previous one is that in the previous case the information from the nodes of rank less than i were enough to allow to delete some edges. In this case we also use information concerning the successors at the same rank of the node.

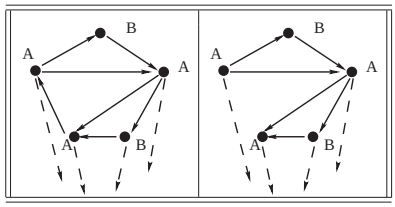


Fig. 5. Example of the third kind of topological optimization.

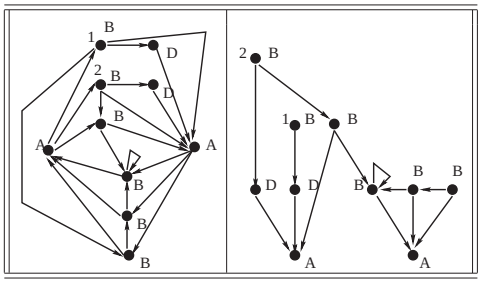


Fig. 6. Example of the fourth kind of topological optimization.

Notice that if we are considering a rank i on which no further splits are necessary, this optimization allows us to reach the solution in one step. Let $D_i = \{X_1, \ldots, X_k\}$; if for all $j \leqslant k$, for all $u, v \in X_j$ $succ(u) = succ(v)$, then for all $j \leqslant k$ all the nodes in X_j are bisimilar. In fact it is immediate to prove that the relation defined as $u \sim v$ iff $\exists j \leqslant k (u \in X_j \wedge v \in X_j)$ is the maximal bisimulation.

We conclude by observing that the set-theoretic point of view can easily suggest further optimizations of this sort. For example, if a set is a common element of all the elements of a given X in D_i, then it can be ignored (delete all the entering edges).

6.4. Limited negative strategy

The last topological optimization can be applied considering not only the successors of a node but all the paths of length less than or equal to a constant ℓ. An estimate of the value of ℓ guaranteeing good performances is necessary: if we do not put a bound on ℓ we obtain exactly the negative strategy. In Fig. 6 we show an example in which using this optimization our algorithm works in linear time. Labels A, B, C, and D indicate nodes in the same block. With $\ell = 2$ we are able to discover that the two nodes labeled A of the leftmost figure must be distinguished. In particular, the leftmost node reaches a node of the block D with a path of length 2, while the rightmost does not. Hence, we can remove all their (nodes labeled A) outgoing edges. In the rightmost figure we report the effect of this removal (we have reordered the

nodes in order to make more explicit the form of the graph obtained). Applying Paige and Tarjan' algorithm to this case it is possible to recognize that the node 1 is not bisimilar to node 2 only after four iterations, while the optimization allowed us to do it in one step.

Notice also that in this example the second topological optimization (Section 6.2), looking only for paths of length 1, does not allow to distinguish the two nodes.

6.5. Rank-based optimizations

The definition of *rank* given in Section 5 ensures that if two nodes are bisimilar, then they have the same rank (cf. Proposition 5.4). In this section we refine this definition, i.e. we give a definition of rank which still satisfies the property above, but such that there are *less* nodes with the same rank. This implies that the part of graph to be processed at each iteration is smaller and hence there is a higher probability that one of the topological optimizations could be applied. Any definition of rank satisfying Proposition 5.4 finer than the definition of rank given in Section 5, and computable in linear time, can be used to optimize our bisimulation algorithm.

The feature that we believe is crucial in the following definitions is that the rank is no longer integer number but a more structured notion (e.g. a set of integers). In the second case the rank cannot be computed a priori. Clearly, from a computational point of view there is a trade-off between the cost involved in determining the rank and the overall complexity of the algorithm.

In the inductive part of Definition 5.3, we have taken into consideration only the *maximum* of a set of ranks. We refine this definition considering the whole set of ranks.

Definition 6.3. Given a graph $G = \langle N, E\rangle$, then the rank$'$ of a node is recursively defined as follows:

$$
\begin{aligned}
rank'(n) &= \{0\} \quad \text{if } n \text{ is a leaf in } G\\
rank'(n) &= \{-\infty\} \quad \text{if } c(n) \text{ is a leaf in } G^{scc} \text{ and } n \text{ is not a leaf in } G\\
rank'(n) &= \{1 + m' : \langle c(n), c(m)\rangle \in E^{scc}, m \in WF(G), m' \in rank'(m)\}\\
&\quad \cup \{m' : \langle c(n), c(m)\rangle \in E^{scc}, m \notin WF(G), m' \in rank'(m)\} \quad \text{o.w.}
\end{aligned}
$$

A possible way to efficiently compute such rank is by maintaining a bit-vector of length ρ (the maximal well-founded rank in G) for each node. Such vectors can be computed using bit-wise disjunctions over the vectors associated to the successors.

Proposition 6.4. *Let m and n be nodes of a graph G:*
(i) $m \equiv \Omega \leftrightarrow rank'(m) = \{-\infty\}$.
(ii) $m \equiv n \rightarrow rank'(m) = rank'(n)$.

Proof. The argument for (i) and for the first part of the argument for (ii) are the same as in Proposition 5.4.

Let us focus on the case in which both m and n are not well-founded and not bisimilar to Ω. Let $h \in rank'(m)$ and let a be a well-founded node reachable from m such that $h-1 \in rank'(a)$. Since $m \equiv n$, there exists a node b reachable from n such that $a \equiv b$. Since a and b are well-founded $rank'(a) = rank'(b)$ and this implies $rank'(m) \subseteq rank'(n)$. Symmetrically we can conclude that $rank'(m) \supseteq rank'(n)$. □

It easy to see that the above definition of rank refines properly the previous one and that it is possible to give an order in which rank components can be correctly processed. To that purpose we also compute the auxiliary vector $rank'_d$ that assigns a unique integer to each different set of integers.

Algorithm 3 (Procedure $Comp\_Rank'(G)$).

(1) **for** $c(n) \in N^{scc}$ **compute** $rank(c(n)) \in G^{scc}$;
(2) **compute** $WF(G)$; — compute the well-founded part of G
(3) $\theta := 3 + \max\{rank(c(n)) : n \in WF(G)\}$;
(4) $\sigma := \max\{rank(c(n)) : n \in G\}$;
(5) **for** $n \in N^{scc}$ **do**
  for $1, \ldots, \theta$ *do* $rank'(n)[i] := 0$; — initialize the bit-vectors
(6) **if** $rank(c(n)) = 0$ **and** n **is a leaf in** G **then** $rank'(n)[2] := 1$;
  — compute $rank'$ of nodes bisimilar to $\emptyset$
(7) **if** $rank(c(n)) = 0$ **and** n **is not a leaf in** G **then** $rank'(n)[1] := 1$;
  — compute $rank'$ of nodes bisimilar to Ω
(8) **for** $i = 1, \ldots, \sigma$ **do** — compute $rank'$ of the remaining nodes
  if $rank(c(n)) = i$ **then**
    $A := \{c(m) : \langle c(n), c(m)\rangle \in E^{scc}, m \in WF(G)\}$;
    $B := \{c(m) : \langle c(n), c(m)\rangle \in E^{scc}, m \notin WF(G)\}$;
    for $j = 1, \ldots, \theta$ **do**
      $rank'(c(n))[j] := \bigvee_{a \in A} rank'(a[j-1]) \vee \bigvee_{b \in B} rank'(b[j])$;
(9) **for** $n \in N$ **do** $rank'_d(n) = \sum_{j=1}^{\theta} rank'(c(n))[j] * 10^j$ — order the $rank'$'s

Algorithm 4 (General case using $rank'$).

(1) **Comp_Rank'**(G); — compute ranks
(2) $\rho := \max\{rank'_d(n) : n \in N\}$;
(3) **for** $i = 10, 11, \ldots, \rho$ **do** $B_i := \{n \in N : rank'_d(n) = i\}$;
(4) $C := \{B_i : i = 10, 100, 101, \ldots, \rho \wedge B_i \neq \emptyset\}$;
  — C is the partition to be refined initialized by the B_i's
(5) $G :=$ **collapse**(G, B_{10}); — collapse all the nodes of $rank'$ $\{-\infty\}$
(6) **for** $n \in B_{10}$ **do** — refine blocks at higher ranks
  for $C' \in C$ **and** $C' \neq B_{10}$ **do**
    $C := (C \setminus \{C'\}) \cup \{\{m \in C' : \langle m, n\rangle \in E\}, \{m \in C' : \langle m, n\rangle \notin E\}\}$
(7) **for** $i = 100, \ldots, \rho$ **and** $B_i \neq \emptyset$ **do**
  (a) $D_i := \{X \in C : X \subseteq B_i\}$; — determine the blocks at $rank'_d = i$
    $G_i := \langle N \cap B_i, E \restriction B_i\rangle$; — isolate the subgraph to process
    $D_i :=$ **Paige–Tarjan**(G_i, D_i); — process G_i

(b) **for** $X \in D_i$ **do**
$G := \mathbf{collapse}(G, X)$ — collapse nodes at $rank'_d = i$
(c) **for** $n \in B_i$ **do** — refine blocks at higher ranks
for $C' \in C$ **and** $C' \subseteq B_{i+1} \cup \cdots \cup B_\rho$ **do**
$C := (C \setminus \{C'\}) \cup \{\{m \in C' : \langle m, n\rangle \in E\}, \{m \in C' : \langle m, n\rangle \notin E\}\}$

Proposition 6.5. *If $G \equiv \langle N, E\rangle$ is a graph, then the above algorithm on input G correctly computes the maximal bisimulation on G and the worst case complexity is* $O(|E| \log |N|)$.

Proof. The correctness result can be proved using the same considerations used to prove Proposition 5.5. In general, a correctness result can be proved for our algorithm each time a notion r of rank which satisfies the following three properties is employed:
(a) the nodes collapsed in step (5) are exactly all the nodes bisimilar to Ω;
(b) if $m \equiv n$, then $r(m) = r(n)$;
(c) if $\langle n, m\rangle \in E$, then $r(m) \leqslant r(n)$.

From Proposition 6.4 we have that $rank'_d$ satisfies (a) and (b). The definitions of $rank'$ and $rank'_d$ ensure that $rank'_d$ satisfies (c).

As far as complexity is concerned, the only step different from the algorithm for the general case is step (1). Thus, we prove that the complexity of the procedure **Comp_Rank**$'(G)$ is $O(|E| + |N|)$. From Proposition 5.5 we have that step (1) of **Comp_Rank**$'(G)$ can be performed in $O(|E^{scc}| + |N^{scc}|)$. To execute step (2) of the procedure in linear time observe that it is possible to decide whether $n \in WF(G)$ using a breadth-first visit of $G(n)$. A node n is in $WF(G)$ if and only if n is a leaf or $\forall m(\langle n, m\rangle \in E \rightarrow m \in WF(G))$.

Steps (3)–(7) are trivial. Step (8) can be implemented using a breath-first visit of G^{scc} and a bit-sum over the ranks' vectors. □

The last rank optimization we sketch consists in determining the rank component on-line starting from the following observation. Let X and Y be two strongly connected components whose rank (according to Definition 5.3) is the same and such that Y is reachable from X in G^{scc}. For a given node a let us call *label* the block of C to which a belongs. X and Y can contain bisimilar nodes only if the set of labels of nodes in X is contained in the set of labels of nodes in Y.

On the ground of the above observation we could proceed by computing a $rank''$ for nodes belonging to the same strongly connected components on-line during the main loop of the algorithm.

We conclude noticing that it would be interesting to study the problem of the existence of a notion of rank (refining the ones given above) with respect to which the computation of the maximal bisimulation can be performed at optimal cost.

Our intuition is that, for a given non-well-founded set a, such an optimal notion should somehow encode the entire topology of the well-founded sets in the transitive closure of a.

7. Labeled graphs

In several applications (e.g., Concurrency, Databases, Verification) edges and nodes of the graphs to be tested for bisimilarity are *labeled*. Labels on edges can denote *actions* associated to the effect of moving from one state to another (e.g. in Concurrency) or they can contribute in defining different *relations* (e.g. in Databases). Labels on nodes typically identify a *property* that holds in that node.

The definition of bisimulation needs to be refined in order to take into consideration labels on nodes and edges:

Definition 7.1. Let A and L be two finite sets of labels. Given a labeled graph $G = \langle N, E, \ell \rangle$, with $E \subseteq N \times A \times N$ and $\ell : N \to L$, a labeled bisimulation on G is a relation $b \subseteq N \times N$ such that: [1]

- if $u_0 \, b \, u_1$, then $\ell(u_0) = \ell(u_1)$;
- if $u_0 \, b \, u_1$, then for $i = 0, 1$: if $u_i \xrightarrow{a} v_i \in E$, then there is an edge $u_{1-i} \xrightarrow{a} v_{1-i} \in E$ and $v_0 \, b \, v_1$.

We first assume that only nodes are labeled (e.g., when a unique label is used for all edges) and we discuss how to modify the algorithm in this case. We then consider labeled edges and nodes and we show how to reduce also this problem to the unlabeled one.

7.1. Labeled nodes

Let us consider the minimization problem on a graph $G = \langle N, E, \ell \rangle$ whose nodes are labeled. The only change with respect to the algorithm for the pure (unlabeled) case is in the initialization phase: the partition suggested by the *rank* function must be refined so as to leave in the same block only nodes with the same label. Then the previously presented algorithms can be employed without further changes. Thus, the unique overhead is that of performing a partition refinement guided by the set of node labels. Assuming that the set of labels employed is known in advance, this task can be done in time $O(|N|)$.

The correctness of this procedure is justified in the remaining part of the section. We will use the notion of m-chain to remove node labels: for $m \in \mathbb{N}$, let us call an *m-chain* a graph of the form

$$v_1 \to v_2 \to \cdots \to v_m$$

Node v_1 denotes the set $\{\cdots\{\overbrace{\emptyset\}\cdots\}}^{m-1}$. We identify such a node with $\{\emptyset\}^{m-1}$. The reader can easily check that:

Lemma 7.2. *Let $m, n \in \mathbb{N}$ and $m \neq n$. Then $\{\emptyset\}^m$ is not bisimilar to $\{\emptyset\}^n$.*

[1] We use $u \xrightarrow{a} v \in E$ for $\langle u, a, v \rangle \in E$.

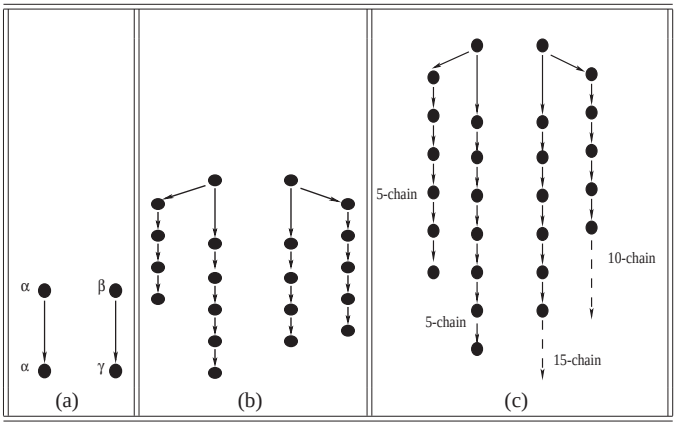


Fig. 7. A labeled graph, its wrong and correct unlabeling.

Given a graph with labeled nodes, the idea is to distinguish nodes with different labels by adding outgoing edges reaching different nodes of the kind $\{\emptyset\}^m$. The next example shows that chains must be chosen carefully.

Example 7.3. Consider the labeled graph G in Fig. 7(a). Using a 4-chain to replace the label α, a 5-chain for label β, and a 3-chain for label γ, we obtain the graph in Fig. 7(b). Observe that the two highest nodes (one coming from a node labeled α, the other from a node labeled β) are bisimilar.

The above problem can be easily solved as follows:

Definition 7.4. Let $G = \langle N, E, \ell \rangle$ be a graph with labeled nodes and assume, without loss of generality, that $\ell(N) = \{1, \ldots, k\}$. G' is the graph obtained from G by adding an outgoing edge from each node $u \in N$ according to the following rule:

$$(\ell(u) = i) \Rightarrow \text{add the edge } u \to \{\emptyset\}^{(n+1)*i}$$

where $n = |N|$, and adding all the required chains.

Example 7.5. Applying the graph construction technique of Definition 7.4 to the graph G in Fig. 7(a), we obtain the graph G' in Fig. 7(c). Observe that differently labeled nodes produce non-bisimilar nodes.

Proposition 7.6. *Let G be a graph with node labels and G' be the graph obtained from G using the technique of Definition* 7.4. *Let $\equiv'$ be the maximum bisimulation on G'. Then, the restriction of $\equiv'$ to the nodes of G is the maximum bisimulation on G.*

Proof. It is immediate to prove that a bisimulation b on G can be extended to a bisimulation b' on G'.

In order to prove that the restriction to G of a bisimulation b' on G' is a bisimulation on G we have to prove that

$$u\,b\,v \rightarrow \ell(u) = \ell(v).$$

Let us assume, by contradiction, that ubv, $\ell(u)=i$, and $\ell(v)=j$ with $i \neq j$. By construction, we know that there is a $(n+1)*i$-chain that is reached with an edge by u, with n the number of nodes of G. Since we assumed that ubv, there must be a $(n+1)*i$-chain reached by v. By assumption and construction, we know that there is a chain of length $(n+1)*j$ starting from v, with $i \neq j$. Moreover, all other chains starting from v have length $r+1+(n+1)*s$ with $r \leqslant n-1$. Hence, the length of these chains is not a critical of $(n+1)$, i.e. they cannot have length $(n+1)*i$. □

Thus, after preprocessing, we can run the algorithm for the pure case on this G'. The construction ensures that classes at lowest ranks contain the nodes of newly introduced chains. The effect of the introduction of such nodes is to split the classes of the original graph in the same way as if guided by labels. Of course, this reduction is only of theoretical interest: if implemented exactly as described it would requires the introduction of $O(|N|^2)$ new nodes. However, we could introduce only one copy of the longest chain and subsume all the others. In this way, we introduce at most $O(|N|)$ new nodes.

7.2. General case

Assume now that in $G=\langle N,E,\ell\rangle$ nodes and edges are labeled. Assume also, without loss of generality, that the labels of nodes and edges are picked from disjoint sets (i.e., $A \cap L = \emptyset$ — cf. Definition 7.1). We suggest the following encoding:

Definition 7.7. Obtain a graph G' from G applying the following rewriting rule until it is no longer possible: for each pair of nodes u,v and for each label a such that there is an edge $u \xrightarrow{a} v \in E$ (see also Fig. 8):

- remove the edge $u \xrightarrow{a} v$;
- add a new node μ, labeled by a;
- add the two (unlabeled) new edges $u \rightarrow \mu, \mu \rightarrow v$.

We obtain a new graph $G'=\langle N',E',\ell\rangle$, with labeled nodes that can be treated with the technique described in Section 7.1.

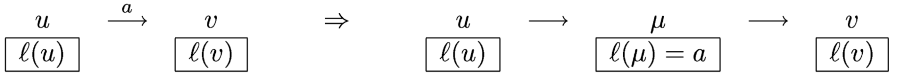


Fig. 8. Removing edges labels.

Proposition 7.8. *Let G be a labeled graph and G' be the graph obtained from G as described in Definition 7.7. Let $\equiv'$ be the maximum bisimulation on G'. The restriction $\equiv$ of $\equiv'$ to G is the maximum bisimulation on G.*

Proof. We prove that each bisimulation b over G can be extended to a bisimulation b' on G' and, vice versa, each bisimulation b' over G' can be restricted to a bisimulation b on G.

We denote by $\mu_{u,a,v}$ the new node introduced to replace the labeled edge $u \xrightarrow{a} v$.

Given a bisimulation b on G let us define its extension to G' as

$$\forall \mu_{u,a,v}, \mu_{u',a',v'} \in (N' \setminus N)(\mu_{u,a,v}\ b'\ \mu_{u',a',v'} \leftrightarrow (u\ b\ u' \wedge v\ b\ v' \wedge a = a')).$$

It is immediate to prove that b' is a bisimulation over G'.

Given a bisimulation b' over G', consider its restriction b over G':

$$\forall u, v \in N(u\ b\ v \leftrightarrow u\ b'\ v).$$

Again, it is immediate to prove that b is a bisimulation over G.

As a consequence, the restriction of the maximum bisimulation over G' to G its the maximum bisimulation over G. □

Let us now estimate the potential increase in time and space complexity introduced by the above rewriting method. We know that:

- $|N'| = |N| + |E|$,
- $|E'| = 2|E|$.

As far as time is concerned, we know that the algorithm will run in time $O(|N'| + |E'| \log |N'|) = O(|E| \log |N|)$.

As far space the method introduces a new node for each original edge. This seems to lead to a non-acceptable waste of space. Let us analyze in detail this problem. Since we are not using symbolic representations (e.g., OBDDs), the more compact way to represent a graph is by adjacency lists.

- We maintain an array of $|N|$ pairs (node label, list). Each list's cell stores:
  - an edge label,
  - the identifier of the reached node, and
  - the pointer to the 'next' cell of the list

  Let us assume that each of the three pieces of data uses one word w. Globally, we need space: $2|N|w + 3|E|w$.
- Let us consider the adjacency list representation of G'. We have now a vector of $|N| + |E|$ pairs (label, list). However, the first N lists are now simpler than those used for G: a cell stores only the reached node and the pointer to the 'next' cell. The remaining $|E|$ lists are lists made by a unique cell (newly introduced nodes have a unique outgoing edge reaching a node among the first $|N|$). Hence the list field of the vector cell will need to store only one reference. Globally, we need space $2|N|w + 4|E|w$.

  To sum up, memory requirement increases by a factor less than $\frac{4}{3}$.

8. Testing

We present here some tests performed on the implementation of our bisimulation algorithm. First we motivate, from a theoretical point of view, the set of tests we have chosen.

To the best of our knowledge there is no *official* set of benchmarks for testing an algorithm such as the one we propose. We decided to test our implementation in the context of formal verification using model checkers (such as SPIN) and considering the transition graphs they generate from a given program. In [30] it is described how to check that the implementation of a protocol conforms a formal specification. To this purpose the implementation is translated into interacting finite-state machines. Usually the graphs of these machines are composed by a unique strongly connected component. The theory presented in [30] finds application in the model checker SPIN [31]. We have taken into consideration the transition systems generated by the examples in the SPIN package. The alphabets which label the edges of the graphs of these transition systems have usually a large number of characters. When we translate these graphs into graphs without edge labels (see Section 7.2), we obtain graphs on which we can perform the second optimization proposed in Section 6. Such an optimization allows us to delete edges in the graphs, obtaining, in a significant number of cases, graphs on which our algorithm works in linear time. For example, in Fig. 9 we show the graph we obtain for the process Cp0 of the Snooping Cache protocol. We obtain similar results for all its procedures. From left to right, we show: the graph with edge labels, its translation into a graph without edge labels, the graph obtained applying our optimization, and finally, is bisimulation contraction. Observe that the graph in the third column can be computed from the initial one in linear time and our algorithm determines its bisimulation contraction in linear time.

In Fig. 10 we show the graphs of the process Node in the Leader protocol from the SPIN package. This protocol implement the Dolev, Klawe, and Rodeh' algorithm for leader election in unidirectional ring [18]. The upper graph is obtained applying the mapping described in Section 7 in order to eliminate the edge labels. The lower graph is obtained by applying the second optimization described in Section 6. The latter is an acyclic graph, hence our algorithm computes its bisimulation contraction in linear

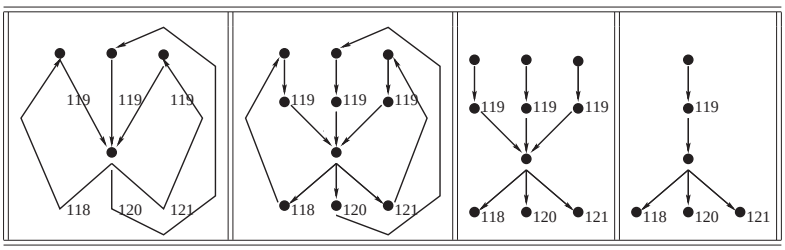


Fig. 9. Bisimulation contraction of Cp0 from Snoopy.

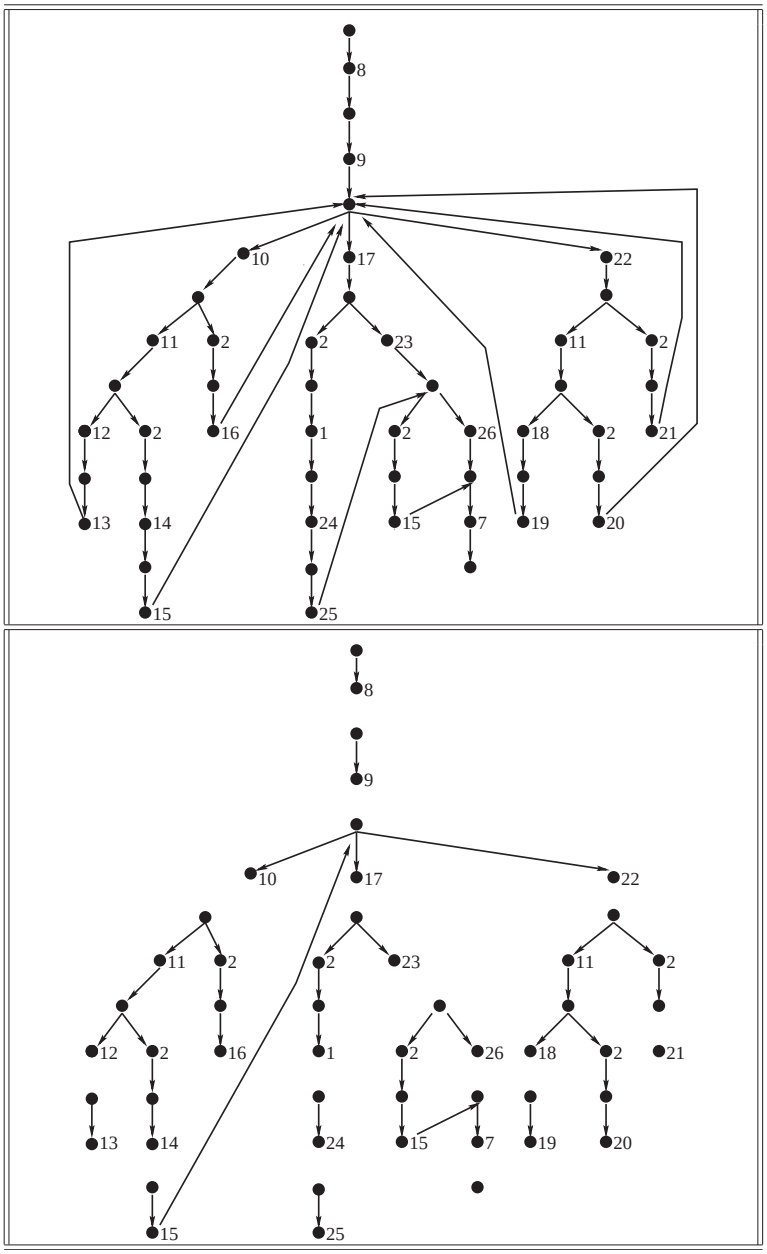


Fig. 10. Bisimulation contraction of Node from Leader.

time. These considerations about the topology of the graphs coming from the context of verification suggest us some further examples on which compare the performances of our algorithm with those of Paige and Tarjan' algorithm.

Nodes :	Alg	PT
80005	.5188	.9000
120005	.7893	1.3807
160005	1.0560	1.8773
200005	1.3250	2.3690
240005	1.5900	2.8620
280005	1.8540	3.3840

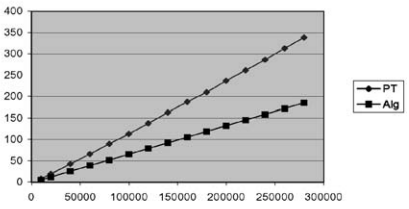
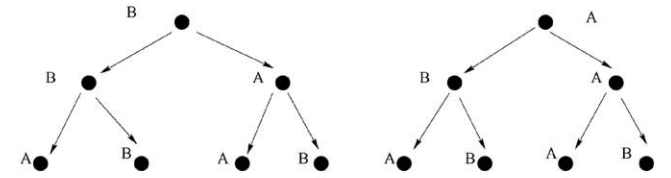


Fig. 11. Test (1) (Hopcroft's graphs).

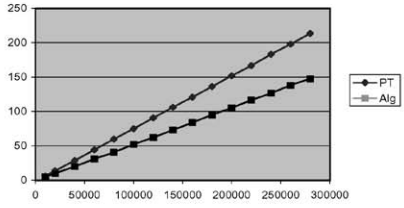
The algorithm has been implemented in standard C and compiled using the C++ Builder 5 Compiler. As far as the optimizations presented in Section 6 are concerned, we implemented only the optimization relative to the use of **Paige–Tarjan–Bonic**, when it is possible (see Section 6.1). Tests have been executed on a PIII, 600 MHz PC, OS Windows NT. We compare the execution times of the **Paige–Tarjan** procedure [40] with those of our algorithm. In order to make the comparison meaningful, the **Paige–Tarjan** procedure has been implemented in the same code and the two algorithms employ the same data structures for graphs and partitions, as well as the same auxiliary procedures (e.g., that performing blocks splitting).

In Figs. 11 and 12 we report the running times of the two algorithms over 6 families of graphs. In the tables, Alg denotes the running time of our algorithm and PT denotes the running time of the Paige and Tarjan' procedure. Times are expressed in seconds. We briefly describe below these families of graphs.

(1) This is the graph used by Hopcroft in [33] as an example of automata on which its algorithm runs in time proportional to $|E|\log|N|$ (Fig. 11).
(2) Inspired by the previous graphs, we have tested the algorithm over graphs of the form reported on the top of Fig. 12. A and B are the node labels that split the initial partition.
(3) This is the graph of a function, i.e. each node has one outgoing-edge, with an initial partition of the nodes of the graph into three classes, A, B and C. Each node in $A\cup B$ reaches a node in C and each node in B reaches a node in A or a node in C. On this graph Paige and Tarjan's algorithm runs in time proportional to $|E|\log|N|$. The errors in **Paige–Tarjan** for more than 240,000 nodes are due to the fact that in general **Paige–Tarjan** requires more memory, since it always works on the global graph.
(4) Generalization of Test (3). This graph at each rank represents a function similar to the one in Test (3). All the nodes at rank i are also connected to nodes at rank $i-1$, hence the global graph is not the graph of a function.
(5) Labeled function. This is the graph of a function with an initial partition of the nodes chosen in such a way that there are few equivalence classes in the initial partition, but all the nodes are not bisimilar. As for Test (3), **Paige–Tarjan** with more than 240,000 nodes does not run on the system we used.
(6) Rank graph. In this graph the subgraph at rank i is a complete graph. The nodes at rank i are connected to the nodes at rank $i-1$ in such a way that all the nodes at rank i are bisimilar.

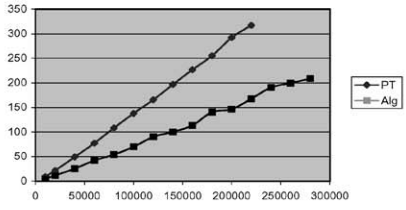


Nodes :	Alg	PT
80003	.4083	.6000
120003	.6240	.9070
160003	.8380	1.2100
200003	1.0530	1.5200
240003	1.2660	1.8300
280003	1.4760	2.1360



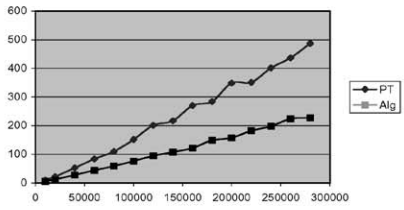
Test (2): graphs' picture and results

Nodes :	Alg	PT
80003	.5384	1.0852
120003	.9020	1.6575
160003	1.1353	2.2680
200003	1.4630	2.9230
240003	1.9060	ERR
280003	2.0880	ERR



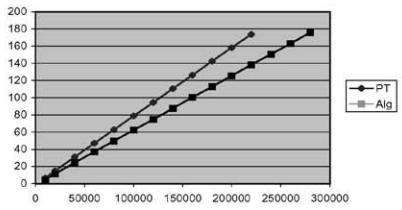
Test (3)

Nodes :	Alg	PT
80006	.5872	1.0972
120006	.9473	2.0100
160006	1.2190	2.7000
200006	1.5740	3.4880
240006	1.9760	4.0120
280006	2.2740	4.8660



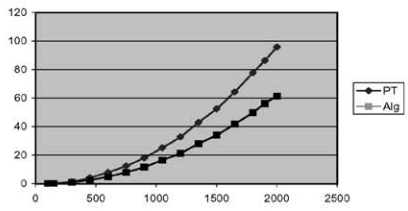
Test (4)

Nodes :	Alg	PT
80003	.4976	.6280
120003	.7485	.9435
160003	1.0047	1.2620
200003	1.2550	1.5830
240003	1.5060	ERR
280003	1.7600	ERR



Test (5)

Nodes :	Alg	PT
601	.048	.079
901	.116	.181
1201	.213	.328
1501	.341	.525
1801	.498	.777
2001	.614	.958



Test (6)

Fig. 12. Computational tests.

9. The symbolic rank-based algorithm

The proposed algorithm is implemented and tested on the explicit representation of the graph. In this section we analyze how the same ideas can be exploited in a symbolic setting, namely when the graph is symbolically represented using OBDDs [10]. We focus on the pure (unlabeled) case.

Before defining the rank-based symbolic bisimulation algorithm we review some basic notions on OBDDs and the computational complexity of symbolic procedures. Any Boolean function $f(x_1,\ldots,x_k)$ can be represented by a binary tree of height k, whose leaves are labeled by 0 or 1. A path from the root to a leaf represents a Boolean assignment $b_1 \ldots b_k$ for the variables $x_1,\ldots,x_k$. The labels of the leaves will be either 0 or 1 according to the Boolean value of $f(b_1,\ldots,b_k)$. Such a tree is called *Binary Decision Tree* (*BDT*) for the function f. This BDT can be processed bottom-up so as to obtain an acyclic graph that stores the same information in a more compact way: the OBDD for the function f (see [13]). OBDDs are canonical representations for Boolean functions since two Boolean functions are equivalent if and only if they are associated to the same OBDD [10].

The way OBDDs are usually employed in Model Checking to represent the states' space N, sets of states $S \subseteq N$, and the transition relation E, is based on the following observations [13]:

- we can assume that $N \subseteq \{0,1\}^u$, i.e. each node is encoded as a binary number (and $u = \lceil \log |N| \rceil$);
- a set $S \subseteq N$ is a set of binary strings of length u, specified by its characteristic (Boolean) function $\chi_S : \{0,1\}^u \to \{0,1\}$, where

$$\chi_S(s_1,\ldots,s_u) = 1 \Leftrightarrow \langle s_1,\ldots,s_u\rangle \in S;$$

- $E \subseteq N \times N$ is a set of binary strings of length $2u$ and it can be described by its characteristic function

$$\chi_E(x_1,\ldots,x_u,y_1,\ldots,y_u) = 1 \Leftrightarrow \langle x_1,\ldots,x_u\rangle E \langle y_1,\ldots,y_u\rangle.$$

Since χ_S and χ_E are Boolean functions, they can be represented using OBDDs. In particular, in the OBDD representing E the first u levels (variables) represent the codes of the source nodes, while the second u levels represent the codes of the target nodes. If B is an OBDD, then $|B|$ denotes the number of its nodes.

Various packages have been developed to manipulate OBDDs: Somenzi's CUDD from Colorado University [46], Lind-Nielsen's BuDDy, Biere's ABCD, Janssen's OBDD package from Eindhoven University of Technology, Carnegie Mellon's OBDD package, the CAL package from Berkeley [44], K. Milvang-Jensen's parallel package BDDNOW, Yang's PBF package. All these packages are endowed with a number of built-in operations which allow to manipulate and combine OBDDs. Here we are interested in some of these operations: the equality test, the Boolean operations $\cup, \cap, \setminus$, and in the `img` (image) and `preimg` (pre-image) operations.

Equality test can be considered a constant time operation. This is reasonable because if f and g are represented by two OBDDs in a *unique table*, then the functions are

equal if and only if f and g are two pointers to the same memory location in the table.

Let us assume that B_1 and B_2 are the OBBDs representing the Boolean functions $f_1(x_1,\ldots,x_k)$ and $f_2(x_1,\ldots,x_k)$, respectively. Then $B_1 \cup B_2$ is an OBDD that represents the function $f_1(x_1,\ldots,x_k) \vee f_2(x_1,\ldots,x_k)$ and can be computed by dynamic programming in time $O(|B_1||B_2|)$, (similarly for $\cap$ and $\setminus$).

The graph operations $\mathsf{img}(A,G)$ and $\mathsf{preimg}(A,G)$ allow to find the nodes that can be reached in one step forward (resp. backward) from a set of nodes A. They are implemented using relational products and have a worst-case complexity which is exponential with respect to $|A|$ and $|G|$. In practical cases the cost of the operations img and preimg even though acceptable is the crucial one. Thus, in the area of the symbolic algorithms [45], the operations img and preimg are referred as *symbolic steps* and the time complexities of symbolic algorithms are usually expressed as the number of symbolic steps that are performed.

9.1. The symbolic rank-based bisimulation algorithm

In order to define a symbolic version of the algorithm proposed in the previous sections we mainly need to efficiently compute the rank-partition of the graph. All other operations can be considered standard in symbolic terms:

- **collapse**(G,X) (steps (5) and (7b)) means that all the nodes in X are bisimilar. They are already in the same set symbolically represented: no further operation is needed.
- The operations in the **for**-loops at steps (6) and (7c) are standard splitting operations, i.e. they replace C with $C \cap \mathsf{preimg}(X)$ and $C \setminus \mathsf{preimg}(X)$.
- The extraction of the subgraph $G_i = E \restriction B_i$ at step (7a) corresponds to the Boolean operation

$$E(\bar{x},\bar{y}) \wedge (B_i(\bar{x}) \wedge B_i(\bar{y}))$$

  where $E(\bar{x},\bar{y})$ is the OBDD for the set of edges, while $B_i(\cdot)$ is the OBDD for the set of nodes of rank i. If $\bar{c}$ is the Boolean code of a node n, $B_i(\bar{c})$ is true if and only if n has rank i.
- The operation **Bisim**(G_i,D_i) at step (7a) can be performed by using a symbolic bisimulation algorithm and it will be briefly discussed in Section 9.2.

Therefore, in this section we focus only on the rank computation.

In the explicit case Tarjan's algorithm [47] identifies, in $O(|N|+|E|)$ steps, all the strongly connected components of G. Once the graph G^{scc} has been computed, it is possible to assign to each node of G its rank, accordingly to Definition 5.3, through a visit of G. Such a two-step procedure is applicable also symbolically, however, the algorithm in [47] cannot be used as a subroutine: the efficient computation of G^{scc} in [47] relies on the labelling of each node of the input graph. In other words [47] is an explicit algorithm that cannot be translated symbolically. The most efficient symbolic algorithms to determine G^{scc} are those described in [6], that requires $O(|N|\log|N|)$ symbolic steps, and the algorithm recently presented in [26] which is linear and would therefore solve the problem.

Here, as an alternative, we show how the explicit construction of the SCC can be avoided. We start revisiting the Definition 5.3 to give a different characterization of the notion of rank. Such a reformulation leads us to the definition of a procedure performing the rank-layering of a graph in $O(|N|)$ symbolic steps, avoiding the computation of G^{scc}.

Definition 9.1. Let $G = \langle N, E\rangle$. For each node $n \in N$ let $\overline{rank}(n)$ be defined as follows:

$$\begin{aligned}
\overline{rank}(n) &= 0 \quad \text{if } n \text{ is a leaf of } G\\
\overline{rank}(n) &= \max(\{1 + \overline{rank}(m) : \langle n, m\rangle \in E\}) \quad \text{if } n \in WF(G) \text{ is not a leaf}\\
\overline{rank}(n) &= \max(\{-\infty\} \cup \{1 + \overline{rank}(m) :\\
&\qquad m \in WF(G) \wedge path(n, m)\}) \quad \text{if } n \notin WF(G)
\end{aligned}$$

where $path(n, m)$ is true iff there is a path connecting n to m in G.

The following lemma states the equivalence between the above definition and Definition 5.3.

Lemma 9.2. *Let $G = \langle N, E\rangle$. For each node $n \in N$ it holds*:

$$rank(n) = \overline{rank}(n).$$

Proof. Consider $G^{scc} = \langle N^{scc}, E^{scc}\rangle$. We start by observing that if $m, n \in N$ belong to the same strongly connected component, then by Definition 5.3 it holds that $rank(m) = rank(n)$. Since two nodes in the same strongly connected component reach exactly the same nodes, it also holds, by Definition 9.1, that $\overline{rank}(m) = \overline{rank}(n)$. After the above considerations we can proceed in the proof by induction on the height of G^{scc}.

For the base case, let $n \in N$ be such that $c(n)$ is a leaf in G^{scc}. Then, either n is a leaf of G or there is no path from n to any node in $WF(G)$. Hence, by Definition 5.3, either $rank(n) = \overline{rank}(n) = 0$, or $rank(n) = \overline{rank}(n) = -\infty$.

For the inductive step, let $n \in N$ be such that $c(n)$ has height $h+1$ in G^{scc}. If $n \in WF(G)$ then $\langle n, m\rangle \in E$ iff $\langle c(n), c(m)\rangle \in E^{scc}$. Moreover, if $\langle c(n), c(m)\rangle$ is an edge of G^{scc}, then m is a well-founded node. Hence, exploiting the inductive hypothesis together with Definition 5.3 and Definition 9.1 it holds that:

$$\max(\{1 + \overline{rank}(m) : \langle n, m\rangle \in E\}) = \max(\{1 + rank(m) : \langle c(n), c(m)\rangle \in E^{scc}\})$$

and $rank(n) = \overline{rank}(n)$.

If $n \notin WF(G)$, consider the set $S = \{m \mid \langle c(n), c(m)\rangle \in E^{scc}\}$. Since a well-founded node is reachable from n iff it is reachable from some $m \in S$, it holds that $\overline{rank}(n)$ is:

$$\max(\{\overline{rank}(m) : m \in S \cap WF(G)\} \cup \{\overline{rank}(m) : m \in S \setminus WF(G)\}) \cup \{-\infty\}.$$

The inductive hypothesis and the definition of *rank* allow to easily get the thesis. □

Hence, the rank of a well-founded node is the maximum length of a path starting from it, while the rank of a non-well-founded node is 1 plus the maximum length of

a path starting from one of its well-founded descendants (or $-\infty$ if such a path does not exist). The symbolic rank-layering algorithm (Algorithm 5) proceed as follows: it identifies the well-founded nodes, starting from rank 0 up to rank p; then, it uses the well-founded nodes to compute the ranks of the non-well-founded ones. In particular, first it uses the well-founded nodes at rank p to determine the non-well-founded nodes at rank $p+1$, then it uses the well-founded rank $p-1$ to determine the non-well-founded rank p, and so on. The linear complexity of the procedure follows from the fact that each pre-image computation discovers at least one new node of the graph. Hence, the number of symbolic steps is linear in the number of nodes of the graph. Theorems 9.3 and 9.4 state the correctness and the complexity of the proposed algorithm.

Algorithm 5 (**Symbolic Rank**($G = \langle N, E\rangle$)).

(1) $i := 0$;
(2) $SET := N$; — SET is the set of not-ranked nodes
(3) $PRESET :=$ preimg(SET); — PRESET= preimage of not-ranked nodes
(4) **while** $SET \neq PRESET$ **do**
  (a) $B_i := SET \setminus PRESET$;— $B_i =$ well-founded nodes of rank i
  (b) $SET := PRESET$;— remove well-founded nodes of rank i from SET
  (c) $PRESET :=$preimg(SET);— update PRESET
  (d) $i := i + 1$;
  — SET now contains only not well-founded nodes
(5) **for** $j = i$ **down to** 1 **do**
  $FRONT := B_{j-1}$;— put in FRONT well-founded nodes of rank $j-1$
  while preimg($FRONT$) $\cap\, SET \neq \emptyset$ **do**
    (a) $FRONT :=$ preimg($FRONT$) $\cap\, SET$;— discover new nodes
    (b) $SET := SET \setminus FRONT$;— remove from SET new nodes
    (c) $B_j := B_j \cup FRONT$;— assign rank j to discovered nodes
(6) **if** $SET \neq \emptyset$ **then** $B_{-\infty} := SET$;— rank $-\infty$ to nodes still in SET
(7) **return** $\{B_{-\infty}, B_0, \ldots, B_\rho\}$;

Theorem 9.3. *Let $G = \langle N, E\rangle$ be a graph. The* **Symbolic Rank** *algorithm always terminates and the classes of the partition over N induced by the rank are $\{B_{-\infty}, B_0, \ldots, B_\rho\}$.*

Proof. Let us consider the set of nodes SET, that is initialized in step (2) to N. Then, whenever it is modified, some nodes are removed from it and no node is added. In particular, each iteration of the first while-loop assigns to SET its pre-image. Such a pre-image is always a subset of SET. Each iteration of the second while-loop removes from SET the subset $SET \cap FRONT$ which is not empty (condition of the loop). The above considerations ensure the termination of the two while-loop as well as of the **Symbolic Rank** algorithm. Moreover, as soon as a subset has been removed from SET it is inserted in one of the B_i (steps (4(a)) and (5(c))), while $B_{-\infty}$ (step (6)) collects whatever remain in SET. Thus $\{B_{-\infty}, B_0, \ldots, B_\rho\}$ is a partition of N. We will now prove that each $n \in WF(G)$ is put in the right rank-set ($B_{rank(n)}$), during the $rank(n) + 1$th iteration of the first while-loop. Let us proceed by induction on the

rank of $n \in WF(G)$. The first iteration of the loop in step (4) puts in B_0 all nodes in $N \setminus \mathsf{preimg}(N)$ and it is entered only if such a set is not empty. Thus, if there are non-well-founded nodes ($N \setminus \mathsf{preimg}(N)$), the first while-loop is not executed. Otherwise, all the leaves of the graph are put in B_0 during its first iteration. For the inductive step, note that steps (4(b)) and (4(c)) ensure that, as soon as a vertex is assigned to a rank, it is removed from *SET*. Hence, at the beginning of the $(j+1)$th iteration of the first loop with $j+1 \leqslant max\{rank(n)+1 | n \in WF(G)\}$, *SET* is N deprived of all well-founded nodes having height less then j. If *SET* is equal to its preimage (*PRESET*) we have that $SET = N \setminus WF(G)$ and the loop is not entered. Otherwise B_j contains all well-founded nodes having height j. Now, consider the for-loop (step (5)) and let $\gamma = max\{rank(n) | n \in WF(G)\}$. We have just proved that, upon entering such loop, *SET* contains all non-well-founded nodes of N. The first for-loop iteration is executed only if $i \geqslant 1$ (i.e. only if some well-founded rank has been generated) and inserts in $B_{\gamma+1}$ all nodes having some descendent in B_γ. Moreover, step (5(b)) removes from *SET* all nodes just assigned to a rank. Thus, an inductive argument can again be used to prove that the jth iteration, with $j \in \{1, \ldots, \gamma+1\}$, puts in $B_{\gamma+2-j}$, all non-well-founded nodes whose maximal-height well-founded descendent has *rank* $\gamma+1-j$. Hence, when step (6) is executed, *SET* contains all nodes having no well-founded descendent which are put in $B_{-\infty}$. We can conclude that $\{B_{-\infty}, B_1, \ldots, B_i\}$ are the classes of the partition of N induced by the *rank*. □

Theorem 9.4. *Let $G = \langle N, E \rangle$ be a graph. The* **Symbolic Rank** *algorithm performs* $O(|N|)$ *symbolic steps to produce the partition* $\{B_{-\infty}, B_0, \ldots, B_\rho\}$ *of N.*

Proof. Let γ be the maximum rank of a well-founded node and $M = N \setminus WF(G)$. We will prove that Algorithm 5 performs at most $O(\gamma + |M|)$ symbolic steps. Trivially $\gamma \leqslant |WF(G)|$, hence $O(\gamma + |M|) = O(|N|)$. The jth iteration of the first while-loop discovers exactly those well-founded nodes having rank $j-1$ performing only one symbolic step (step (4(c))). Hence, to execute steps (1)–(4) we perform at most γ symbolic steps. As stated by Theorem 9.3, before entering the for-loop *SET* is $N \setminus WF(G) = M$. During each iteration of the innermost while-loop at least one node is removed from *SET* (step (5(b))), since $FRONT \cap SET \neq \emptyset$ because of the while-guard. Moreover, *SET* is never augmented during the computation. Since during each iteration of the innermost while-loop only one pre-image operation is executed the global cost of steps (5)–(7) is $O(|M|)$ symbolic steps.

Note that also the number of set-differences, intersections and unions involved in the procedure is $O(|N|)$. □

9.2. Local bisimulation splitting

As we said in Section 2, in [22] Fisler and Vardi analyzed the symbolic cost of three symbolic bisimulation algorithms. In particular, they prove that for the symbolic version of the Paige and Tarjan algorithm the overall complexity depends on $\alpha(2M+D+I+Q)$, where α is the number of iterations necessary to reach the fix-point, M is the cost of

an image or preimage operation and D, I, and Q are the costs of one operation of difference, intersection, and equality test, respectively.

A symbolic version of the Paige and Tarjan' algorithm can be used in step (7(a)) of our symbolic algorithm. The differences between using directly the symbolic version of the Paige and Tarjan' algorithm and using it inside our routine, correspond to the differences that arise in the explicit case. First, we start with an initial partition, the rank-partition, which is finer than the one used in Paige and Tarjan' algorithm, hence, in general, our computation requires less iterations to converge to a fix-point. Moreover, during the ith iteration we work on the OBDDs representing the graph G_i, instead of working on the OBDD representing the graph G. This implies that we perform pre-image computations on smaller sets of nodes. Finally, we use the edges which connect nodes at different ranks only once, while this is not necessarily the case in Paige and Tarjan' algorithm.

The notion of rank provides a partition finer than the trivial partition $\{N\}$, which can be used in any algorithm which computes the maximum bisimulation relation $\equiv$ using a negative strategy. Bouali and de Simone' algorithm [Bds92] starts with the total relation $R_0 = \{\langle n, m\rangle : n, m \in R\}$, where R is the subset of N of reachable nodes and during the ith iteration it refines R_{i-1} in order to determine the relation R_i as follows:

$$R_{i-1} \setminus \{\langle n, m\rangle, \langle m, n\rangle : \exists\ n_1(\langle n, n_1\rangle \in E \wedge \forall m_1(\langle m, m_1\rangle \in E \rightarrow \langle n_1, m_1\rangle \notin R_{i-1}))\}.$$

It terminates when it reaches a fix-point which, in particular, coincides with the maximum bisimulation relation. The correctness of Bouali and de Simone' algorithm remains valid whenever the starting relation R_0 contains the maximum bisimulation relation $\equiv$, i.e. $\equiv\ \subseteq R_0$. The more relation R_0 approximates the relation $\equiv$, the less is the number of iterations necessary to compute $\equiv$. Hence, once the rank has been symbolically computed we can exploit it to speed up Bouali and de Simone' algorithm by starting with

$$R_0 = \{\langle n, m\rangle : \mathit{rank}(n) = \mathit{rank}(m)\}.$$

The Ordered Binary Decision Diagram for R_0 can be built from the OBDDs for $B_{-\infty}, B_0, \ldots, B_\rho$, since the characteristic function of R_0 is

$$(B_{-\infty}(\bar{x}) \wedge B_{-\infty}(\bar{y})) \vee \bigvee_{i=0}^{\rho} B_i(\bar{x}) \wedge B_i(\bar{y}).$$

10. Conclusion and further developments

We proposed algorithms to determine the minimum, bisimulation equivalent, representation of a directed graph or, equivalently, to test bisimilarity between two directed graphs. The algorithms are built making use of algorithmic solution to the relational and single-function coarsest partition problems as subroutines. In the acyclic case the performance of the presented algorithm is linear while, in the cyclic case turns out to be linear when a condition (absence of critical nodes) is satisfied. In general the

performance is no worse than that of the best-known solution for the relational coarsest partition problem.

Fisler and Vardi [22] compare three minimization algorithms (cf. Section 2). An important conclusion of that paper is that the Paige and Tarjan' algorithm runs faster than algorithms developed specifically for verification purposes. This suggests that "minimization algorithms tailored to verification settings should pay attention to choosing splitters carefully". The algorithm we have presented here is again not specifically tailored to verification, and its main difference with respect to Paige and Tarjan's is that it performs better choices of the splitters and of the initial partition thanks to the use of the notion of rank. In some cases we obtain linear time performances, moreover the initial partition we use allows to process the input without storing the entire structure in memory at the same time. This allows (potentially) to deal with larger graphs than those treatable using a Paige and Tarjan-like approach.

Many other lines of research could be pursued using the same circle of ideas. For example, it would be stimulating to extend and further study the connection and analogy with algorithms and problems considered by the scientific community studying formalism to denote concurrent processes. Moreover, it would be interesting to see if the algorithm studied for the case of hyperset could be adapted to study the case of hyper-multisets. In general, it would be interesting to study the adaptation of our work to the case in which the underlying theory of non-well-founded sets assumes axioms different from AFA [1].

Recently, connections between databases designed to easily access the web and hypersets has been pointed out in [36]. For such applications, bisimulation-matching is the engine of the operational semantics of graphical query languages for web-like databases [17].

Further studies relative to the applicability of the ideas presented here to the problem of determining *simulations* have also been presented (see [11,25,28]).

Acknowledgements

We thank Nadia Ugel for her C implementation and many useful discussions, and Raffaella Gentilini and Elio Panegai for their contribution in the development and implementation of the Symbolic Algorithm of Section 9.1.

References

[1] P. Aczel, Non-Well-Founded Sets, CSLI Lecture Notes, Vol. 14, Stanford University Press, Stanford, 1988.

[2] A. Aziz, V. Sighal, G. Swamy, R. Brayton, Minimizing interacting finite state machines: a compositional approach to language containment, in: Proc. Internat. Conf. on Computer Design (ICCD'94), IEEE Computer Society Press, Silver Spring, MD, 1994, pp. 255–261.

[3] J. Barwise, L. Moss, Hypersets, The Math. Intelligencer 13 (4) (1991) 31–41.

[4] J. Barwise, L. Moss, vicious circles. On the Mathematics of non-well-founded phenomena, CSLI Lecture Notes, Vol. 60, Stanford University Press, Stanford, Stanford, 1996.

[5] J. van Benthem, Modal correspondence theory, Ph.D. Thesis, Universiteit van Amsterdam, Instituut voor Logica en Grondslagenonderzo ek van Exacte Wetenschappen, 1976.
[6] R. Bloem, H.N. Gabow, F. Somenzi, An algorithm for strongly connected component analysis in $n \log n$ symbolic steps, in: W.A. Hunt Jr., S.D. Johnson, (Eds.), Proc. Internat. Conf. on Formal Methods in Computer-Aided Design (FMCAD'00), Lecture Notes in Computer Science, Vol. 1954, Springer, Berlin, 2000, pp. 37–54.
[7] A. Bouajjani, J.C. Fernandez, N. Halbwachs, Minimal model generation, in: E. Clarke, R. Kurshan (Eds.), Proc. Internat. Conf. on Computer Aided Verification (CAV'90), Lecture Notes in Computer Science, Vol. 531, Springer, Berlin, 1990, pp. 197–203.
[8] A. Bouali, XEVE, an ESTEREL verification environment, in: A.J. Hu, M.Y. Vardi (Eds.), Proc. Internat. Conf. on Computer Aided Verification (CAV'98), Lecture Notes in Computer Science, Vol. 1427, Springer, Berlin, 1998, pp. 500–504.
[9] A. Bouali, R. de Simone, Symbolic bisimulation minimization, in: G. von Bochmann, D.K. Probst (Eds.), Proc. Internat. Conf. on Computer Aided Verification (CAV'92), Lecture Notes in Computer Science, Vol. 663, Springer, Berlin, 1992, pp. 96–108.
[10] R.E. Bryant, Graph based algorithms for boolean function manipulation, IEEE Trans. Comput. C-35 (8) (1986) 677–691.
[11] D. Bustan, O. Grumberg, Simulation based minimization, in: D.A. McAllester (Ed.), Proc. 17th Internat. Conf. on Automated Deduction (CADE'00), Lecture Notes in Computer Science, Vol. 1831, Springer, Berlin, 2000, pp. 255–270.
[12] D. Cantone, E.G. Omodeo, A. Policriti, Set theory for computing. From decision procedures to declarative programming with sets, Monographs in Computer Science, Springer, Berlin, 2001.
[13] E.M. Clarke, O. Grumberg, D.A. Peled, Model Checking, The MIT Press, Cambridge, MA, 1999.
[14] R. Cleaveland, J. Parrow, B. Steffen, The concurrency workbench: a semantics based tool for the verification of concurrent systems, ACM Trans. Programming Languages Systems 15 (1) (1993) 36–72.
[15] R. Cleaveland, S. Sims, The NCSU concurrency workbench, in: R. Alur, T.A. Henzinger (Eds.), Proc. Internat. Conf. on Computer Aided Verification (CAV'96), Lecture Notes in Computer Science, LNCS, Vol. 1102, Springer, Berlin, pp. 394–397.
[16] R. Cleaveland, O. Sokolsky, Equivalence and preorder checking for finite-state systems, in: J.A. Bergstra, A. Ponse, S.A. Smolka (Eds.), Handbook of Process Algebra, North-Holland, Amsterdam, 2001, pp. 391–424.
[17] A. Cortesi, A. Dovier, E. Quintarelli, L. Tanca, Operational and abstract semantics of the query language G-Log, Theoret. Comput. Sci. 275 (1–2) (2002) 521–560.
[18] D. Dolev, M.M. Klawe, M. Rodeh, An $O(n \log n)$ unidirectional distributed algorithm for extrema finding in a circle, J. Algorithms 3 (3) (1982) 245–260.
[19] A. Dovier, R. Gentilini, C. Piazza, A. Policriti, Rank-based symbolic bisimulation (and model checking), in: R.J. Guerra, B. de Queiroz (Eds.), WoLLIC'2002, 9th Workshop on Logic, Language, Information and Computation, Electronic Notes in Theoretical Computer Science, Vol. 67, Elsevier, Amsterdam, 2002.
[20] A. Dovier, C. Piazza, A. Policriti, A fast bisimulation algorithm, in: G. Berry, H. Comon, A. Finkel (Eds.), Proc. Internat. Conf. on Computer Aided Verification (CAV'01), Lecture Notes in Computer Science, Vol. 2102, Springer, Berlin, 2001, pp. 79–90.
[21] J.C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, M. Sighireanu, CADP: a protocol validation and verification toolbox, in: R. Alur, T.A. Henzinger (Eds.), Proc. Internat. Conf. on Computer Aided Verification (CAV'96), Lecture Notes in Computer Science, Vol. 1102, Springer, Berlin, 1996, pp. 437–440.
[22] K. Fisler, M.Y. Vardi, Bisimulation and model checking, in: L. Pierre, T. Kropf (Eds.), Proc. Correct Hardware Design and Verification Methods (CHARME'99), Lecture Notes in Computer Science, Vol. 1703, Springer, Berlin, 1999, pp. 338–341.
[23] R. Focardi, R. Gorrieri, The compositional security checker: a tool for the verification of information flow security properties, IEEE Trans. Software Eng. 23 (9) (1997) 550–571.
[24] M. Forti, F. Honsell, Set theory with free construction principles, Annali Scuola Normale Superiore Pisa, Cl. Sc. IV (10) (1983) 493–522.

[25] R. Gentilini, C. Piazza, A. Policriti, Simulation as coarsest partition problem, in: J.-P. Katoen, P. Stevens (Eds.), Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS-02), Lecture Notes in Computer Science, Vol. 2280, Springer, Berlin, 2002, pp. 415–430.
[26] R. Gentilini, C. Piazza, A. Policriti, Computing strongly connected components in a linear number of symbolic steps, in: Proc. of Internat. Symp. on Discrete Algorithms (SODA'03), ACM, New York, 2003, pp. 573–582.
[27] M. Hennessy, H. Lin, Symbolic bisimulations, Theoret. Comput. Sci. 138 (2) (1995) 353–389.
[28] M.R. Henzinger, T.A. Henzinger, P.W. Kopke, Computing simulations on finite and infinite graphs, in: Proc. 36th IEEE Symp. on Foundations of Computer Science, (FOCS'1995), IEEE Computer Society Press, Silver Spring, MD, 1995, pp. 453–462.
[29] Y. Hirshfeld, M. Jerrum, F. Moller, A polynomial time algorithm for deciding bisimulation equivalence of normed basic parallel processes, Math. Struct. Comput. Sci. 6 (1996) 251–259.
[30] G.J. Holzmann, Design and Validation of Computer Protocols, Prentice-Hall, Englewood Cliffs, NJ, 1991.
[31] G.J. Holzmann, The model checker SPIN, IEEE Trans. Software Eng. 23 (5) (1997) 279–295.
[32] G.J. Holzmann, A. Puri, A minimized automaton representation of reachable states, Software Tools Technol. Transfer 2 (3) (1999) 270–278.
[33] J.E. Hopcroft, An $n \log n$ algorithm for minimizing states in a finite automaton, in: Kohavi, Paz (Eds.), Theory of Machines and Computations, Academic Press, New York, 1971, pp. 189–196.
[34] P.C. Kannellakis, S.A. Smolka, CCS expressions, finite state processes, and three problems of equivalence, Inform. Comput. 86 (1) (1990) 43–68.
[35] D. Lee, M. Yannakakis, Online minimization of transition systems, in: Proc. 24th ACM Symposium on Theory of Computing (STOC'92), ACM Press, New York, 1992, pp. 264–274.
[36] A. Lisitsa, P. Sazonov, V. Yu, Bounded hyperset theory and web-like data bases, in: Gottlob, Leitsch, Mundici (Eds.), Proc. Computational Logic and Proof Theory, 5th Kurt Gödel Colloquium, Lecture Notes in Computer Science, Vol. 1289, Springer, Berlin, 1997, pp. 172–185.
[37] K.L. McMillan, Symbolic Model Checking: An Approach to the State Explosion Problem, Kluwer Academic Publishers, Dordrecht, 1993.
[38] R. Milner, Operational and algebraic semantics of concurrent processes, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Elsevier Science, Amsterdam, 1990 (Chapter 19).
[39] R. Paige, R.E. Tarjan, R. Bonic, A linear time solution to the single function coarsest partition problem, Theoret. Comput. Sci. 40 (1985) 67–84.
[40] R. Paige, R.E. Tarjan, Three partition refinement algorithms, SIAM J. Comput. 16 (6) (1987) 973–989.
[41] D. Park, Concurrency and automata on infinite sequences, in: P. Deussen (Ed.), Proc. 5th Internat. Conf. on Theoretical Computer Science, Lecture Notes in Computer Science, Vol. 104, Springer, Berlin, 1981, pp. 167–183.
[42] F. Rahim, Property-dependent modular model checking application to VHDL with computational results, in: Proc. Internat. Workshop HLDVT, La Jolla, California, 1998.
[43] A.W. Roscoe, Model checking CSP, in: A.W. Roscoe (Ed.), A Classical Mind: Essays in Honour of C.A.R. Hoare, Prentice-Hall, Englewood Cliffs, NJ, 1994.
[44] J.V. Sanghavi, R.K. Ranjan, R.K. Brayton, A. Sangiovanni-Vincentelli, High performance bdd package based on exploiting memory hierarchy, in: Proc. ACM/IEEE Design Automation Conference, Las Vegas, Nevada, 1996.
[45] F. Somenzi, Binary decision diagrams, in: Calculational System Design, NATO Science Series F: Computer and Systems Sciences, Vol. 173, IOS Press, Amsterdam, 1999, pp. 303–366.
[46] F. Somenzi, CUDD: CU decision diagram package release 2.3.1, Department of Electrical and Computer Engineering University of Colorado at Boulder, 2001. Available at http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html.
[47] R.E. Tarjan, Depth first search and linear graph algorithms, SIAM J. Comput. 1 (2) (1972) 146–160.