Journal of Automated Reasoning **31:** 73–103, 2003.

From Bisimulation to Simulation: Coarsest Partition Problems

R. GENTILINI, C. PIAZZA, and A. POLICRITI
Dip. di Matematica e Informatica, Università di Udine, Via Le Scienze, 206, 33100 Udine, Italy.
e-mail: {gentilin,piazza,policrit}@dimi.uniud.it

(Received: 13 August 2003)

Abstract. The notions of *bisimulation* and *simulation* are used for graph reduction and are widely employed in many areas: modal logic, concurrency theory, set theory, formal verification, and so forth. In particular, in the context of formal verification they are used to tackle the so-called state-explosion problem. The faster algorithms to compute the maximum bisimulation on a given labeled graph are based on the crucial equivalence between maximum bisimulation and *relational coarsest partition problem*. As far as simulation is concerned, many algorithms have been proposed that turn out to be relatively inexpensive in terms of either time or space. In this paper we first revisit the state of the art about bisimulation and simulation, pointing out the analogies and differences between the two problems. Then, we propose a generalization of the relational coarsest partition problem, which is equivalent to the simulation problem. Finally, we present an algorithm that exploits such a characterization and improves on previously proposed algorithms for simulation.

Key words: bisimulation, simulation, partition refinement problems.

1. Introduction

The behavior of a system or of a set of programs implementing a collection of cooperating subunits is naturally modeled by structures whose nodes describe the *possible states* and arrows represent *actions*. Even though very old and very much studied, this simple idea, so familiar to computer scientists and logicians, is recurrently at the ground of important progress in a variety of subfields of informatics.

As is natural to expect, the main, somehow intrinsic, difficulties with this approach to modeling are the intricacies buried in the modeling activity and the sheer size of the obtained structures. In fact, the principal purpose of modeling is to be able to perform automatically, within structures, the search for properties, specified in suitably designed languages that can be proved or disproved during the design phase. In this general framework, one of the main tools available to cope with the explosion of the size of the modeling structures (Kripke structures, automata, transition systems of various kind) is algorithms reducing their size before any "reasoning" takes place on them.

In this paper we consider systematically two such possible reductions (*bisimulation* and *simulation*) from an algorithmic point of view. In particular, our purpose is to show how there is a *fil rouge* connecting such notions, which consists of casting them into partition refinement problems (coarsest partition problems). Such a formulation is then showed to be an engine for the design of fast and efficient algorithms along rather natural paths in both cases and, we hope, should also shed some light on the concrete nature of this kind of graph reduction.

The paper continues, after introducing some preliminary material, with a section dealing with the state of the art relative to the study of bisimulation and simulation, respectively, providing a specific preliminary discussion on some history on each of the two notions and the algorithmic tools used for their implementations. In the subsequent section we present a new result in the form of a novel algorithm for simulation, based on the mapping of the notion on a suitable coarsest partition problem.

Preliminary versions of the results presented here were discussed in [20, 19] and [25].

2. Preliminaries

We introduce here the basic notations we use in the rest of the paper.

Let $Q \subseteq T \times T$ be a binary relation over a set T:

- Q is a *quasi order* if and only if Q is reflexive and transitive;
- Q is a *partial order* if and only if Q is an antisymmetric quasi order;
- Q is *acyclic* if and only if its transitive closure is antisymmetric.

We will use Q^+ to refer to the transitive closure of Q and Q^* to refer to the reflexive and transitive closure of Q.

DEFINITION 2.1 (Labeled Graphs). A triple $G = (N, \rightarrow, \Sigma)$ is said to be a *labeled graph* if and only if $G^- = (N, \rightarrow)$ is a finite direct graph and Σ is a partition over N.

Given a node $a \in N$, we denote by $[a]_\Sigma$ (or $[a]$, if Σ is clear from the context) the class of Σ to which a belongs. We say that two nodes $a, b \in N$ have the same *label* if they belong to the same class of Σ.

An equivalent way to define *labeled graphs* is to use a labeling function $\ell \colon N \rightarrow L$, where L is a finite set of labels (inducing a partition Σ_L on N). Given a partition Σ, we will use the letters $\alpha, \beta, \gamma, \ldots$ to denote a generic element, *block*, of Σ.

EXAMPLE 2.2. A *Kripke structure* is a labeled graph and, vice versa, each labeled graph can be seen as a Kripke structure in which two worlds satisfy the same set of atomic propositions if and only if their labels are equal.

DEFINITION 2.3 (Bisimulation). Let $G = (N, \rightarrow, \Sigma)$. A relation $\asymp \subseteq N \times N$ is a *bisimulation* over G if and only if

(1) $a \asymp b \Rightarrow [a]_\Sigma = [b]_\Sigma$;
(2) $(a \asymp b \wedge a \to c) \Rightarrow \exists d(c \asymp d \wedge b \to d)$;
(3) $(a \asymp b \wedge b \to d) \Rightarrow \exists c(c \asymp d \wedge a \to c)$.

Two nodes a and b are *bisimilar* ($a \equiv_b b$) if there is a bisimulation $\asymp$ such that $a \asymp b$.

A bisimulation can be neither reflexive nor transitive, and not even symmetric. However, the reader can easily verify that, given a bisimulation, its reflexive, symmetric, and transitive closure is still a bisimulation. The proof of the following lemma can be found in [41].

LEMMA 2.4. *Let $G = (N, \to, \Sigma)$. The relation $\equiv_b$ is an equivalence relation over N and a bisimulation over G. Moreover, $\equiv_b$ is the maximum bisimulation; that is, if $\asymp$ is a bisimulation over G, then $\asymp \subseteq \equiv_b$.*

Since $\equiv_b$ is an equivalence relation, it is possible to consider the quotient $B = N/{\equiv_b}$. We will use the notation $[a]_b$ to denote the equivalence class to which a belongs in B.

DEFINITION 2.5 (Bisimulation Problem). Given a labeled graph $G = (N, \to, \Sigma)$ the *bisimulation problem* over G consists in computing the quotient $B = N/{\equiv_b}$, where $\equiv_b$ is the maximum bisimulation over G.

DEFINITION 2.6 (Simulation). Let $G = (N, \to, \Sigma)$. A relation $\leq \subseteq N \times N$ is said to be a *simulation* over G if and only if

(1) $a \leq b \Rightarrow [a]_\Sigma = [b]_\Sigma$;
(2) $(a \leq b \wedge a \to c) \Rightarrow \exists d(c \leq d \wedge b \to d)$.

In this case b *simulates* a.

Two nodes a and b are *sim-equivalent* ($a \equiv_s b$) if there exist two simulations $\leq_1$ and $\leq_2$, such that $a \leq_1 b$ and $b \leq_2 a$.

Again, a simulation can be neither reflexive nor transitive (e.g., the empty relation is always a simulation); but, given a simulation, its reflexive and transitive closure is still a simulation.

A simulation $\leq_s$ over G is said to be *maximal* if for all the simulations $\leq$ over G it holds $\leq \subseteq \leq_s$. The proofs of the following results can be found in [38].

LEMMA 2.7. *Given $G = (N, \to, \Sigma)$ there always exists a unique maximal simulation $\leq_s$ over G. Moreover, $\leq_s$ is a quasi order.*

COROLLARY 2.8. *Let $G = (N, \to, \Sigma)$. The relation $\equiv_s$ is an equivalence relation over N.*

Also in the case of simulation, since $\equiv_s$ is an equivalence relation, it is possible to consider the quotient $S = N/{\equiv_s}$. We will use the notation $[a]_s$ to denote the equivalence class to which a belongs in S.

DEFINITION 2.9 (Simulation Problem). Given a labeled graph $G = (N, \to, \Sigma)$, the *simulation problem* over G consists in computing the quotient $S = N/{\equiv_s}$, where $\equiv_s$ is the sim-equivalence over G.

3. State of the Art

3.1. BISIMULATION

The notion of bisimulation, formally defined in Section 2, has been introduced with different purposes in many areas related to computer science. In modal logic it was introduced by van Benthem [45] as an equivalence principle between Kripke structures. In concurrency theory it was introduced by Park [41] for testing observational equivalence of the calculus of communicating systems. In set theory, it was introduced by Forti and Honsell [24] as a natural principle replacing extensionality in the context of nonwell-founded sets. As far as formal verification is concerned (see [11]), several existing verification tools use bisimulation in order to minimize the state spaces of systems descriptions [6, 13, 21, 44], since bisimulation preserves the whole μ-calculus. The reduction of the number of states is important both in compositional and in noncompositional model checking. Bisimulation serves also as a means of checking equivalence between transition systems. In the context of security many noninterference properties are based on checking bisimulation between systems (see [23]).

From a computational point of view, the main reason for the fortune of bisimulation and for its fast solution lies in the equivalence between the bisimulation problem and the problem of determining the *coarsest partition* of a set *stable* with respect to a given relation [33].

DEFINITION 3.1 (Stability). Let $\to$ be a binary relation on N, $\to^{-1}$ be its inverse relation, and Σ be a partition of N. Σ is said to be *stable* with respect to $\to$ iff for each pair α, γ of blocks of Σ, either $\alpha \subseteq \ \to^{-1}(\gamma)$ or $\alpha \ \cap \to^{-1}(\gamma) = \emptyset$.

We say that a partition Π *refines* a partition Σ (Π is *finer* than Σ) if each block of Π is included in a block of Σ.

DEFINITION 3.2 (Coarsest Stable Partition Problem). Given a set N, consider a binary relation $\to$ on N and a partition Σ over N. The *coarsest stable partition problem* is the problem of finding the coarsest partition B refining Σ stable with respect to $\to$.

This problem, which emerged also naturally in automata minimization, is equivalent to the bisimulation problem.

PROPOSITION 3.3. *Let $G = (N, \rightarrow, \Sigma)$ be a labeled graph.*

(i) *Let Π be a partition over N refining Σ and stable with respect to $\rightarrow$. Then $\asymp_\Pi$, defined as $a \asymp_\Pi b$ iff $\exists\alpha \in \Pi\,(a \in \alpha \wedge b \in \alpha)$, is a bisimulation over G.*

(ii) *Let $\asymp$ be a bisimulation over G that is also an equivalence relation. Then $\Pi_\asymp = \{[a]_\asymp \mid a \in N\}$, where $[a]_\asymp = \{b \in N \mid a \asymp b\}$, is a partition stable with respect to $\rightarrow$.*

The next corollary follows immediately.

COROLLARY 3.4. *Let $G = (N, \rightarrow, \Sigma)$. Computing the maximum bisimulation $\equiv_b$ on G or finding the coarsest stable partition of N refining Σ and stable with respect to $\rightarrow$ are equivalent problems.*

The first significant result related to the algorithmic solution of the bisimulation problem is in [31], where Hopcroft presents an algorithm for the minimization of the number of states in a given finite state automaton. Hopcroft's result has been subsequently clarified and improved in [28, 34]. 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 [40], where the authors show how to solve it in linear time in case of a single function. In [32], Kanellakis and Smolka solved the problem for the general case (which is the same as solving the bisimulation problem) in which the stability requirement is relative to a relation $\rightarrow$ (on a set N). The algorithm by Kanellakis and Smolka requires $O(|{\rightarrow}||N|)$ steps and was outperformed by the $O(|{\rightarrow}|\log|N|)$ procedure in [39], realized by Paige and Tarjan. The main feature of the linear solution to the single function coarsest partition problem (cf. [40]) 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 through 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 ("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 [39]. The process the smallest-half policy establishes that if a block α' is split into α and $\alpha' \setminus \alpha$, and α has fewer elements than $\alpha' \setminus \alpha$, then we can use (only) α as splitter and ignore $\alpha' \setminus \alpha$. In their generalization of this policy Paige and Tarjan determine how to perform $|\alpha|$ operations even when it is necessary to use both α and $\alpha' \setminus \alpha$ as splitters.

In [33] Kannellakis and Smolka notice that the algorithm by Paige and Tarjan [39] for the relational coarsest partition problem can be used to determine the maximum bisimulation over a graph.

In [5] Bouajjani, Fernandez, and Halbwachs propose an algorithm for the relational coarsest partition problem tailored for the context of on-the-fly model checking; in others words, they stabilize only *reachable* blocks. In [35] Lee and Yannakakis improve this method by using only reachable blocks to stabilize the reachable blocks.

From a more abstract point of view, an interesting property of the notion of bisimulation is that the bisimulation problem over labeled graphs is equivalent to the one over unlabeled ones. Given a labeled graph $G = (N, \rightarrow, \Sigma)$, it is possible to build a graph $G' = (N', \rightarrow')$, with $N \subseteq N'$ and $\rightarrow \subseteq \rightarrow'$, such that for all $a, b \in N$ $a \equiv_b b$ over G if and only if $a \equiv_b b$ over G' (see [19, 20, 42]). As a matter of fact, in the context of non-well-founded sets, graphs are used to represent sets (see [1]); hence they have no labels, and the notion of bisimulation defines set-equalities. The fact that the problem with labels can be reduced to the one without labels means that the set-theoretic formulation of the bisimulation problem is general enough to embed all the other bisimulation problems.

In [19, 20], exploiting the set-theoretic formulation of the bisimulation problem, an algorithm that optimizes the algorithm in [39] is presented. The worst-case complexity of the algorithm described in [19, 20] is equal to the worst-case complexity of the algorithm proposed in [39], but in a large number of cases it obtains better performances. In particular, this algorithm integrates positive and negative strategies, and the combined strategy is driven by the set-theoretic notion of *rank* of a set. In the case of acyclic graphs the rank of a node a is nothing but the length of the longest path from a to a leaf.

DEFINITION 3.5. Let $G = (N, \rightarrow)$ be an acyclic graph. The *rank* of a node a is recursively defined as follows:

$$\begin{cases} rank(a) = 0 & \text{if } a \text{ is a leaf,} \\ rank(a) = 1 + \max\{rank(c) \mid a \rightarrow c\} & \text{otherwise.} \end{cases}$$

In the general case, the rank of a node a is the length of the longest path from a node c, reachable from a, to a leaf such that all the nodes involved in the path do not reach cycles. To give a formal definition, we introduce the notion of graph of the strongly connected components.

DEFINITION 3.6 (Strongly Connected Components). Given a graph $G = (N, \rightarrow)$, let $G^{\text{scc}} = (N^{\text{scc}}, \rightarrow^{\text{scc}})$ be the graph obtained as follows:

$$N^{\text{scc}} = \{C \mid C \text{ is a strongly connected component in } G\}$$
$$\rightarrow^{\text{scc}} = \{\langle C(a), C(c)\rangle \mid C(a) \neq C(c) \text{ and } a \rightarrow c\}.$$

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

Observe that G^{scc} is acyclic.

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

DEFINITION 3.7 (Well-Founded Part). Let $G = (N, \rightarrow)$ and $a \in N$. $G(a) = (N(a), \rightarrow\lceil N(a))$ is the subgraph[⋆] of G of the nodes reachable from a. $WF(G)$, the well-founded part of G, is $WF(G) = \{a \in N \mid G(a) \text{ is acyclic}\}$.

DEFINITION 3.8 (Rank). Let $G = (N, \rightarrow)$. The *rank* of a node a of G is defined as follows:

$$\begin{cases} \mathit{rank}(a) = 0 & \text{if } a \text{ is a leaf in } G, \\ \mathit{rank}(a) = -\infty & \text{if } C(a) \text{ is a leaf in } G^{\mathrm{scc}} \text{ and } a \text{ is not a leaf in } G, \\ \mathit{rank}(a) = \max(\{1 + \mathit{rank}(c) \mid C(a) \rightarrow^{\mathrm{scc}} C(c), c \in WF(G)\} \cup & \\ \qquad\qquad \{\mathit{rank}(c) \mid C(a) \rightarrow^{\mathrm{scc}} C(c), c \notin WF(G)\}) & \text{otherwise.} \end{cases}$$

Two important properties of the notions of rank, proved in [19, 20], suggest to exploit it in the bisimulation problem:

- if $a \equiv_b b$, then $\mathit{rank}(a) = \mathit{rank}(b)$;
- if $\equiv_b$ has been computed on the nodes of rank less than i, then it is possible to compute $\equiv_b$ on the nodes at rank i.

Given a graph $G = (N, \rightarrow)$ the ranks can be assigned to the nodes of G in time $O(|N| + |\rightarrow|)$ using Tarjan's algorithm for the strongly connected components. This means that using the ranks inside a bisimulation algorithm does not increase its asymptotic time complexity.

The algorithm in [19, 20] first splits the graph into ranks, then uses [40] and [39] as subroutines at subsequent levels, in increasing order of rank. The overall procedure terminates in linear time in many cases, for example when the input problem corresponds to a bisimulation problem on acyclic structures. It 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 the algorithm by Paige and Tarjan [39]. Moreover, the partition imposed by the rank allows to process the input without storing the entire structure in main memory. This allows one (potentially) to deal with larger graphs than those treatable using a Paige and Tarjan-like approach. A symbolic version of the algorithm in [19, 20] has been presented in [18], where it has also been proposed the use the notion of rank inside model checking procedures (i.e., not only to compute the bisimulation quotient, but also to evaluate the CTL formulae).

In [43] the bisimulation problem is tackled again from a set-theoretic point of view. It is shown how in the case of acyclic graphs a compact version of the Ackermann's encoding can be used to solve the bisimulation problem, then the encoding is extended to the general case.

In Section 2 we presented the bisimulation problem over a labeled graph G. It is possible to formulate the problem by using two graphs G_1 and G_2 for which it is convenient to define *labeled graphs* using a labeling function ℓ: $N \rightarrow L$, where L is a finite set of labels common for all the graphs.

[⋆] We use $\rightarrow\lceil N(a)$ to denote the restriction of $\rightarrow$ to $N(a)$.

DEFINITION 3.9 (Two-Graphs Bisimulation). Let $G_1 = (N_1, \rightarrow_1, \ell_1)$ and $G_2 = (N_2, \rightarrow_2, \ell_2)$ be two labeled graphs. A relation $\asymp \subseteq N_1 \times N_2$ is a *bisimulation* between G_1 and G_2 if and only if

(1) $a \asymp b \Rightarrow \ell_1(a) = \ell_2(b)$;
(2) $(a \asymp b \wedge a \rightarrow c) \Rightarrow \exists d(c \asymp d \wedge b \rightarrow d)$;
(3) $(a \asymp b \wedge b \rightarrow d) \Rightarrow \exists c(c \asymp d \wedge a \rightarrow c)$;
(4) for each $a \in N_1$ there exists $b \in N_2$ such that $a \asymp b$;
(5) for each $b \in N_2$ there exists $a \in N_1$ such that $a \asymp b$.

Two graphs G_1 and G_2 are *bisimilar* if there exists a bisimulation between G_1 and G_2.

Either definition of bisimulation allows us to quotient (reduce) G on the ground of the following property of $\equiv_b$ (and of all the bisimulations that are equivalence relations):

$$\forall a, b, c\ ((a \rightarrow c \wedge b \in [a]_b) \Rightarrow \exists d(d \in [c]_b \wedge b \rightarrow d)). \qquad (\flat)$$

The quotient graph $G/{\equiv_b} = (N_{\equiv_b}, \rightarrow_{\equiv_b}, \ell_{\equiv_b})$ is defined as

$$N_{\equiv_b} = N/\equiv_b,$$
$$[a]_b \rightarrow_{\equiv_b} [c]_b \Leftrightarrow \exists c_1(c_1 \in [c]_b \wedge a \rightarrow c_1),$$
$$\ell_{\equiv_b}([a]_b) = \ell(a).$$

PROPOSITION 3.10. *Let $G = (N, \rightarrow, \ell)$ be a labeled graph. The graph $G/{\equiv_b}$ is the minimum graph bisimilar to G.*

3.2. SIMULATION

The notion of simulation, introduced in Section 2 and first defined by Milner in [37] as a means to compare programs, is very close (less demanding, in fact) to the notion of bisimulation. Since the conditions in the definition of simulation are weaker than the ones in the definition of bisimulation, simulation provides, for example, in the context of formal verification, a better space reduction tool than bisimulation. Nevertheless, simulation is still adequate for the verification of all formulae of the branching temporal logic without quantifiers switches [16] (e.g., the formulae of ACTL*; see [29]). As explained in [30], “in many cases, neither trace equivalence nor bisimilarity, but similarity is the appropriate abstraction for computer-aided verification ....” In the case of finite-state systems the similarity quotients can be computed in polynomial time, while this is not the case for trace equivalence quotients. On infinite-state systems, finitely represented using hybrid automaton and other formalisms, the similarity quotients can be computed symbolically, and in many cases the quotients are finite (see [30]).

Several polynomial-time algorithms to solve the simulation problem on finite graphs have been proposed: the ones in [3, 12, 14] achieve $O(|N|^6|\rightarrow|)$, $O(|N|^4|\rightarrow|)$, and $O(|\rightarrow|^2)$ time complexities, respectively. A simulation procedure running in $O(|N||\rightarrow|)$ time was independently discovered in [30] and [4]. All of the algorithms just mentioned [3, 4, 12, 14, 30] obtain the similarity quotient $S = N/{\equiv_s}$ as a by-product of the computation of the entire similarity relation $\leq_s$ on the set of states N. Their space complexity is then limited from below by $O(|N|^2)$.

Recently Bustan and Grumberg in [9] and Cleaveland and Tan in [15] improved the above results.

The procedure by Bustan and Grumberg [9] gives in the output the quotient structure with respect to $\equiv_s$ and the simulation relation among classes of $S = N/_{\equiv_s}$ without computing the entire simulation on N. Hence, its space requirements (often critical, especially in the field of verification) depend on the size of S and are lower than the ones of the algorithms in [3, 4, 12, 14, 30]. In more detail, the algorithm described in [9] uses only $O(|S|^2+|N|\log(|S|))$ space, whereas its time complexity is rather heavy: it is $O(|S|^4(|\rightarrow| + |S|^2) + |S|^2|N|(|N| + |S|^2))$.

The procedure in [15] combines the fix-point calculation techniques in [4] and [30] with the bisimulation-minimization algorithm in [39]. It is determined whether a system G_2 is capable of simulating the behavior of G_1, by interleaving the minimization via bisimulation of the two systems with the computation of the set of classes in G_2 able to simulate each class in G_1. The time complexity achieved is $O(|B_1||B_2|+|\rightarrow_1|\log(|N_1|)+|B_1||\rightarrow_2|+|\varepsilon_1||B_2|)$, where ε_i and B_i represent the bisimulation reduced relation and state-space of T_i, respectively. Compared with the time complexities of [30] and [4], the latter expression has many occurrences of $|N_i|$ and $|\rightarrow_i|$ replaced with $|B_i|$ and $|\varepsilon_i|$. Indeed, experimental results in [15] prove that the procedure by Cleaveland and Tan outperforms the ones in [30] and [4]. The space complexity of [15] depends on the product of the sizes of the two bisimulation quotients involved. Bisimulation being an equivalence relation finer than simulation, such a space requirement may be more demanding than the one in [9].

As in the case of bisimulation, it is possible to formulate the simulation problem between two graphs. In particular, a "two graphs" formulation is used in [15].

DEFINITION 3.11 (Two-Graphs Simulation). Let $G_1 = (N_1, \rightarrow_1, \ell_1)$ and $G_2 = (N_2, \rightarrow_2, \ell_2)$ be two labeled graphs. A relation $\leq \subseteq N_1 \times N_2$ is a *simulation* from G_1 to G_2 if and only if

(1) $a \leq b \Rightarrow \ell_1(a) = \ell_2(b)$;

(2) $(a \leq b \wedge a \rightarrow c) \Rightarrow \exists d(c \leq d \wedge b \rightarrow d)$;

(3) for each $a \in N_1$ there exists $b \in N_2$ such that $a \leq b$.

A graph G_1 is *simulated* by a graph G_2 (G_2 simulates G_1) if there exists a simulation from G_1 to G_2. Two graphs G_1 and G_2 are sim-equivalent if G_1 simulates G_2 and G_2 simulates G_1.

Hence, given a graph G, one can consider the problem of determining the minimum graph sim-equivalent to G. In the context of simulation, minimality is measured in terms of the number of both nodes and edges. In [9] it has been proved that there always exists a unique smallest labeled graph that is sim-equivalent to G; that is, there is a unique way to put a minimum number of edges between the elements of $S = N/{\equiv_s}$ in order to obtain a labeled graph sim-equivalent to G. In the case of bisimulation this was a trivial consequence of the property ($\flat$) (see Section 3.1), since if a node reaches with an edge an equivalence class, then all the nodes that are equivalent to it reach with an edge the same equivalence class. This is equivalent to saying that

$$\forall a, c \quad ([a]_b \cap \to^{-1}([c]_b) \neq \emptyset \Rightarrow [a]_b \subseteq \to^{-1}([c]_b)),$$

which is at the basis of the characterization of the bisimulation problem as (relational) coarsest partition problem. In the case of simulation there may exist a node a that reaches with an edge a node c, while a node that is sim-equivalent to a does not reach any node sim-equivalent to c. When this situation occures, we always have that there exists a node d such that a reaches d and $c \leq_s d$, and we say that c is a *little brother* of d. Hence, the property that holds for the maximum simulation is that

$$\forall a, b, c \quad ((a \to c \wedge b \in [a]_s) \Rightarrow \exists d(c \leq_s d \wedge b \to d)), \qquad (\sharp)$$

which in particular, since we are working on finite graphs, implies that

$$\forall a, c \quad ([a]_s \cap \to^{-1}([c]_s) \neq \emptyset \Rightarrow \exists [d]_s([c]_s \leq_s [d]_s \wedge [a]_s \subseteq \to^{-1}([d]_s))).$$

This property, at the basis of the *generalized stability* condition that we will introduce in the second part of this paper, provides a characterization of the simulation problem in terms of a partitioning problem. An immediate consequence is that there is a unique way to put a minimum number of edges among the elements of $N/{\equiv_s}$ in order to obtain a graph sim-equivalent to G: put an edge from $[a]_s$ to $[d]_s$ if and only if it holds $[a]_s \subseteq \to^{-1}([d]_s)$ and there is no class $[c]_s$ such that $[a]_s \cap \to^{-1}([c]_s) \neq \emptyset$ with $d \leq_s c$ (i.e., d is not a little brother of any c).

EXAMPLE 3.12. Consider the two graphs in Figure 1. The graph on the right is the quotient with respect to the maximum simulation of the graph on the left. In the graph on the left there are no sim-equivalent nodes; hence all the classes in the quotient structure are singletons. In the graph on the left there is a node with label B that is a little brother; hence in the quotient structure we delete an edge.

A further important difference between bisimulation and simulation is that, while in the case of bisimulation working without labels is not a restriction (see Section 3.1), in the case of simulation it is. This is a consequence of the following result whose proof is left to the reader.

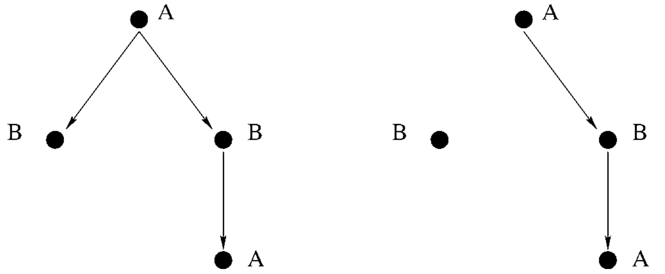


Figure 1. Example of little brother and of minimization in terns of edges.

LEMMA 3.13. *Let $G = (N, \to, \Sigma)$ be a graph such that $\Sigma = \{N\}$, that is, all the nodes have the same label. Given a node $a \in N$, let $G(a)$ be the subgraph of G of the nodes reachable from a.*

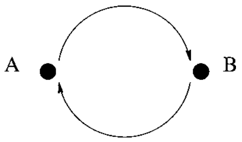
- *If $G(a)$ is acyclic, then for all $b \in N$*

$$b \leq_s a \quad \text{iff } G(b) \text{ is acyclic and } rank(b) \leq rank(a).$$

- *If $G(a)$ is cyclic, then for all $b \in N$ it holds $b \leq_s a$.*

This means that in the case without labels (i.e., only one label) all the nodes that belong to the non-well-founded part of the graph are sim-equivalent.

EXAMPLE 3.14. Consider the graph in the figure below on the left.



If we try to remove the labels A and B by adding new nodes and we do not want to use labels also on the new nodes, then, no matter how many new nodes we add, the two nodes of the original graph become sim-equivalent.

The previous lemma has also strong implications on the use of the notion of rank in the computation of $\equiv_s$. It implies that

$$a \equiv_s b \not\Rightarrow rank(a) = rank(b),$$

since in a graph with only one label all the non-well-founded nodes are sim-equivalent even if they are at different ranks. Lemma 3.13 suggests that a reasonable definition of rank to be used in the computation of $\equiv_s$ could be

$$rank^*(a) = \begin{cases} \max\{1 + rank(c) \mid C(a) \to^{\text{scc}} C(c)\} & \text{if } a \in WF(G); \\ +\infty & \text{otherwise;} \end{cases}$$

that is, on the well-founded part of the graph we use the usual notion of rank, while all the nodes in the nonwell-founded part have rank $+\infty$. Using such a definition, we obtain

$$a \equiv_s b \Rightarrow rank^*(a) = rank^*(b);$$
$$a \leq_s b \Rightarrow rank^*(a) \leq rank^*(b).$$

This implies that, whenever we use a technique that needs to compute $\leq_s$ in order to determine $\equiv_s$, when we process nodes at rank i, we have to take into account also all the nodes at rank less than i.

A second disadvantage of the use of *rank** is that it is reasonable to believe that at least one-half of the nodes have rank $+\infty$, which means that it does not split the computation enough.

For this last reason, in order to exploit a notion of rank in the simulation computation, the opposite approach is taken in [26]; that is, a notion of rank is introduced that splits the graph as much as possible, admitting to have sim-equivalent nodes with different ranks. In any case, what is required for the notion of rank is the property that the information at rank i is enough to compute the similarity relations among the nodes of rank at most i. In particular, we want to avoid the situation in which a and b are two nodes whose rank is at most i, at the end of the ith iteration we "believe" that $a \equiv_s b$ (resp. $a \not\equiv_s b$), and at the end of the $(i+k)$th iteration we discover that $a \not\equiv_s b$ (resp. $a \equiv_s b$). A necessary and sufficient condition to obtain the above property is that

$$\text{if } a \to b, \quad \text{then the rank of } b \text{ is not greater than the rank of } a.$$

The following notion of rank turns out to satisfy the above condition and splits the graph as much as possible.

DEFINITION 3.15 (sim-Rank). Let $G = (N, \to, \Sigma)$, the $rank_s$ of a node is recursively defined as follows:

$$rank_s(a) = \begin{cases} 0 \quad \text{if } C(a) \text{ is a leaf in } G^{\text{scc}}, \\ \max\{1 + rank_s(c) \mid C(a) \to^{\text{scc}} C(c)\} \quad \text{otherwise.} \end{cases}$$

This means that at each rank a set of strongly connected components is considered and the order in which these sets of strongly connected components are considered is determined by the usual (well-founded) notion of rank on the graph of the strongly connected components.

3.3. MODEL CHECKING AND SYMBOLIC ALGORITHMS

As already said, one of the main applications of the notions of bisimulation and simulation comes from the area of formal verification. The main disadvantage of model checking is the so-called state explosion that can occur if the system being verified has many components that can make no parallel transitions. In this case the number of global system states may grow exponentially with the number of processes. Different techniques have been developed and are still under active investigation to solve this problem: OBDD and symbolic model checking [36], abstract model checking [17], partial order reductions [11], and equivalence reductions [30, 35]. Bisimulation and simulation belong to this last group of techniques, since they allow to obtain a smaller structure that satisfies the same properties of the input one (preservation of various logics). Bisimulation guarantees preservation of branching-time temporal logics such as CTL and CTL* [10], while simulation preserves the universal fragment of these logics (ACTL and ACTL* [29]).

For this reason many model-checking tools integrate subroutines to perform both bisimulation and simulation reductions [6, 13, 21, 44]. The verification environment XEVE [6] provides bisimulation tools that can be used for both minimization and equivalence test. The Concurrency Workbench (CWB) [12] tests bisimulation using techniques based on 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) [13] the underling bisimulation algorithm is Paige and Tarjan's, while the simulation algorithms used are the ones presented in [4, 15].

Sometimes, different techniques are put together in order to obtain better reductions in memory requirements. In particular, symbolic model checking, which is based on the use of ordered binary decision diagrams (OBDDs) (see [8, 36]) to represent the input structure and sets of states, is sometimes associated to the use of bisimulation and simulation. Unfortunately, not all the algorithms fit with the use of OBDDs. The main advantage of OBDDs is that they compactly represent large sets of states; hence, it is not reasonable to use algorithms that need to consider each state separately. A classical example is Tarjan's algorithm (or any version of it) for the strongly connected components, which is inapplicable on OBDDs because it requires manipulating each node singularly in order to assign it its finishing-time.

In the *symbolic* case, that is, when OBDDs are used as basic data-structures, a popular bisimulation algorithm is [7] by Bouali and de Simone. This procedure implements the naïve negative strategy by 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 $\langle a, b \rangle$ for which it is possible to prove that a is not bisimilar to b are removed. In [7] experimental results on the performances of the algorithm are presented, while there is no through discussion of its complexity in terms of basic OBDD operations.

In [22] Fisler and Vardi analyze the complexity of the symbolic versions of the algorithms by Paige and Tarjan [39]; Bouajjani, Fernandez, and Halbwachs [5]; and Lee and Yannakakis [35]. They determine the number of basic symbolic operations involved in each iteration by the three algorithms and conclude, through experimental results, that an optimized version of the algorithm in [39] (splitting only reachable blocks) performs better than the other two algorithms because it mostly gains from the *good* choice of the splitters.

In [18] a symbolic version of the algorithm previously presented in [19, 20] is proposed. The main problem in making the algorithm in [19, 20] symbolic is that the strongly connected components are used to compute the ranks and, in the symbolic case, the computation of the strongly connected components would require $O(|N| \log |N|)$ operations on OBDDs [2]). For this reason, [18] defines a procedure that computes the ranks in O($|N|$) steps, avoiding the computation of the strongly connected components.

In [30] a symbolic version of their simulation algorithm is also discussed. Even if in [9] the authors do not consider this issue, their algorithm seems to be particularly suitable for a symbolic implementation, since it always works on classes of nodes. Such a nice feature is shared also by the algorithm we present in the next section.

4. A New Simulation Algorithm

In this section we study the simulation problem in a general setting. This study leads us to the definition of a generalization of the coarsest partition problem, which is at the basis of our time/space efficient simulation algorithm. A preliminary version of the algorithm we describe here has been presented in [25]. The detailed proofs of the results presented in this section can be found in [27].

4.1. SIMULATIONS AS PARTITIONING PROBLEMS

We introduce the *generalized coarsest partition problem* (GCPP), which is the central notion in our approach. The term *generalized* is used because we are going to deal not only with partitions to be refined, as in the classical coarsest partition problems (see Section 3.1), but also with *pairs* in which we have a partition and a binary relation over the partition. The equivalence between the simulation problem and the GCPP will be proved at the end of this section.

DEFINITION 4.1 (Partition Pair). Let $G = (N, \rightarrow)$ be a finite graph. A *partition pair* over G is a pair $\langle \Sigma, P \rangle$ in which Σ is a partition over N, and $P \subseteq \Sigma^2$ is a reflexive and acyclic relation over Σ.

Notice that a labeled graph $G = (N, \rightarrow, \Sigma)$ can be seen as a graph $G' = (N, \rightarrow)$ together with the partition pair $\langle \Sigma, I \rangle$, where I is the identity relation over Σ. Given a graph $G = (N, \rightarrow, \Sigma)$, we will start considering the partition pair $\langle \Sigma, I \rangle$, and we will modify it in order to obtain a partition pair $\langle S, \preceq \rangle$, where S is the simulation quotient $N/{\equiv_s}$, while $\preceq$ is the maximal simulation over $N/{\equiv_s}$.

Given two partitions Π and Σ, such that Π is finer than Σ (i.e., each block of Π is included in a block of Σ), and a relation P over Σ, we use the notation $P(\Pi)$ to refer to the relation *induced* on Π by P:

$$\forall \alpha, \beta \in \Pi((\alpha, \beta) \in P(\Pi) \Leftrightarrow \exists \alpha', \beta'((\alpha', \beta') \in P \wedge \alpha \subseteq \alpha' \wedge \beta \subseteq \beta')).$$

Denoting by $\mathcal{P}(G)$ the set of partition pairs over G below, we introduce the partial order over which we will define the notion of coarsest partition pair.

DEFINITION 4.2. Let $\langle \Sigma, P\rangle, \langle \Pi, Q\rangle \in \mathcal{P}(G)$:

$$\langle \Pi, Q\rangle \sqsubseteq \langle \Sigma, P\rangle \quad \text{iff } \Pi \text{ is finer than } \Sigma \text{ and } Q \subseteq P(\Pi).$$

The search for the coarsest partition pair will proceed by using the following transition relations.

DEFINITION 4.3 ($\rightarrow_\exists, \rightarrow_\forall$). Let $G = (N, \rightarrow)$, and Π be a partition of N.
The $\exists$-*transition* on Π is the relation

$$\alpha \rightarrow_\exists \gamma \quad \text{iff } \exists a \exists c (a \in \alpha \wedge c \in \gamma \wedge a \rightarrow c).$$

The $\forall$-*transition* on Π relation

$$\alpha \rightarrow_\forall \gamma \quad \text{iff } \forall a (a \in \alpha \Rightarrow \exists c (c \in \gamma \wedge a \rightarrow c)).$$

Hence, $\alpha \rightarrow_\forall \gamma$ is a shorthand for $\alpha \subseteq \rightarrow^{-1}(\gamma)$, while $\alpha \rightarrow_\exists \gamma$ stands for $\alpha \cap \rightarrow^{-1}(\gamma) \neq \emptyset$. Similar notations (called *relation transformers* and *abstract transition relations*) were introduced in [17] in order to combine model checking and abstract interpretation.

We are now ready to introduce the fundamental notion in the generalized coarsest partition problems, the notion of stability of a partition pair with respect to a relation.

DEFINITION 4.4 (Stability). Let $G = (N, \rightarrow)$. A partition pair $\langle \Sigma, P\rangle$ over G is *stable* with respect to the relation $\rightarrow$ if and only if

$$\forall \alpha, \beta, \gamma \in \Sigma((\alpha, \beta) \in P \wedge \alpha \rightarrow_\exists \gamma \Rightarrow \exists \delta \in \Sigma((\gamma, \delta) \in P \wedge \beta \rightarrow_\forall \delta)).$$

The condition in the definition of stability is equivalent to

$$\begin{aligned}&\forall \alpha, \beta, \gamma \in \Sigma((\alpha, \beta) \in P \wedge \alpha \cap \rightarrow^{-1}(\gamma) \neq \emptyset \\ &\quad \Rightarrow \exists \delta \in \Sigma((\gamma, \delta) \in P \wedge \beta \subseteq \rightarrow^{-1}(\delta))).\end{aligned}$$

As will be proved (see Theorem 4.10), the stability condition is exactly the condition holding between two classes of $N/{\equiv_s}$: if $\alpha, \beta \in N/{\equiv_s}$ with $\alpha \leq_s \beta$ (i.e., all the elements of α are simulated by all the elements of β) and an element a in α reaches an element c in γ, then all the elements b of β must reach at least one element d that simulates c. In particular, considering all the $\leq_s$-maximal elements d simulating c reached by elements in β, we have that all the elements in β reach a class δ that simulates c and, hence, that simulates γ.

EXAMPLE 4.5. Consider the graph G depicted on the left in Figure 2 and the partition pair $\langle \Sigma, I\rangle$, where $\Sigma = \{\alpha, \beta\}$, with α and β shown in Figure 2, and I is the identity relation. The partition pair $\langle \Sigma, I\rangle$ is not stable because $\beta \rightarrow_\exists \alpha$, α is the only class in Σ such that $(\alpha, \alpha) \in I$, and it does not hold $\beta \rightarrow_\forall \alpha$.

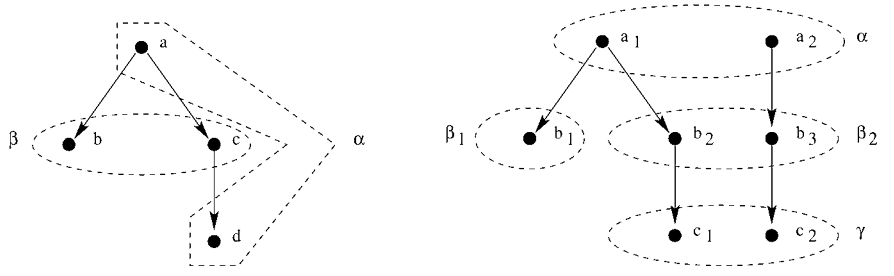


Figure 2. G and Σ on the left, G' and Π on the right.

Consider now the graph G' depicted on the right in Figure 2 and the partition pair $\langle \Pi, P\rangle$, where $\Pi = \{\alpha, \beta_1, \beta_2, \gamma\}$ is shown in Figure 2, while $P = \{(\beta_1, \beta_2)\} \cup I$, with I identity relation. It is easy to prove that $\langle \Pi, P\rangle$ is stable.

If the the partition $\{\{a_1, a_2\}, \{b_1, b_2, b_3\}, \{c_1, c_2\}\}$ represents the labeling on G', then Π and P correspond to the simulation equivalence and the maximal simulation on G', respectively.

We use the notion of stability of a partition pair with respect to a relation in the definition of generalized coarsest partition problem, in the same way as the notion of stability of a partition with respect to a relation is used in the definition of coarsest partition problem.

DEFINITION 4.6 (Generalized Coarsest Partition Problem). Given a graph G and partition pair $\langle \Sigma, P\rangle$ over G, the *generalized coarsest partition problem* consists in finding a partition pair $\langle S, \preceq\rangle$ such that

(a) $\langle S, \preceq\rangle \sqsubseteq \langle \Sigma, P^+\rangle$;
(b) $\langle S, \preceq\rangle$ is stable with respect to $\rightarrow$;
(c) $\langle S, \preceq\rangle$ is $\sqsubseteq$-maximal satisfying (a) and (b).

The partition pair $\langle S, \preceq\rangle$ is a *stable refinement* of $\langle \Sigma, P\rangle$.

Notice that in the above definition $\langle S, \preceq\rangle$ is a refinement of $\langle \Sigma, P^+\rangle$, while it can be the case that it is not a refinement of $\langle \Sigma, P\rangle$. This ensures that $\preceq$ is transitive. In general, the solution of the GCPP can always be found by a suitable sequence of splits (of classes) and adequate completion of the relation in order to take the newly generated classes into account.

Remark 4.7. It is important that $\preceq$ be reflexive (which is the case because $\langle S, \preceq\rangle$ is a partition pair). This is necessary in order to prove that there is always a unique maximal solution. Consider the graph $G = (N, \rightarrow)$ with $N = \{a, b, c\}$ and $\rightarrow = \{(a, a), (a, b), (b, c)\}$, and the partition pair $\langle \Sigma, P\rangle$ with $\Sigma = \{N\}$ and $P = \{(N, N)\}$. Assume we would not require, in the solution to the GCPP, to have a reflexive relation on the final partition. Then, we would have the following four maximal solutions to the GCPP:

- $\langle \Pi = \{\alpha, \beta, \gamma\}, Q = \{(\gamma, \beta), (\beta, \alpha), (\alpha, \alpha), (\beta, \beta), (\gamma, \gamma)\}\rangle$, where $\alpha = \{a\}, \beta = \{b\}, \gamma = \{c\}$;
- $\langle \Pi' = \{\alpha', \beta'\}, Q' = \{(\beta', \alpha'), (\alpha', \alpha')\rangle$, where $\alpha' = \{a\}, \beta' = \{b, c\}$;
- $\langle \Pi'' = \{\alpha'', \beta''\}, Q'' = \{(\beta'', \beta''), (\beta'', \alpha'')\}\rangle$, where $\alpha'' = \{a, b\}, \beta'' = \{c\}$;
- $\langle \Sigma, \emptyset\rangle$.

Note that only the first pair enjoys reflexivity and is a partition pair.

Our next task is to prove that a given GCPP has always a unique solution. To this end we exploit a connection between the simulation problems and the GCPP's. All the details about this connection can be found in [27].

THEOREM 4.8 (Existence and Uniqueness). *Given $G = (N, \rightarrow)$ and a partition pair $\langle \Sigma, P\rangle$ over G, the GCPP over G and $\langle \Sigma, P\rangle$ has always a unique solution.*

Again, by exploiting the connection between the simulation problems and the GCPPs, we can prove the following result.

COROLLARY 4.9. *If $\langle S, \preceq\rangle$ is the solution of the GCPP over G and $\langle \Sigma, P\rangle$, then $\preceq$ is a partial order over S.*

Hence, each generalized coarsest partition problem has a unique solution that can be determined by solving a simulation problem. The following easily proved result states that the converse also holds.

THEOREM 4.10 (Simulation as GCPP). *Let $G = (N, \rightarrow, \Sigma)$ and let $\langle S, \preceq\rangle$ be the solution of the GCPP over $G^- = (N, \rightarrow)$ and $\langle \Sigma, I\rangle$, where I is the identity relation. S is the simulation quotient of G, that is, $S = N/{\equiv_s}$, and $\leq_s$ defined as*

$$a \leq_s b \quad \textit{iff} \quad [a]_s \preceq [b]_s$$

is the maximal simulation over G.

By the above theorem, in order to solve the problem of determining the simulation quotient of a labeled graph $G = (N, \rightarrow, \Sigma)$ we can equivalently solve the generalized coarsest partition problem over $(N, \rightarrow)$ and $\langle \Sigma, I\rangle$.

4.2. SOLVING THE GENERALIZED COARSEST PARTITION PROBLEMS

To give an operational content to the results in the preceding section, we now introduce an operator σ, mapping partition-pairs into partition-pairs, which will turn out to be the engine of our algorithm. A procedure that computes σ will be used to solve GCPPs and, hence, to compute similarity quotients: it is necessary to iterate the computation of σ at most only $|S|$ times.

The operator σ is defined in such a way that it refines the partition-pair $\langle\Sigma, P\rangle$ to obtain a partition-pair $\langle\Pi, Q\rangle$ that is *more stable* than $\langle\Sigma, P\rangle$ and is never finer than the solution of the GCPP over $\langle\Sigma, P\rangle$.

Three conditions are used to specify σ. The first one we impose simply to split those classes of Σ that do not respect the stability condition with respect to themselves: If a class α is such that $\alpha \to_{\exists} \gamma$ and there does not exist a class δ such that $(\gamma, \delta) \in P$ and $\alpha \to_{\forall} \delta$, then the pair (α, α) does not respect the stability condition; hence we must split α. The first condition is used to build Π, while the second and the third conditions in σ are used to define Q on Π. Intuitively, the second and the third conditions allow one to obtain Q from P by starting from $P(\Pi)$ and removing the maximum number of pairs that contradict stability.

DEFINITION 4.11 (Operator σ). Let $G = (N, \to)$ and $\langle\Sigma, P\rangle$ be a partition pair over G. The partition pair $\langle\Pi, Q\rangle = \sigma(\langle\Sigma, P\rangle)$ is defined as follows:

(1σ) Π is the coarsest partition finer than Σ such that

(a) $\forall\alpha \in \Pi\, \forall\gamma \in \Sigma(\alpha \to_{\exists} \gamma \Rightarrow \exists\delta \in \Sigma((\gamma, \delta) \in P \wedge \alpha \to_{\forall} \delta))$;

(2σ) Q is maximal such that $Q \subseteq P(\Pi)$ and if $(\alpha, \beta) \in Q$, then

(b) $\forall\gamma \in \Sigma(\alpha \to_{\forall} \gamma \Rightarrow \exists\gamma' \in \Sigma((\gamma, \gamma') \in P \wedge \beta \to_{\exists} \gamma'))$ and
(c) $\forall\gamma \in \Pi(\alpha \to_{\forall} \gamma \Rightarrow \exists\gamma' \in \Pi((\gamma, \gamma') \in Q \wedge \beta \to_{\exists} \gamma'))$.

By abuse of notation we use $\to_{\exists}$ and $\to_{\forall}$ also when the classes belong to different partitions.

Condition (a) in (1σ) is imposed to suitably split the classes of the partition Σ: these splits are forced by the fact that we are looking for a partition-pair *stable* with respect to the relation of the given graph and we know that in each partition-pair $\langle\Sigma, P\rangle$ the second component is reflexive. Using condition (b) in (2σ), together with condition (a) in (1σ) and exploiting the fact that P is acyclic, we can prove that Q is acyclic: the acyclicity of P ensures that a cycle could arise only among classes of Π that are all contained in a unique class of Σ. Then, using condition (a) in (1σ) and (b) in (2σ), we obtain a contradiction. Condition (c) is fundamental, together with condition (a), in order to obtain the result in Theorem 4.14: If it holds that $\alpha \to_{\forall} \gamma$, then *no matter how we split* α one of the subclasses generated from α has a chance (in the solution of GCPP) to be in relation with at least one of the subclasses generated from β only if $\beta \to_{\exists} \gamma'$ and γ is in relation with γ'. The following results (proved in [27]) guarantee the correctness of σ and, hence, of our approach.

LEMMA 4.12 (Existence). *Let $G = (N, \to)$ and $\langle\Sigma, P\rangle$ be a partition pair over G. There exists at least one partition pair $\langle\Pi, Q\rangle$ that satisfies the conditions in Definition* 4.11*; that is, σ is always defined.*

THEOREM 4.13 (Uniqueness). *Let $G = (N, \to)$ and $\langle\Sigma, P\rangle$ be a partition pair over G. There exists a unique maximal pair $\langle\Pi, Q\rangle$ that satisfies the conditions in Definition* 4.11*; that is, σ is a function.*

Fix points of the σ operator can be used to compute the solution of the GCPP.

THEOREM 4.14 (Fix Point). *Let $G = (N, \to)$ and $\langle \Sigma, P \rangle$ be a partition pair over G with P transitive. Let $\langle S, \preceq \rangle$ be the solution of the GCPP over G and $\langle \Sigma, P \rangle$. If n is such that $\sigma^n(\langle \Sigma, P \rangle) = \sigma^{n+1}(\langle \Sigma, P \rangle)$, then $\sigma^n(\langle \Sigma, P \rangle) = \langle S, \preceq \rangle$.*

Estimating how large is the index n in the worst case allows us to point out another important property of the operator σ. When we iteratively apply the operator σ until we reach a fix point, at each iteration we refine a partition and we remove pairs from a relation. What we prove in the following theorem is that at each iteration we do refine the partition. In a certain sense this means that the two conditions we have given in (2σ) to remove pairs are *optimal*.

THEOREM 4.15 (Complexity). *Let $G = (N, \to)$ and $\langle \Sigma, P \rangle$ be a partition pair over G with P transitive. Let $\langle S, \preceq \rangle$ be the solution of the GCPP over G and $\langle \Sigma, P \rangle$. If i is such that $\sigma^i(\langle \Sigma, P \rangle) = \langle \Sigma_i, \mathsf{P}_i \rangle$ and $\sigma^{i+1}(\langle \Sigma, P \rangle) = \langle \Sigma_i, \mathsf{P}_{i+1} \rangle$, then $\mathsf{P}_{i+1} = \mathsf{P}_i$ and $\langle \Sigma_i, \mathsf{P}_i \rangle = \langle S, \preceq \rangle$.*

COROLLARY 4.16. *The solution $\langle S, \preceq \rangle$ of the GCPP over a graph G and a partition pair $\langle \Sigma, P \rangle$ can be computed iterating σ at most $|S|$ times.*

The characterizations obtained in this section allow us to conclude that if we are able to define a procedure that, given a partition-pair $\langle \Sigma, P \rangle$, computes $\sigma(\langle \Sigma, P \rangle)$, then we can use it to solve GCPPs and hence to compute similarity quotients. In particular, we recall that given a similarity quotient problem over a labeled graph $G = (N, \to, \Sigma)$ in order to solve it, one need only to solve the GCPP over G and $\langle \Sigma, I \rangle$ (notice that I is trivially transitive).

Moreover, the last corollary ensures that it will be necessary to iterate the procedure that computes σ at most $|S|$ times. This is a first improvement on time complexity with respect to the algorithm presented in [9]: in a certain sense [9] computes an operator that refines the partition-pair less than σ, and hence it is possible that the computation has to be iterated up to $|S|^2$ times.

In the next section we present the procedure that computes σ.

4.3. PARTITIONING ALGORITHM

In this section we propose an algorithm that solves the generalized coarsest partition problem we have presented in the preceding section. The operator σ, also introduced in the preceding section, will be the engine of our procedure. The **Partitioning Algorithm** (see Figure 3) takes as input a graph $G = (N, \to)$. A partition pair $\langle \Sigma, P \rangle$, with P transitive, calls the two functions **Refine** (see Figure 4) and **Update** (see Figure 5) until a fix point is reached, and returns the partition pair $\langle S, \preceq \rangle$ that is the solution of the GCPP over G and $\langle \Sigma, P \rangle$. In order to solve the GCPP over G and $\langle \Sigma, P \rangle$, with P not transitive, one first computes P^+. The

```
**Partitioning Algorithm**$((N, \to), \langle \Sigma, P \rangle)$
    change $:= \top$;
    $i := 0$; $\Sigma_0 := \Sigma$; $\mathsf{P}_0 := P$;
    while change do
            change $:= \bot$;
            $\Sigma_{i+1} :=$ **Refine**$(\Sigma_i, \mathsf{P}_i,$ change$)$;
            $\mathsf{P}_{i+1} :=$ **Update**$(\Sigma_i, \mathsf{P}_i, \Sigma_{i+1})$;
            $i := i + 1$;
```

Figure 3. The **Partitioning** algorithm.

```
**Refine**$(\Sigma_i, \mathsf{P}_i,$ change$)$
    $\Sigma_{i+1} := \Sigma_i$;
    for each $\alpha \in \Sigma_{i+1}$ do Stable$(\alpha) := \emptyset$;
    for each $\gamma \in \Sigma_i$ do Row$(\gamma) := \{\gamma' \mid (\gamma, \gamma') \in \mathsf{P}_i)\}$;
    Let Sort be a reverse topological sorting of $\langle \Sigma_i, \mathsf{P}_i \rangle$;
    while Sort $\neq \emptyset$ do
        $\gamma := dequeue($Sort$)$;
        $A := \emptyset$;
        for each $\alpha \in \Sigma_{i+1}, \alpha \to_\exists \gamma$, Stable$(\alpha) \cap$ Row$(\gamma) = \emptyset$ do
            $\alpha_1 := \alpha \cap \to^{-1}(\gamma)$;
            $\alpha_2 := \alpha \setminus \alpha_1$;
            if $\alpha_2 \neq \emptyset$ then change $:= \top$;
            $\Sigma_{i+1} := \Sigma_{i+1} \setminus \{\alpha\}$;
            $A := A \cup \{\alpha_1, \alpha_2\}$;
            Stable$(\alpha_1) :=$ Stable$(\alpha) \cup \{\gamma\}$;
            Stable$(\alpha_2) :=$ Stable$(\alpha)$;
        $\Sigma_{i+1} := \Sigma_{i+1} \cup A$;
        Sort $:=$ Sort $\setminus \{\gamma\}$;
    return $\Sigma_{i+1}$
```

Figure 4. The **Refine** function.

cost of this operation is $O(\Sigma^3)$, and hence it does not affect the global cost of the algorithm. The function **Refine** takes as input a partition pair $\langle \Sigma_i, P_i \rangle$, and it returns the partition Σ_{i+1} that is the coarsest that satisfies the condition in (1σ) of Definition 4.11. The function **Update** takes as input a partition pair $\langle \Sigma_i, P_i \rangle$ and the refinement Σ_{i+1}, and it produces the acyclic and reflexive relation over Σ_{i+1} that is the greatest satisfying the conditions in (2σ) of Definition 4.11. The quotient structures defined below will be used.

DEFINITION 4.17 (Quotient Structures). Let $G = (N, \to)$ and Π be a partition of N. Let $\to_\exists$ and $\to_\forall$ be the relations in Definition 4.3.

```
Update($\Sigma_i$, $\mathsf{P}_i$, $\Sigma_{i+1}$)
    $\mathsf{Ind}_{i+1} := \{(\alpha_1, \beta_1) \mid \alpha_1, \beta_1 \in \Sigma_{i+1}, \alpha_1 \subseteq \alpha, \beta_1 \subseteq \beta(\alpha, \beta) \in \mathsf{P}_i\}$;
    $\Sigma_{i\exists\forall}(\Sigma_{i+1}) := \langle \Sigma_{i+1}, \rightarrow_{\exists}^{\Sigma_i}(\Sigma_{i+1}), \rightarrow_{\forall}^{\Sigma_i}(\Sigma_{i+1})\rangle$;
    $\mathsf{Ref}_{i+1} :=$ New_HHK($\Sigma_{i\exists\forall}(\Sigma_{i+1}), \mathsf{Ind}_{i+1}, \bot$);
    $\Sigma_{(i+1)\exists\forall} := \langle \Sigma_{i+1}, \rightarrow_{\exists}, \rightarrow_{\forall}\rangle$;
    $\mathsf{P}_{i+1} :=$ New_HHK($\Sigma_{(i+1)\exists\forall}, \mathsf{Ref}_{i+1}, \top$);
    return $\mathsf{P}_{i+1}$
```

Figure 5. The **Update** function.

The $\exists$-*quotient structure* over Π is the graph $\Pi_{\exists} = (\Pi, \rightarrow_{\exists})$.
The $\forall$-*quotient structure* over Π is the graph $\Pi_{\forall} = (\Pi, \rightarrow_{\forall})$.
The $\exists\forall$-*quotient structure* over Π is the structure $\Pi_{\exists\forall} = (\Pi, \rightarrow_{\exists}, \rightarrow_{\forall})$.

DEFINITION 4.18 (Induced Structure). Let $G = (N, \rightarrow)$. Let Σ and Π be two partitions of N with Π finer than Σ. The $\exists\forall$-*induced quotient structure* over Π is the structure $\Sigma_{\exists\forall}(\Pi) = (\Pi, \rightarrow_{\exists}^{\Sigma}(\Pi), \rightarrow_{\forall}^{\Sigma}(\Pi)))$, where

$$\begin{aligned} \alpha \rightarrow_{\exists}^{\Sigma} (\Pi)\gamma \quad &\text{iff} \quad \alpha \cap \rightarrow^{-1}(\gamma) \neq \emptyset, \\ \alpha \rightarrow_{\forall}^{\Sigma} (\Pi)\gamma \quad &\text{iff} \quad \alpha \rightarrow_{\exists}^{\Sigma} (\Pi)\gamma \wedge \alpha \subseteq \rightarrow^{-1}(\gamma') \wedge \gamma \subseteq \gamma' \in \Sigma \end{aligned}$$

with $\alpha, \beta \in \Pi$.

At the end of each iteration of the while-loop in the **Partitioning Algorithm** we have that $\langle \Sigma_{i+1}, P_{i+1}\rangle = \sigma(\langle \Sigma_i, P_i\rangle)$.

We immediately see that the **Refine** function works exactly as described in the proof of Theorem 4.13, and hence it produces the partition that is the first element of $\sigma(\Sigma_i, \mathsf{P}_i)$.

COROLLARY 4.19. *If* $\sigma(\langle \Sigma_i, \mathsf{P}_i\rangle) = \langle \Pi, Q\rangle$, *then* $\Pi = \Sigma_{i+1}$.

Update removes pairs from $\mathsf{P}_i(\Sigma_{i+1}) = \mathsf{Ind}_{i+1}$ in order to obtain the relation P_{i+1} that satisfies the two conditions in (2σ) of Definition 4.11. We start with the relation Ind_{i+1}, which is simply the relation induced on Σ_{i+1} by P_i, and the $\exists\forall$-induced structure on Σ_{i+1}. We obtain Ref_{i+1} by removing from Ind_{i+1} all the pairs (α, β) such that there exists $\gamma \in \Sigma_i$ such that $\alpha \rightarrow_{\forall} \gamma$ and there does not exists $\gamma' \in \Sigma_i$ such that $\beta \rightarrow_{\exists} \gamma'$. Hence, Ref_{i+1} satisfies condition (b), but not necessarily condition (c), of (2σ). In order to guarantee also condition (c), we build the quotient structure $\Sigma_{(i+1)\exists\forall}$ and we remove all pairs not satisfying condition (c) from Ref_{i+1}. Hence, P_{i+1} satisfies (2σ).

The deletion of "wrong" pairs is performed by **New_HHK** (see Figure 6), which is a version of [30] adapted to our purposes here.

Notice that the space complexity of the calls to the adapted version of [30] remains limited because they are made on quotient structures. This function is based on the use of the two structures $\Sigma_{i\exists\forall}(\Sigma_{i+1})$ (see Definition 4.18) and $\Sigma_{i+1\exists\forall}$

```
New_HHK($(T, R_1, R_2), K, U$)
    $P := K$;
    for each $c \in T$ do
        $sim(c) := \{e \mid (c, e) \in K\}$;
        $rem(c) := T \setminus pre_1(sim(c))$;
    while $\{c \mid rem(c) \neq \emptyset\} \neq \emptyset$ do
        let $c \in \{c \mid rem(c) \neq \emptyset\}$
        while $rem(c) \neq \emptyset$ do
            let $b \in rem(c)$
            $rem(c) := rem(c) \setminus \{b\}$
            for each $a \in pre_2(c)$ do
                if $b \in sim(a)$ then
                  $sim(a) := sim(a) \setminus \{b\}$;
                  $P := P \setminus \{(a, b)\}$;
                  if $U$ then for each $b_1 \in pre_1(b)$ do
                      if $post_1(b_1) \cap sim(a) = \emptyset$ then
                          $rem(a) := rem(a) \cup \{b_1\}$;
    return $P$
```

Figure 6. The **New_HHK** function.

(see Definition 4.17), and on the following equivalent formulation of the second condition in (2σ).

PROPOSITION 4.20. *Let $G = (N, \rightarrow)$, $\langle \Sigma, P \rangle$ be a partition pair and Π be a partition finer than Σ. Q satisfies (2σ) of Definition 4.11 if and only if Q is the maximal relation over Π such that $Q \subseteq P(\Pi)$ and if $(\alpha, \beta) \in Q$, then*

$$\forall \gamma \in \Pi(\alpha \rightarrow_{\forall}^{\Sigma} (\Pi)\gamma \Rightarrow \exists \gamma' \in \Pi((\gamma, \gamma') \in P(\Pi) \wedge \beta \rightarrow_{\exists}^{\Sigma} (\Pi)\gamma')),$$
$$\forall \gamma \in \Pi(\alpha \rightarrow_{\forall} \gamma \Rightarrow \exists \gamma' \in \Pi((\gamma, \gamma') \in Q \wedge \beta \rightarrow_{\exists} \gamma')),$$

where $\rightarrow_{\forall}^{\Sigma}(\Pi)$ and $\rightarrow_{\exists}^{\Sigma}(\Pi)$ are the edges of the $\exists\forall$-induced quotient structure, while $\rightarrow_{\exists}$ and $\rightarrow_{\forall}$ are the edges of the $\exists\forall$ quotient structure.

We present the conditions in Proposition 4.20 describing one iteration of the **Partitioning Algorithm** on a simple example.

EXAMPLE 4.21. Consider the graph G and the partition $\Sigma = \{\alpha, \beta\}$ shown on the left in Figure 7.

The partition Σ_1 we obtain by applying **Refine**$(\Sigma, I, \top)$ is the partition $\{\alpha_1, \alpha_2, \beta_1, \beta_2\}$ shown on the right in Figure 7.

The relation Ind_1 is formed by the pairs (α_i, α_j), (β_i, β_j), with $i, j \in \{1, 2\}$.

The pair (β_2, β_1) does not satisfies condition (b) in (2σ). In fact, $\beta_2 \rightarrow_{\forall} \alpha$, while β_1 does not reach existentially any class of Σ. With the conditions in Proposition 4.20 this corresponds to the fact that $\beta_2 \rightarrow_{\forall}^{\Sigma} (\Sigma_1)\alpha_2$, while β_1 does not reach through $\rightarrow_{\exists}^{\Sigma}(\Sigma_1)$ any class.

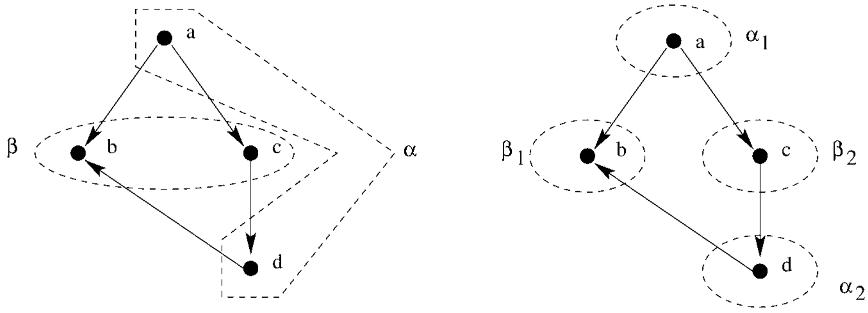


Figure 7. The conditions in (2σ) on the quotient structures.

By computing **New_HHK**$(\Sigma_{0\exists\forall}(\Sigma_1), \mathsf{Ind}_1, \bot)$ we obtain

$$\mathsf{Ref}_1 = \{(\alpha_1, \alpha_2), (\alpha_2, \alpha_1), (\beta_1, \beta_2)\} \cup I.$$

The pair (α_2, α_1) does not satisfies condition (c) in (2σ). In fact, $\alpha_2 \to_\exists \beta_2$, while α_1 does not reach existentially any class δ such that $(\beta_2, \delta) \in \mathsf{Ref}_1$. This is equivalent to what is stated in the second condition of Proposition 4.20.

By computing **New_HHK**$(\Sigma_{1\exists\forall}, \mathsf{Ref}_1, \top)$ we obtain

$$\mathsf{P}_1 = \{(\alpha_1, \alpha_2), (\beta_1, \beta_2)\} \cup I,$$

that is, the maximal relation satisfying the conditions in Proposition 4.20.

The computation performed by the function **Update** corresponds to determining the largest relation included in P_i and satisfying conditions (2σ), thereby getting us closer to stability. The correctness of **Update** is based on some technical lemmas reported in [27]. Hence we can prove the following invariant relative to the functions **Update** and **Refine**.

THEOREM 4.22 (Refine-Update Invariant). *The following holds:*

$$\langle \Sigma_{i+1}, \mathsf{P}_{i+1} \rangle = \sigma(\langle \Sigma_i, \mathsf{P}_i \rangle).$$

As a consequence of Theorem 4.15 and of Corollary 4.16, since the **Partitioning Algorithm** terminates whenever $\Sigma_{i+1} = \Sigma_i$, we can conclude that the **Partitioning Algorithm** computes the solution $\langle S, \preceq \rangle$ of the GCPP over G and $\langle \Sigma, P \rangle$, with P transitive, performing at most $|S|$ iterations of the while-loop.

THEOREM 4.23 (Time Complexity). *Given a graph $G = (N, \to)$ and a partition pair $\langle \Sigma, P \rangle$, with P transitive, the algorithm* **Partitioning Algorithm** *computes the solution $\langle S, \preceq \rangle$ of the GCPP over them in time* $O(|S|^2|\to|)$.

Proof. From Corollary 4.16 we have that at most $|S|$ iterations of the while-loop are performed. We now prove that, in each iteration of the **Partitioning Algorithm**, the **Refine** function takes $O(|\Sigma_i||\to|) = O(|S||\to|)$ time. The cost of the refining steps over the entire **Partitioning Algorithm** is then $O(|S|^2|\to|)$.

The initialization phase (the instructions before the while-loop) in **Refine** takes time $O(|\Sigma_i|^2)$. During each iteration of the while-loop an element γ is taken out of Sort, and it is never reinserted in it. As there are $|\Sigma_i|$ classes in Sort, the while-loop is iterated at most $|\Sigma_i|$ times. Once γ has been dequeued, the set $\mathsf{Split}(\gamma) = \{\alpha \mid \to^{-1}(\gamma) \cap \alpha \neq \emptyset\}$ can be computed in $O(|{\to^{-1}(\gamma)}|)$ time. Each instruction in the while-loop, different from the for-loop, costs $O(1)$; without considering the innermost for-loop, the global cost of the while-loop, in a refining step, is then $O(\to^{-1}(\gamma_1) + \cdots + \to^{-1}(\gamma_{|\Sigma_i|})) + O(|\Sigma_i|) = O(|{\to}|)$.

$\mathsf{Split}(\gamma)$ contains at most $|{\to^{-1}(\gamma)}|$ elements, and hence the number of for-loop iterations is bounded by $|{\to^{-1}(\gamma)}|$. Assuming P_i is represented as an $\Sigma_i \times \Sigma_i$ adjacency matrix, the check $\mathsf{Stable}(\alpha) \cap \mathsf{Row}(\gamma) = \emptyset$ can be implemented in $O(|\mathsf{Stable}(\alpha)|) = O(|\Sigma_i|)$. As far as the remaining operations in the for-loop are concerned, we observe that, for each class $\alpha \in \Sigma_{i+1}$ with $\to^{-1}(\gamma) \cap \alpha \neq \emptyset$, the sets $\alpha_1 = \to^{-1}(\gamma) \cap \alpha$ and $\alpha_2 = \alpha \setminus \alpha_1$ can be provided while computing (i.e., at the cost of computing) $\mathsf{Split}(\gamma)$: strategies similar to the ones suggested in [39] can be used to this purpose. The For-loop instructions involving the updating of Σ_{i+1} and the setting of the Stable sets (relative to the new Σ_{i+1} classes) can be implemented in $O(1)$. Thus the global cost of the for-loop in a refining step turns out to be $O(\to^{-1}(\gamma_1)|\Sigma_i| + \cdots + \to^{-1}(\gamma_{|\Sigma_i|})|\Sigma_i|) = O(|{\to}||\Sigma_i|)$. We have that the complexity of the **Refine** function is $O(|S|^2 + |S||{\to}|) = O(|S||{\to}|)$.[*]

In **Update** the cost of the initialization of $\mathsf{Ind}_{\mathsf{i+1}}$ is $O(\Sigma_{i+1}^2)$. As far as the initialization of the $\exists\forall$-quotient structure and the $\exists\forall$-induced quotient structure are concerned, the following procedure can be used to build the structures $\Sigma_{(i+1)\exists\forall} = \langle \Sigma_{i+1}, \to_\exists, \to_\forall \rangle$ and $\Sigma_{i\exists\forall}(\Sigma_{i+1}) = \langle \Sigma_{i+1}, \to_\exists^{\Sigma_i}(\Sigma_{i+1}), \to_\exists^{\Sigma_i}(\Sigma_{i+1}) \rangle$ in $O(|\Sigma_{i+1}|^2 + |{\to}|)$ time (each iteration of the innermost for-loop can be easily implemented at the cost of computing $\to^{-1}(\alpha')$):

```
1. for each $\alpha \in \Sigma_i$ do
2.     compute $\to^{-1}(\alpha)$ and sign each $\beta' \in \Sigma_{i+1}, \beta' \subseteq \to^{-1}(\alpha)$;
3.     for each $\alpha' \in \Sigma_{i+1}, \alpha' \subseteq \alpha$ do
4.         $\to_\exists := \{\langle \alpha'\beta' \rangle \mid \beta' \in \Sigma_{i+1}, \beta' \cap \to^{-1}(\alpha') \neq \emptyset\}$;
5.         $\to_\exists^{\Sigma_i}(\Sigma_{i+1}) := \{\langle \alpha'\beta' \rangle \mid \beta' \in \Sigma_{i+1}, \beta' \cap \to^{-1}(\alpha') \neq \emptyset\}$;
6.         $\to_\forall := \{\langle \alpha'\beta' \rangle \mid \beta' \in \Sigma_{i+1}, \beta' \subseteq \to^{-1}(\alpha')\}$;
7.         $\to_\exists := \{\langle \alpha'\beta' \rangle \mid \beta' \in \Sigma_i + 1$ has been signed in step 2,
                   $\beta' \cap \to^{-1}(\alpha') \neq \emptyset\}$.
```

Without considering the two calls to the **New_HHK** function, the complexity of the updating steps overall the **Partitioning Algorithm** is then $O(|S|(|{\to}| + |S|^2)) = O(|S|^2|{\to}|)$. To evaluate the cost of performing the entire set of calls to the function **New_HHK**, we can either directly apply the results in [30] to a single procedure's call or use a global argument. We could apply directly the results in [30] because both the calls to **New_HHK**, relative to a single **Update** step, correspond to a call

[*] We assume that $|N| = O(|{\to}|)$ in the graph in input: note that in the context of model checking, Kripke structures modeling the finite systems to validate always satisfy the above assumptions.

to the function in [30] on a graph having Σ_i nodes linked by two types of edges: $\forall$-edges and $\exists$-edges (moreover, the first call is stopped after only one iteration). The only substantial difference between the function in [30] and **New_HHK** is that some of the operations of the predecessor's set's computation are specialized for the $\forall$-edges. As two nodes are linked by a $\forall$-edge only if they are linked by an $\exists$-edge and there are $O(|{\to}|)$ $\exists$-edges, we can conclude, using the results in [30], that each call to **New_HHK** costs $O(|S||{\to}|)$.

However, the results in [30] are based on the use of a special counter table requiring $O(|S|^2 \log(|S|))$ space. This counter table allows one to check the innermost if guard in **New_HHK** in constant time (it keeps track, for each couple of classes $\langle\alpha, \beta\rangle$, of the value $|post(\alpha) \cap\ sim(\beta)|$). Using a global argument to evaluate the cost of the entire set of calls to the function **New_HHK**, we can obtain the same time complexity avoiding the need to maintain the counter tables. In particular, the innermost if statement of **New_HHK** over all the **Partitioning Algorithm** is $O(|S|^2|{\to}|)$, without any counter table. In fact, the innermost if statement in **New_HHK** is executed only after a class, say β, is removed from the set of classes simulating another class, say α; its cost, without counter tables, is easily proved to be $O(|S||{\to}^{-1}(\beta)|)$. Let α_k be a class in Σ_k and consider, for each iteration of the **Partitioning Algorithm** i, $\alpha_i \supset \alpha_k$ and the classes $\beta_i^1, \dots, \beta_i^{m_i}$ removed from $sim(\alpha_i)$ while executing the innermost for-loop in a **New_HHK** call. The classes $\beta_1^1, \dots, \beta_1^{m_1}, \dots, \beta_k^1, \dots, \beta_k^{m_k}$ are mutually disjoint; hence the cost of executing the innermost if statement of **New_HHK**, involving a class just recognized to be not able to simulate $\alpha_i \supset \alpha_k$, is $(|{\to}^{-1}(\beta_1^1)| + \cdots + |{\to}^{-1}(\beta_1^{m_1})| + \cdots + |{\to}^{-1}(\beta_k^1)| + \cdots + |{\to}^{-1}(\beta_k^{m_k})|)|S| = O(|{\to}||S|)$. As there are $|S|$ classes in Σ_k, the innermost if statement of the function **New_HHK** takes, over the entire **Partitioning Algorithm**, $O(|S|^2|{\to}|)$. □

THEOREM 4.24 (Space Complexity). *Given a graph $G = (N, \to)$ and a partition pair $\langle\Sigma, P\rangle$, with P transitive, the algorithm* **Partitioning Algorithm** *computes the solution $\langle S, \preceq\rangle$ of the GCPP in space* $O(|S|^2 + |N| \log(|S|))$.

Proof. During each iteration of the algorithm it is necessary to consider the relation $\to$; the relation P_i (at most $O(|\Sigma_i|^2)$ space); and the relation that maps each node in N into the class of Σ_i to which it belongs (space $O(|N| \log |\Sigma_i|)$).

As observed in [30], we need not keep the relation $\to$ in memory: we can use it only when it is necessary to provide the set of successors and predecessors of a given node. Hence the space complexity is $O(|S|^2 + |N| \log(|S|)))$. □

4.4. IMPLEMENTATION AND TESTS

To assess the performance of the **Partitioning Algorithm**, we have implemented it in Standard ML and interfaced it with the Concurrency Workbench of the New Century (CWB-NC) (see [13]). The CWB-NC release incorporates both the simulation algorithm by Paige and Bloom [4] and the simulation algorithm by Cleaveland and Tan [15]. We tested our routine and the latter-mentioned procedure on some toy examples as well as on case studies included in the CWB-NC.

The CWB-NC analysis routines work on transition systems having labeled edges. Thus, we adapted our algorithm following an approach similar to the one used in [15]. The **Refine** step is performed once with respect to each action, while in the **Update** step an action parameter is employed.

In the sequel we describe some of the data structures used giving several implementation details. Then, some experimental results will be presented. The tests have been executed on a Pentium III, 400 MHz PC, 256 MB RAM, OS Linux Red Hat 6.2. Further details on the implementation as well as on the tests are available in [27].

We use two modifiable arrays of records (coarser_partition_table and finer_partition_table) to represent, in each iteration i of the algorithm, the partition Σ_{i-1} (to be refined) and the partition Σ_i (result of the refinement step). Each record in the finer_partition_table corresponds to a block of Σ_i, and we associate to it the following fields:

- states: a doubly linked list of states representing the set of states belonging to the block;
- touched_states: a doubly linked list of states used to keep trace of the states touched while scanning the elements having transitions into a Σ_{i-1} class;
- superclass: the index in the coarser_partition_table of the Σ_{i-1} class containing the block;
- stable_blocks: a list of indexes in the coarser_partition_table allowing to represent $\mathsf{Stable}(\alpha')$ within the **Refine** step.

Each record in the coarser_partition_table corresponds to a block of Σ_{i-1}, and we keep the following information associated to it:

- splitted_blocks: a list of indexes in the finer_partition_table corresponding to the blocks $\alpha' \in \Sigma_i$ such that $\alpha' \subseteq \alpha$;
- greater_blocks: a list of indexes in the coarser_partition_table corresponding to the blocks $\beta \in \Sigma_{i-1}$ such that $(\alpha, \beta) \in P_{i-1}$.

We do not represent explicitly the set of states of each class $\alpha \in \Sigma_{i-1}$: this set can be retrieved by combining the doubly linked list of states of each Σ_i class contained in α. The states are integers ranging over $[1 \ldots \mathsf{num\_states}]$. Thus, they are used to index the table states_info maintaining, for each state, a pointer to the position in the unique doubly linked list (states or touched_states) they belong to. The relation $\rightarrow^{-1}$ is maintained by means of an adjacency list. This allows one to retrieve, for each node a, the set $\rightarrow^{-1}(a)$ in time proportional to its size.

Figure 8 shows the structures of two examples used in the tests. In these two cases the **Partitioning Algorithm** takes advantage of the fact that its time and space complexities depend on the size of the simulation quotient. In both examples

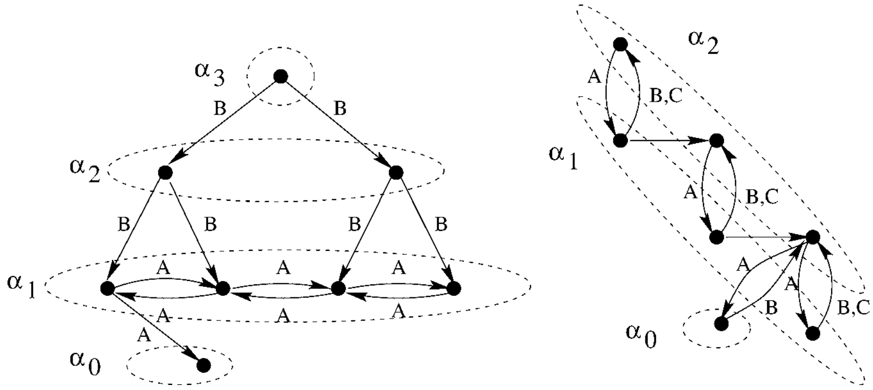


Figure 8. The Tree and the 3-classes examples.

Table I. Results of the tests on Tree, 3-classes, and a variant of Tree

Tree (Figure 8 left)			3-classes (Figure 8 right)			A variant of Tree		
$\|N\|$	**CT**	**PA**	States	**CT**	**PA**	States	**CT**	**PA**
256	0.54	0.13	500	0.14	0.01	256	0.06	0.13
512	2.19	0.28	1000	12.56	0.08	512	0.16	0.3
1024	8.41	0.61	1500	27.65	0.25	1024	0.34	0.71
2048	–	1.42	2000	–	0.32	2048	1.16	1.54
8192	–	8.53	10000	–	2.18	8192	5.98	9.01

all the states are not bisimilar. In the example on the left of Figure 8, which we call Tree, each level of the tree corresponds to a simulation class, and the block α_0 is simulated by all the other blocks. Hence, the size of the simulation quotient is logarithmic with respect to the state space. The example on the right of Figure 8, which we call 3-classes, has three classes of simulation, α_0, α_1, and α_2, and α_0 is simulated by α_1. Thus, the size of the simulation quotient of 3-classes is three, while the size of the bisimulation quotient is equal to the size of the state space.

Table I shows the results of running the two simulation procedures over graphs having the structure of Tree and 3-classes with increasing state space sizes. In the tables we use **CT** to denote the algorithm by Cleveland and Tan and **PA** to denote our **Partitioning Algorithm**. We express the times in seconds; we use "–" in the cases in which we run out of memory. As expected, since in both examples the simulation reduction is larger than the bisimulation one, the **Partitioning Algorithm** uses less time and space than does the procedure by Cleaveland and Tan.

Notice that if in the Tree example we remove the node in the class α_0, we have that the bisimulation and the simulation quotients coincide and they are given by the levels of the tree. In Table I we show also the results of the tests performed

Table II. Tests on four bit-alternating-protocol models included in the CWB-NC

	$\lvert N\rvert$	$\lvert\to\rvert$	$\lvert B\rvert$	$\lvert S\rvert$	**CT**	**PA**
ABP-lossy	57	130	15	14	0.05	0.04
ABP-safe	49	74	18	17	0.04	0.03
Two-link-netw	1589	6819	197	147	7.53	5.16
Three-link-netw	44431	280456	2745	1470	–	1336.24

on this variant of the Tree. In this case the Cleaveland and Tan algorithm takes advantage of the large (maximal) bisimulation reduction and performs better than our algorithm.

In Table II we report the results of tests performed using a benchmark taken from CWB-NC. In particular, Table II shows information about the structures of different models of the alternating-bit protocol included in the CWB-NC release, as well as the times resulting from minimizing the systems. This last example shows the space efficiency of the **Partitioning Algorithm** on a more concrete example.

5. Conclusions

In this paper we discussed the notions of bisimulation and simulation, and their algorithmic counterpart as graph reduction procedures, with special emphasis on their use in verification. The search for efficient procedures determining the bisimulation reduction is shown to be solvable moving to an equivalent coarsest-partition problem first, which is then observed to be simply an equality problem on nonwell-founded sets. As for simulation, the coarsest-partition approach is also shown to be useful in the study and design of fast simulation algorithms and heuristics especially developed to operate in situations in which strong space constraints are present. The use of the ∀∃-structure and the definition of a coarsest partition problems on them seem to be a methodology with some potential in those situations in which a fix point in the lattice of equivalence relations is to be computed.

We plan to work on a symbolic version of our algorithms, which is naturally suggested by the fact that the discussed procedure works on sets of nodes (the classes of the partitions). Finally, an attempt to combine the negative and positive strategies to solve the coarsest-partition problem for simulation presented here (as, in the case of bisimulation, has been done in [19, 20]) is also under study.

Acknowledgments

We thank Eugenio Omodeo, Rance Cleaveland, Li Tan, and the anonymous referees for their useful suggestions.

References

1. Aczel, P.: *Non-Well-Founded Sets*, CSLI Lecture Notes 14, Stanford University Press, 1988.
2. Bloem, R., Gabow, H. N. and Somenzi, F.: An algorithm for strongly connected component analysis in $n \log n$ symbolic steps, in W. A. Hunt, Jr. and S. D. Johnson (eds.), *Proceedings of International Conference on Formal Methods in Computer-Aided Design (FMCAD'00)*, Lecture Notes in Comput. Sci. 1954, Springer, 2000, pp. 37–54.
3. Bloom, B.: Ready simulation, bisimulation, and the semantics of CCS-like languages, Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, August 1989.
4. Bloom, B. and Paige, R.: Transformational design and implementation of a new efficient solution to the ready simulation problem, *Science of Computer Programming* **24**(3) (June 1995), 189–220.
5. Bouajjani, A., Fernandez, J. C. and Halbwachs, N.: Minimal model generation, in E. Clarke and R. Kurshan (eds.), *Proceedings of International Conference on Computer Aided Verification (CAV'90)*, Lecture Notes in Comput. Sci. 531, Springer, 1990, pp. 197–203.
6. Bouali, A.: XEVE, an ESTEREL verification environment, in A. J. Hu and M. Y. Vardi (eds.), *Proceedings of International Conference on Computer Aided Verification (CAV'98)*, Lecture Notes in Comput. Sci. 1427, Springer, 1998, pp. 500–504.
7. Bouali, A. and de Simone, R.: Symbolic bisimulation minimization, in G. von Bochmann and D. K. Probst (eds.), *Proceedings of International Conference on Computer Aided Verification (CAV'92)*, Lecture Notes in Comput. Sci. 663, Springer, 1992, pp. 96–108.
8. Bryant, R. E.: Symbolic manipulation of Boolean functions using a graphical representation, in *Proceedings of Design Automation Conference (DAC'85)*, 1985.
9. Bustan, D. and Grumberg, O.: Simulation based minimization, in D. A. McAllester (ed.), *Proceedings of International Conference on Automated Deduction (CADE'00)*, Lecture Notes in Comput. Sci. 1831, Springer, 2000, pp. 255–270.
10. Clarke, E. M. and Emerson, E. A.: Design and synthesis of synchronization skeletons using brancing time temporal logic, in *Proceedings of Workshop on Logic of Programs*, Lecture Notes in Comput. Sci. 131, Springer, 1982, pp. 52–71.
11. Clarke, E. M., Grumberg, O. and Peled, D. A.: *Model Checking*, MIT Press, 1999.
12. Cleaveland, R., Parrow, J. and Steffen, B.: The concurrency workbench: A semantics based tool for the verification of concurrent systems, *ACM Transactions on Programming Languages and Systems* **15**(1) (January 1993), 36–72.
13. Cleaveland, R. and Sims, S.: The NCSU concurrency workbench, in R. Alur and T. A. Henzinger (eds.), *Proceedings of International Conference on Computer Aided Verification (CAV'96)*, Lecture Notes in Comput. Sci. 1102, Springer, 1996, pp. 394–397.
14. Cleaveland, R. and Steffen, B.: A linear-time model-checking algorithm for the alternation-free modal mu-calculus, in K. G. Larsen and A. Skou (eds.), *Proceedings of International Conference on Computer Aided Verification (CAV'91)*, Lecture Notes in Comput. Sci. 575, Springer, 1992, pp. 48–58.
15. Cleaveland, R. and Tan, L.: Simulation revised, in T. Margaria and W. Yi (eds.), *Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, Lecture Notes in Comput. Sci. 2031, Springer, 2001, pp. 480–495.
16. Dams, D., Gerth, R. and Grumberg, O.: Generation of reduced models for checking fragments of CTL, in C. Courcoubetis (ed.), *Proceedings of International Conference on Computer Aided Verification (CAV'93)*, Lecture Notes in Comput. Sci. 697, Springer, 1993, pp. 479–490.
17. Dams, D., Gerth, R. and Grumberg, O.: Abstract interpretation of reactive systems, *ACM Transactions on Programming Languages and Systems* **19**(2) (March 1997), 253–291.
18. Dovier, A., Gentilini, R., Piazza, C. and Policriti, A.: Rank-based symbolic bisimulation (and model checking), in Ruy J. Guerra B. de Queiroz (ed.), *Proceedings of Workshop on Language,*

Logic, Information, and Computation (Wollic'02), ENTCS 67, Elsevier Science, 2002, pp. 167–184.

19. Dovier, A., Piazza, C. and Policriti, A.: A fast bisimulation algorithm, in G. Berry, H. Comon and A. Finkel (eds.), *Proceedings of International Conference on Computer Aided Verification (CAV'01)*, Lecture Notes in Comput. Sci. 2102, Springer, 2001, pp. 79–90.
20. Dovier, A., Piazza, C. and Policriti, A.: A fast bisimulation algorithm, *J. Theoret. Comput. Sci.* (2003). Accepted for publication, to appear.
21. Fernandez, J. C., Garavel, H., Kerbrat, A., Mateescu, R., Mounier, L. and Sighireanu, M.: CADP: A protocol validation and verification toolbox, in R. Alur and T. A. Henzinger (eds.), *Proceedings of International Conference on Computer Aided Verification (CAV'96)*, Lecture Notes in Comput. Sci. 1102, Springer, 1996, pp. 437–440.
22. Fisler, K. and Vardi, M. Y.: Bisimulation and model checking, in L. Pierre and T. Kropf (eds.), *Proceedings of Correct Hardware Design and Verification Methods (CHARME'99)*, Lecture Notes in Comput. Sci. 1703, Springer, 1999, pp. 338–341.
23. Focardi, R. and Gorrieri, R.: The compositional security checker: A tool for the verification of information flow security properties, *IEEE Transaction on Software Engineering* **23**(9) (1997), 550–571.
24. Forti, M. and Honsell, F.: Set theory with free construction principles, *Ann. Scuola Norm. Sup. Pisa Cl. Sc.* **IV**(10) (1983), 493–522.
25. Gentilini, R., Piazza, C. and Policriti, A.: Simulation as coarsest partition problem, in J. P. Katoen and P. Stevens (eds.), *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, Lecture Notes in Comput. Sci. 2280, Springer, 2002, pp. 415–430.
26. Gentilini, R., Piazza, C. and Policriti, A.: Simulation reduction as constraint, in M. Comini and M. Falaschi (eds.), *Proceedings of Workshop on Functional and Constraint Logic Programming (WFLP'02)*, ENTCS 76, Elsevier Science, 2002.
27. Gentilini, R., Piazza, C. and Policriti, A.: From bisimulation to simulation: Coarsest partition problems, RR 12-2003, Dep. of Computer Science, University of Udine, Italy, 2003.
28. Gries, D.: Describing an algorithm by Hopcroft, *Acta Inform.* **2** (1973), 97–109.
29. Grumberg, O. and Long, D. E.: Model checking and modular verification, *ACM Transactions on Programming Languages and Systems* **16**(3) (May 1994), 843–871.
30. Henzinger, M. R., Henzinger, T. A. and Kopke, P. W.: Computing simulations on finite and infinite graphs, in *Proceedings of Symposium on Foundations of Computer Science (FOCS'95)*, IEEE Computer Society Press, 1995, pp. 453–462.
31. Hopcroft, J. E.: An $n \log n$ algorithm for minimizing states in a finite automaton, in Z. Kohavi and A. Paz (eds.), *Theory of Machines and Computations,* Academic Press, 1971, pp. 189–196.
32. Kanellakis, C. and Smolka, S. A.: CCS expressions, finite state processes, and three problems of equivalence, in *ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, 1983, pp. 228–240.
33. Kannellakis, P. C. and Smolka, S. A.: CCS expressions, finite state processes, and three problems of equivalence, *Inform. and Comput.* **86**(1) (1990), 43–68.
34. Knuutila, T.: Re-describing an algorithm by Hopcroft, *Theoret. Comput. Sci.* **250** (2001), 333–363.
35. Lee, D. and Yannakakis, M.: Online minimization of transition systems, in *Proceedings of ACM Symposium on Theory of Computing (STOC'92)*, ACM Press, 1992, pp. 264–274.
36. McMillan, K. L.: *Symbolic Model Checking: An Approach to the State Explosion Problem*, Kluwer Academic Publishers, 1993.
37. Milner, R.: An algebraic definition of simulation between programs, in *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI'71)*, Morgan Kaufmann, 1971, pp. 481–489.

38. Milner, R.: A calculus of communicating systems, in G. Goos and J. Hartmanis (eds.), Lecture Notes in Comput. Sci. 92, Springer, 1980.

39. Paige, R. and Tarjan, R. E.: Three partition refinement algorithms, *SIAM J. Comput.* **16**(6) (1987), 973–989.

40. Paige, R., Tarjan, R. E. and Bonic, R.: A linear time solution to the single function coarsest partition problem, *Theoret. Comput. Sci.* **40**(1) (1985), 67–84.

41. Park, D.: Concurrency and automata on infinite sequences, in P. Deussen (ed.), *Proceedings of International Conference on Theorical Computer Science (TCS'81)*, Lecture Notes in Comput. Sci. 104, Springer, 1981, pp. 167–183.

42. Piazza, C.: *Computing in Non Standard Set Theories*, Ph.D. thesis, Department of Computer Science, University of Udine, 2002. Electronically available from `http://www.dimi.uniud.it/~piazza`.

43. Piazza, C. and Policriti, A.: Ackermann encoding, bisimulations, and OBDD's, in M. Leuschel, A. Podelski, C. R. Ramakrishnan and U. Ultes-Nitsche (eds.), *Proceedings of International Workshop on Verification and Computational Logic (VCL'01)*, Southampton University Technical Report DSSE-TR-2001-3, ACM Digital Library, 2001, pp. 43–53.

44. Roscoe, W. R.: *A Classical Mind: Essays in Honour of C. A. R. Hoare*, Prentice Hall, 1994, Chapter 'Model Checking CSP'.

45. van Benthem, J.: Modal correspondence theory, Ph.D. thesis, Universiteit van Amsterdam, Instituut voor Logica en Grondslagenonderzoek van Exacte Wetenschappen, 1976.