Rank-Based Symbolic Bisimulation (and Model Checking)

Electronic Notes in Theoretical Computer Science 67 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume67.html 18 pages

Rank-Based Symbolic Bisimulation (and Model Checking)

A. Dovier [a,2] R. Gentilini [a,3] C. Piazza [b,4] A. Policriti [a,5]

[a] *Dip. di Matematica e Informatica, Univ. di Udine. Via delle Scienze 206, 33100 Udine (Italy).*

[b] *Dip. di Informatica, Univ. Ca' Foscari di Venezia. Via Torino 155, 30173 Mestre – Venezia (Italy).*

Abstract

In this paper we propose an efficient *symbolic* algorithm for the problem of determining the maximum *bisimulation* on a finite structure. The starting point is an algorithm, on explicit representation of graphs, which saves both time and space exploiting the notion of *rank*. This notion provides a layering of the input model and allows to proceed bottom-up in the bisimulation computation. In this paper we give a procedure that allows to compute the rank of a graph working on its symbolic representation and requiring a linear number of symbolic steps. Then we embed it in a fully symbolic, rank-driven, bisimulation algorithm. Moreover, we show how the notion of rank can be employed to optimize the CTL Model Checking procedures.

Key words: bisimulation, ordered binary decision diagrams, symbolic algorithms, model checking.

1 Introduction

Space requirement are a crucial parameter to be kept into account in many Computer Science areas (e.g., System Verification, Concurrency Theory, Data Bases, ...). The so-called *Symbolic* approach is an attractive and promising approach to the problem of saving space. The adjective *Symbolic* identifies all those data representations in which there is not an explicit exposition of the whole information. Sharing of the same information is used as much

[1] The work is partially supported by MIUR project: *Automatic Aggregate —and number— Reasoning for Computing.*

[2] Email: dovier@dimi.uniud.it
[3] Email: gentilin@dimi.uniud.it
[4] Email: piazza@dimi.uniud.it
[5] Email: policrit@dimi.uniud.it

as possible, producing an implicit view of data. Algorithms manipulating symbolic data are referred to as *Symbolic Algorithms*. Both the development and the complexity analysis of symbolic algorithms are substantially different from the explicit case ones. Algorithms born for explicit representation are often scarcely adaptable to the symbolic world. In graph algorithms, this happens typically when the algorithms need to dynamically store one data structure for each node. This kind of algorithms have to be completely redefined in a set-based fashion (c.f., for instance, [21] and [3]).

Ordered Binary Decision Diagrams (OBDDs) are a typical symbolic data-structure. They are widely recognized as a fundamental mean to deal with the *state-explosion* problem in *Model Checking* [8]. Bryant introduced OBDDs in the context of digital circuit analysis and specification [7], since they allow to compactly represent boolean functions. In [15] Mc Millan suggests how to exploit their power of reducing memory requirements in the verification domain, giving life to what is nowadays known as *Symbolic Model Checking*. In *Symbolic Model Checking* the models of the systems are compactly OBDD-represented and the algorithms which check the validity of a formula take advantage of the compact space representation. Rather than *explicitly* manipulate single states, they always deal with OBDDs representing subsets of states.

Bisimulation is another central notion in Model Checking which allows to reduce the systems state-spaces. The bisimulation quotient of a model (graph) can be used to replaced it, since it strongly preserves the whole μ-calculus. Many algorithms for bisimulation computation have been proposed in the literature both in the explicit [16,4,13] and in the symbolic [5] case.

In [9] the notion of rank has been introduced to develop a fast bisimulation algorithm working on the explicit representation of graphs. This notion, based on the concept of strongly connected component, allows to partition the nodes of a graph and to drive a bottom-up computation of the bisimulation. The algorithm runs in time $O(|E|)$ on various classes of graphs, where $|E|$ is the size of the set of edges. Its worst case complexity is $O(|E| \log |N|)$, where $|N|$ is the size of the set of nodes. In 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. In the symbolic case, the computation of the strongly connected components would require $O(|N| \log |N|)$ operations [3]. On this ground, we develop a symbolic rank-based bisimulation algorithm. Moreover, we point out the usefulness of the notion of rank to improve both explicit and symbolic CTL Model Checking procedures.

The paper is organized as follows: Section 2 presents the bisimulation problem, while Section 3 reviews the main approaches for solving it both in

the explicit and in the symbolic case. In Section 4 we introduce the notion of rank and the explicit rank-based bisimulation algorithm presented in [9]. Section 5 concerns with OBDDs representations of sets and relations. Section 6 is devoted to the description of the symbolic algorithm for computing the rank of a graph, while Section 7 discusses how to include it in a fully symbolic rank-based bisimulation algorithm. In Section 8 the employment of the notion of rank for explicit and implicit CTL Model Checking is described. Some conclusions are drawn in Section 9.

2 The Bisimulation Problem

In this section we introduce some basic notions used in the rest of the paper.

Definition 2.1 Given a graph $G = \langle N, E\rangle$, a *bisimulation* on G is a relation $b \subseteq N \times N$ such that:

(i) $n_1\, b\, n_2 \wedge \langle n_1, m_1\rangle \in E \Rightarrow \exists m_2(m_1\, b\, m_2 \wedge \langle n_2, m_2\rangle \in E)$

(ii) $n_1\, b\, n_2 \wedge \langle n_2, m_2\rangle \in E \Rightarrow \exists m_1(m_1\, b\, m_2 \wedge \langle n_1, m_1\rangle \in E)$.

It is well-known (see [1]) that given a graph G the *maximum bisimulation* $\equiv$ on G always exists, it is unique, and it is an equivalence relation over the set of nodes of G. This notion allows us to find a *minimum* representation of a graph G: $G/\equiv$, where $G/\equiv$ is the graph obtained from G by collapsing all equivalent nodes into a single one. This graph is usually called *bisimulation contraction of* G.

It is important to point out that the problem of determining the maximum bisimulation on a graph and the problem of recognizing whether two graphs are bisimilar are equivalent.

The problem we consider is that of finding the *bisimulation contraction* of a graph. The algorithms described in the literature to solve the bisimulation contraction problem are mainly based on a characterization of it in terms of *coarsest stable partition problem*. As a matter of fact, the notion of bisimulation can be connected to the notion of *stability*:

Definition 2.2 Let E be a relation on the set N, E^{-1} be its inverse relation, and P be 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 $B_1 \cap E^{-1}(B_2) = \emptyset$.

A partition P refines (is finer than) a partition Q if all the blocks in P are subsets of blocks in Q. Equivalently, Q is said to be coarser than P.

Definition 2.3 Let E be a relation on the set N and Q be a partition of N. The *coarsest stable partition problem* is the problem of finding the coarsest partition P refining Q that is stable w.r.t. E. When Q is not given, we assume it to be the trivial partition $Q = \{N\}$.

The above defined problem, which emerged in automata minimization, is equivalent to the problem of finding the bisimulation contraction of a graph.

Theorem 2.4 *Let $G = \langle N, E\rangle$ be a graph and P be the coarsest partition of N stable w.r.t. E. The equivalence relation induced by P on N is the maximum bisimulation $\equiv$ on G.*

Being able to reduce a graph by bisimulation is a critical step in Model Checking. The bisimulation-quotient strongly preserves *LTL*, *CTL*, CTL^* and the whole *μ-calculus* [8]. However it is important that the bisimulation computation is not an heavy overload for the model checking procedure.

3 Related Works

In the area of *explicit Model Checking* the models of the systems to be analyzed are represented by using adjacency-lists or other data-structures in which all the states are kept separately. On the other hand, the philosophy behind *Symbolic Model Checking* is that of using data-structures which maximize the sharing of information. In particular, in Symbolic Model Checking *OBDDs* (Ordered Binary Decision Diagrams) are used to give a compact representation of both the labelled graph modelling the system and of all partial computation results. Evidences of the usefulness of symbolic techniques to deal with the notorious *state-explosion* problem arising in Model Checking can be found in [8]. Since bisimulation is used in this context to reduce the state-spaces it is important/necessary to consider the problem from both an explicit and a symbolic point of view.

As far as the explicit case is concerned, the first significant algorithmic result for the solution of the bisimulation problem is due to Hopcroft. In [11], he presented an $O(|N|\log(|N|))$ algorithm for the minimization of the number of states of a 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 [17], where it is shown how to solve it in linear time in case of a single function. Finally, in [16] Paige and Tarjan solved the problem for the general case (i.e., bisimulation) 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. [17]) 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, both Hopcroft's solution to the (more difficult) many functions coarsest partition problem and Paige and Tarjan bisimulation algorithm, are based on a (more natural) *negative* strategy. The starting partition is the input partition and during each step the classes which do not satisfy the stability condition are split. The appealing complexity results of the procedures in [11] and [16] come from the use of a clever ordering (the so-called *process the smallest half* policy) for processing classes.

In [9] a bisimulation algorithm for explicit representation which combines positive and negative strategies has been presented. Such a combination relies on the possibility of layering the input model and proceed in the stabilization process bottom-up, on the defined layers. The notion of *rank* drives the above model stratification [6]. The just mentioned rank-based bisimulation procedure [9] has a time complexity which is in some cases linear, while in the worst case is the same as the one of Paige and Tarjan. Moreover, since during each iteration it refers only to the nodes at a fixed rank, it is clearly space efficient. In the next section we recall how the explicit routine in [9] works.

Other explicit procedures for the bisimulation problem, tailored for on-the-fly model checkers, have been presented by Bouajjani, Fernandez, and Halbwachs in [4] and by Lee and Yannakakis in [14].

In the *symbolic* case a popular bisimulation algorithm is the one proposed in [5] by Bouali and de Simone. It 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. Experimental results about the performances of the algorithm are presented, while there is not a deep discussion of its complexity in terms of basic OBDD operations.

In [10] Fisler and Vardi analyze the complexity of the symbolic versions of the algorithms of Paige and Tarjan [16], Bouajjani, Fernandez, and Halbwachs [4], and Lee and Yannakakis [14]. In particular, they determine the number of basic symbolic operations involved in each iteration of the three algorithms and they conclude, through experimental results, that an optimized version of the algorithm in [16], which splits only reachable blocks, performs better than the other two algorithms, since it gains from the *right* choice of the splitters.

4 The Explicit Rank-Based Bisimulation Algorithm

The main idea behind the explicit bisimulation algorithm presented in [9] is the use of the notion of *rank* to both initialize the partition and drive the order of the splitting operations. In order to present the notion of *rank* we first recall the notion of strongly connected components.

Definition 4.1 Given a graph $G = \langle N, E\rangle$, let $G^{scc} = \langle N^{scc}, E^{scc}\rangle$ be the graph obtained as follows:

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

[6] We refer to rank-based algorithms whenever some notion of rank is exploited in this sense.

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

We need to distinguish the acyclic (well-founded) part of a graph G from its cyclic (non-well-founded) part.

Definition 4.2 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. $WF(G)$, the well-founded part of G, is $WF(G) = \{n \in N : G(n) \text{ is acyclic}\}$.

The nodes in $WF(G)$ are said to be well-founded, while the other nodes are said to be non-well-founded.

Definition 4.3 Let $G = \langle N, E\rangle$. The *rank* of a node n of G is defined as:

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

In Figure 1 we report the rank-based algorithm presented in [9]. The complexity of this algorithm easily follows from the following considerations. Since G^{scc} can be computed using Tarjan's classical algorithm ([21]) in time $O(|N| + |E|)$, step (1) can be performed in time $O(|N| + |E|)$. Given a graph $G = \langle N, E\rangle$, let $\rho = \max\{rank(n) : n \in N\}$ and, for $i \in \{-\infty, 0, \ldots, \rho\}$, let $B_i = \{n \in N : rank(n) = i\}$. First, the algorithm initializes the partition P to be refined using the blocks B_i. For each rank i a bisimulation procedure **Bisim** is called on $G_i = \langle B_i, E \restriction B_i\rangle$ and the partition P is updated on nodes of rank greater than i. The procedure presented in [16] can be used with a cost $O(|E \restriction B_i| \log |B_i|)$, while [17] would cost $O(|E \restriction B_i| + |B_i|)$. In this way all the edges connecting nodes at different ranks are used only once. Hence, the algorithm correctly computes the bisimulation quotient of the input graph G and can be implemented so as to run with a worst case complexity of $O(|E| \log |N|)$. Moreover, if G is an acyclic graph, step (7) is superfluous and the overall cost is $O(|N| + |E|)$. In [9] further optimizations of the above algorithm are presented allowing a linear time complexity also in other cases.

We conclude this section by observing that the space requirement of the above algorithm can be reduced to the size of the largest G_i to be processed.

5 OBDDs and Symbolic Primitives

Before defining a rank-based symbolic bisimulation algorithm we review some basic notions on OBDDs and computational complexity of symbolic procedures. Binary Decision Diagrams (BDDs) are a fundamental data structure

Rank-Bisimulation$(G = \langle N, E\rangle)$

(1) **for** $n \in N$ **do compute** $rank(n)$; — *compute the ranks*

(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) $P := \{B_i : i = -\infty, 0, \ldots, \rho\}$; – *partition initialized with the B_i's*

(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) $D_i := \{X \in P : X \subseteq B_i\}$; — *find blocks currently at rank i*

    $G_i := \langle B_i, E \restriction B_i\rangle$; — *isolate the subgraph of rank i*

    $D_i :=$ **Bisim**(G_i, D_i); — *process rank i calling either [16] or [17]*

  (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** $C \subseteq B_{i+1} \cup \ldots \cup B_\rho$ **do**

        $P := (P \setminus \{C\}) \cup$

          $\{\{m \in C : \langle m, n\rangle \in E\}, \{m \in C : \langle m, n\rangle \notin E\}\}$;

Fig. 1. The algorithm in [9].

developed for efficiently storing boolean functions. General BDD's were first introduced in [12,2]. Bryant, introducing in [6] an ordering on the nodes of BDDs (OBDDs), attracted attention on the possibility of their use in *logic design verification*. OBDDs can be used to represent *symbolically* each notion which is expressible as a boolean function. In this paper, as it is usually done in Symbolic Model Checking, we are interested in their use for the representation of sets and of binary relations (graphs).

Any boolean function $f(x_1, \ldots, x_k)$ can be represented by a binary tree of height k, whose leaves are labelled by 0 or 1. A path from the root to one leaf represents a boolean assignment $b_1 \ldots b_k$ for the variables $x_1, \ldots, x_k$. The label of the leaf 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. OBDDs are canonical representations for boolean functions

since two boolean functions are equivalent if and only if they are associated to the same OBDD [7].

The way OBDDs are usually employed in Model Checking to represent the state space N, sets of states $S \subseteq N$, and the transition relation E, is based on the following observations [8]:

- we can safely assume that $N = \{0,1\}^u$, i.e. each node is encoded as a binary number;
- a set $S \subseteq N$ is a set of binary strings of length u, characterized 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
$$\chi_E(x_1, \ldots, x_u, y_1, \ldots, y_u) = 1 \Leftrightarrow \langle x_1, \ldots, x_u \rangle E \langle y_1, \ldots, y_u \rangle.$$

As χ_S and χ_E are boolean functions, it is possible to represent them 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.

Various packages have been developed to manipulate OBDDs: Somenzi's CUDD from Colorado University [20], Lind-Nielsen's BuDDy, Biere's ABCD package, Janssen's OBDD package from Eindhoven University of Technology, Carnegie Mellon's OBDD package, the CAL package from Berkeley [18], 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 the OBDDs and to combine them. Here we are interested in some of these operations: the equality test, the boolean operations $\cup, \cap, \setminus$, and in the graph operations img (image computation) and preimg (pre-image computation).

Equality test can be considered a constant time operation. This is possible because if f and g are represented by two OBDDs in the *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 $\setminus$) [7].

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 the relational product and they have a worst-case

[7] If B is an OBDD, then $|B|$ denotes the number of its nodes.

complexity which is exponential w.r.t. $|A|$ and $|G|$. In the practical cases the cost of the operations img and preimg even thought acceptable is the crucial one. Thus, in the area of the symbolic algorithms [19], 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.

6 The Symbolic Rank-based Bisimulation Algorithm

In order to define a symbolic version of the algorithm proposed in [9] (see Figure 1) we mainly need to efficiently compute the rank-partition of the graph. All the other operations involved are standard also in symbolic bisimulation algorithms:

- **collapse**(G, X) (steps (5) and (7.b)) means that all the nodes in X are bisimilar and we do not have to further process them;
- the operations in the **for**-loops at steps (6) and (7.c) 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 at step (7.a) corresponds to the boolean operation $E \restriction B_i(\bar{x}, \bar{y}) = E(\bar{x}, \bar{y}) \wedge (B_i(\bar{x}) \wedge B_i(\bar{y}))$;
- the operation **Bisim**(G_i, D_i) at step (7.a) can be performed by using a symbolic bisimulation algorithm.

For the above reasons in this section we concentrate our efforts on the rank computation.

In the explicit case Tarjan's algorithm ([21]) 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 4.3, through a visit of G. Such a two-step procedure is applicable also symbolically. However, the algorithm in [21] cannot be used as a subroutine. The efficient computation of G^{scc} in [21] relies on the labelling of each node of the input graph. In other words [21] is an explicit algorithm that cannot be translated symbolically. Moreover, the most efficient symbolic algorithm to determine G^{scc}, described in [3], requires $O(|N| \log |N|)$ symbolic steps.

First we rephrase Definition 4.3 exploiting 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 6.1 Let $G = \langle N, E \rangle$. For each node $n \in N$ let $\overline{rank}(n)$ be defined

as follows:

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

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 4.3.

Lemma 6.2 *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 $n, m \in N$ belong to the same strongly connected component, then by Definition 4.3 it holds that $rank(n) = rank(m)$. Since two nodes in the same strongly connected component reach exactly the same nodes, it also holds, by Definition 6.1, that $\overline{rank}(n) = \overline{rank}(m)$. With the above consideration in mind we will proceed in our 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 4.3, 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 4.3 and Definition 6.1 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 to 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 in Figure 2 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 6.3 and 6.4 state the correctness and the complexity of the proposed algorithm.

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 the new nodes*

    (c) $B_j := B_j \cup FRONT$;— *assign rank j to the nodes discovered*

(6) **if** $SET \neq \emptyset$ **then** $B_{-\infty} := SET$;— *rank* $-\infty$ *to the nodes still in SET*

(7) **return** $\{B_{-\infty}, B_0, \ldots, B_\rho\}$;

Fig. 2. The Symbolic Rank algorithm

Theorem 6.3 *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. Consider the set of nodes SET in the **Symbolic Rank** algorithm. Such a set 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 (guard 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 over 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 not 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)–(4.c) of the code 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, with $j+1 \leq max\{rank(n)+1 | n \in WF(G)\}$, of the first loop, 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 is equipped of 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, on the entering to such a loop, SET contains all non-well-founded nodes of N. The first for-loop iteration is executed only if $i \geq 1$ (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 be again used to prove that the j-th 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_∞. We can conclude that $\{B_{-\infty}, B_1, \ldots, B_i\}$ are the classes of the partition over N induced by the *rank*. □

Theorem 6.4 *Let $G = \langle N, E \rangle$ be a graph. The* **Symbolic Rank** *algorithm performs $O(|N|)$ symbolic steps to produce the partition $\{B_{-\infty}, B_0, \ldots, B_\rho\}$ over N.*

Proof. Let γ be the maximum rank of a well-founded node and $M = N \setminus WF(G)$. We will prove that the algorithm in Figure 2 performs at most $O(\gamma + |M|)$ symbolic steps. Trivially $\gamma \leq |WF(G)|$, hence it holds $O(\gamma + |M|) = O(|N|)$. The j-th iteration of the first while-loop discovers exactly those well-founded nodes having rank $j-1$ performing only one symbolic step (line (4.c)). Hence, to execute lines (1)–(4) we perform at most γ symbolic steps. As stated by Theorem 6.3, before entering in the for-loop the set SET is

$N \setminus WF(G) = M$. During each iteration of the innermost while-loop at least one node is removed from SET (line (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 lines (5)–(7) is $O(|M|)$ symbolic steps and we have the thesis. Note that also the number of set-differences, intersections and unions involved in the procedure is $O(|N|)$. □

7 Local Bisimulation Splitting

As we said in Section 3, in [10] 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 are similar to the differences that arises in the explicit case (see [9]). 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 i-th iteration we work on the OBDD's 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 it is possible that in the Paige and Tarjan algorithm these edges are used more times.

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. The Bouali and de Simone algorithm [5] 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 i-th iteration it refines the relation 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 the 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 the relation R_0 approximates the relation $\equiv$, the less iterations are necessary to compute $\equiv$. Hence, once the rank has been symbolically computed we can exploit it to

speed up the Bouali and de Simone algorithm by starting with the relation

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

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

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

8 Rank-based Model Checking

In the previous sections the notion of rank turns out to be at the basis of the development of an efficient bisimulation algorithm. In this section we briefly discuss how the same notion can be exploited in the optimization of the classical Model Checking procedures. We deal with both the explicit and the symbolic Model Checking for CTL [8].

In CTL each temporal operator X (neXt), U (Until), R (Release), F (Finally), and G (Globally), stating how/when a property holds along a path, has to be immediately preceded by one of the path quantifiers A (for all paths) and E (there exists a path). Thus, CTL formulas are built using atomic propositions, the boolean connectives and the operators: AX, EX, AU, EU, AR, ER, AF, EF, AG, and EG.[8]

Standard CTL Model Checking procedures, in both the explicit and the symbolic case, proceed by induction on the subformulas of the input formula: in each iteration the subset of the states satisfying a given subformula is discovered and labelled.

8.1 The Rank in Explicit Model Checking

In the explicit case, the core of the CTL Model Checking procedure [8] is constituted of four subroutines. All of them get as input a CTL formula f and a model whose states are already labelled with the sets of subformulas of f that they satisfy. The first procedure deals with all formulas whose outermost operator is a boolean connective: if, for instance, the input is $f = \neg g$, then upon the return from the subroutine, the states not labelled with g are labelled with f. The second procedure deals with formulas of the kind $f = EXg$: states in the preimage of any state labelled with g are labelled with f. The third and the fourth procedures process formulas whose outermost operators are EU and EG, respectively. In the case of $f = E(hUg)$, first all states labelled with g are labelled with f. Then, recursively, all states labelled with h and contained in the preimage of some state labelled with f, are labelled with f. In case of $f = EGg$, the subgraph M' induced by those states labelled with g

[8] It is sufficient to consider AX, AU, and AG, or equivalently EX, AU, and EG.

is considered. Then, exploiting the algorithm in [21], the strongly connected components of M' are built and each state in a non trivial *scc* is labelled with f. Finally, recursively, all states in the preimage of some state labelled with f are labelled with f.

The above procedures have a time complexity linear in the size of the model. However, they require to keep in the memory the entire system. This could be avoided if we were able to partition the input model in layers over which the computation could be *localized*.

The CTL formulas whose outermost operator is a boolean one obviously could be locally processed using any arbitrary graph stratification. More attention has to be paid to formulas whose outermost operator is one of EX, EU, and EG. For example, consider a model layering ensuring that each path successively encounters lower and lower layers. Just a moment's thought allows to realize that such a stratification permits to localize in successive layers also the determination of the states satisfying formulas of the form EX, EU, and EG. Moreover, in the computation related to a single layer, CTL standard procedures for EX, EU, and EG can be used. It is only necessary to pay attention to correctly initialize the labelling of states in a given layer R_i, provided the labelling on the lower layers $R_{i-1}, \ldots, R_0$ is properly defined. For instance consider the CTL formula $E(hUg)$ and assume to have already determined (i.e. labelled) states in layers $R_0, \ldots, R_{i-1}$ satisfying it. Before applying the classical CTL subroutine for the case EU on R_i we simply have to label with $E(hUg)$ those states in R_i having some successors in a lower layer labelled with $E(hUg)$. Note that this step requires to keep in memory at most two layers of the model at a time.

The notion of rank in Definition 4.3 allows to partition the model in layers over which localizing CTL Model Checking as described above. In order to use a localized CTL Model Checking an alternative notion of rank can be the height of the strongly connected component to which the node belongs. Such a rank definition is suitable for CTL Model Checking since it induces a model stratification such that the layers successively encountered by a path are of decreasing height. This notion of rank performs better than the one in Definition 4.3 relatively to CTL Model Checking. In fact, it induces a finer partition on the model and can be determined with the same time complexity using Tarjan explicit algorithm in [21].

8.2 The Rank in Symbolic Model Checking

Symbolic Model Checking classical routines proceed by determining the subsets of the states satisfying subformulas of the input formula. The above subsets are represented as OBDDs: thus, inexpensive boolean operations on OBDDs (cf. Section 5) allow to deal with subformulas built using propositional connectives. The OBDDs for formulas such as AXg and EXg are obtained symbolically implementing the so-called *relational product* (see [8]). Finally,

symbolic Model Checking, of formulas whose outermost operator is one of AU, EU, AG, and EG, relies on a fix-point characterization of these CTL operators. More specifically, the CTL operators AU and EU can be expressed as least fix-points, whereas AG and EG can be expressed as greatest fix-point. For instance, the fix-point characterization of AGg is the following one:

$$AGg = \nu Z.(g \wedge AXZ).$$

In this section we briefly discuss how to localize the above computation in layers represented using OBDDs, where the graph's layers have the same property required in the explicit case, i.e. they are entered by the paths in a decreasing order. As pointed out in Section 8.1, the notion of rank in Definition 4.3 induces a graph partition with the above property. Moreover, in Section 6 we proved that such a partition can be computed in a linear number of symbolic steps.

Let $B_{-\infty}, B_0(\bar{x}), \ldots, B_\rho(\bar{x})$ be the OBDDs representing the nodes at rank $-\infty, 0, \ldots, \rho$, respectively and $E(\bar{x}, \bar{y})$ be the OBDD representing the relation of the graph. It is possible to partition $E(\bar{x}, \bar{y})$ in the OBDDs:

- $E_{-\infty}(\bar{x}, \bar{y}), E_0(\bar{x}, \bar{y}), \ldots, E_\rho(\bar{x}, \bar{y})$ representing sets of edges among nodes having the same rank, i.e. $E_i(\bar{x}, \bar{y}) = E(\bar{x}, \bar{y}) \wedge (B_i(\bar{x}) \wedge B_i(\bar{y}))$;
- $E \downarrow_1 (\bar{x}, \bar{y}), \ldots, E \downarrow_\rho (\bar{x}, \bar{y})$ representing sets of edges connecting states in a given rank to states in lower ranks, i.e. $E \downarrow_i (\bar{x}, \bar{y}) = E(\bar{x}, \bar{y}) \wedge B_i(\bar{x}) \wedge \bigvee_{j<i} B_j(\bar{y})$.

It is possible to build the OBDDs of states satisfying the CTL formula AGg by ranks: at each rank, i, the OBDDs $B_i(\bar{x}), E_i(\bar{x}, \bar{y})$, and $E \downarrow_i (\bar{x}, \bar{y})$ are considered. The OBDD of the states at rank 0 and satisfying AGg can be built by computing:

$$\nu Z.(B_0 \wedge g \wedge AXZ).$$

Let $AG^* = AG_0 \vee \ldots \vee AG_i$ be the OBDD of all states having rank less than $i+1$ and satisfying AGg. The OBDD of states at rank $i+1$ satisfying AGg can be obtained by computing:

$$K = AX(AG^* \vee B_{i+1}) \qquad \text{and} \qquad \nu Z.(K \wedge g \wedge B_i \wedge AXZ).$$

In the computation of K it is only necessary to use the OBDD $E \downarrow_{i+1} (\bar{x}, \bar{y})$ to implement the AX operation. The AX operation involved in the fix-point can be instead implemented by using $E_{i+1}(\bar{x}, \bar{y})$.

Similar arguments apply to the other CTL operators showing how the notion of *rank* provides a method to distribute the computation on successive layers. Such a layering, on the one hand speeds up the convergence of the fix-point computations. On the other hand, it allows to use the OBDDs representing the *ranks* which could be smaller than the OBDD for the entire model.

9 Conclusions

The notion of rank has been introduced in [9] for developing a fast bisimulation algorithm working on the explicit representation of graphs. This notion, based on the concept of strongly connected component, allows to partition the nodes of a graph and to guide the computation of the bisimulation algorithm. In this work we have shown that the same notion can be used for optimizing bisimulation algorithms on symbolic representation, as well. As a starting point, we have shown how to perform the rank-based partition of a graph in a linear number of symbolic operations. Moreover, we pointed out the usefulness of the same notion for improving explicit and symbolic CTL Model Checking.

Acknowledgement

We thank Elio Panegai for the useful discussions from which this work benefits.

References

[1] Aczel., P., "Non-well-founded sets." Lecture Notes, Center for the Study of Language and Information **14**, Stanford, 1988.

[2] Akers, S. B., *Binary decision diagrams*, IEEE Transaction on Computers **27** (1978), pp. 509–516.

[3] Bloem, R., H. N. Gabow and F. Somenzi, *An algorithm for strongly connected component analysis in $n \log n$ symbolic steps*, in: W. A. Hunt Jr. and S. D. Johnson, editors, *Proc. of Int. Conference on Formal Methods in Computer-Aided Design (FMCAD'00)*, LNCS **1954** (2000), pp. 37–54.

[4] Bouajjani, A., J. C. Fernandez and N. Halbwachs, *Minimal model generation*, in: E. M. Clarke and R. Kurshan, editors, *Proc. Int'l Conference on Computer-Aided Verification CAV90*, LNCS **531** (1990), pp. 197–203.

[5] Bouali, A. and R. de Simone, *Symbolic bisimulation minimization*, in: *Proc. Int'l Conference on Computer-Aided Verification CAV92*, LNCS **663** (1992), pp. 96–108.

[6] Bryant, R. E., *Symbolic manipulation of boolean functions using a graphical representation*, in: *Proc. 22nd Design Automation Conference*, 1985.

[7] Bryant, R. E., *Graph based algorithms for boolean function manipulation*, IEEE Trans. on Computers **C-35** (1986), pp. 677–691.

[8] Clarke, E. M., O. Grumberg and D. A. Peled, "Model checking," MIT Press, 1999.

[9] Dovier, A., C. Piazza and A. Policriti, *A fast bisimulation algorithm*, in: G. Berry, H. Comon and A. Finkel, editors, *Proc. of Int. Conference on Computer Aided Verification (CAV'01)*, LNCS **2102** (2001), pp. 79–90.

[10] Fisler, K. and M. Y. Vardi, *Bisimulation and model checking*, in: *Proc. Correct Hardware Design and Verification Methods*, LNCS **1703** (1999), pp. 338–341.

[11] Hopcroft, J. E., *An nlogn algorithm for minimizing states in a finite automaton*, in: *Theory of Machines and Computations, Ed. by Zvi Kohavi and Azaria Paz*, Academic Press, 1971 pp. 189–196.

[12] Lee, C. Y., *Binary decision programs*, Bell System Technical Journal **38** (1959), pp. 985–999.

[13] Lee, D. and M. Yannakakis, *Online minimization of transition systems*, in: *Proc. 24th ACM Symposium on Theory of Computing*, 1992, pp. 264–274.

[14] Lee, D. and M. Yannakakis, *Online minimization of transition systems*, in: *Proc. of 24th ACM Symposium on Theory of Computing (STOC'92)* (1992), pp. 264–274.

[15] McMillan, K. L., "Symbolic model checking: an approach to the state explosion problem," Kluwer Academic Publishers, 1993.

[16] Paige, R. and R. E. Tarjan, *Three partition refinement algorithms*, SIAM Journal on Computing **16** (1987), pp. 973–989.

[17] Paige, R., R. E. Tarjan and R. Bonic, *A linear time solution to the single function coarsest partition problem*, Theoretical Computer Science **40** (1985), pp. 67–84.

[18] Sanghavi, J. V., R. K. Ranjan, R. K. Brayton and A. Sangiovanni-Vincentelli, *High performance bdd package based on exploiting memory hierarchy*, in: *Proc. of ACM/IEEE Design Automation Conference*, 1996.

[19] Somenzi, F., *Binary decision diagrams* (1999), available at http://citeseer.nj.nec.com/somenzi99binary.html.

[20] Somenzi, F., "CUDD: CU Decision Diagram Package Release 2.3.1," 2001, available at http://vlsi.colorado.edu/ fabio/CUDD/cuddIntro.html.

[21] Tarjan, R. E., *Depth first search and linear graph algorithms*, SIAM Journal on Computing **1** (1972), pp. 146–160.