Rank and simulation: the well-founded case

Published
DOI:10.1093/logcom/ext066

Rank and Simulation: the well-founded case

Raffaella Gentilini[1], Carla Piazza[2], and Alberto Policriti[2]

[1] Dip. di Matematica e Informatica, Università degli Studi di Perugia,
Via Vanvitelli 1, Perugia (IT)
[2] Dip. di Matematica e Informatica, Università degli Studi di Udine,
Via Le Scienze 206, Udine (IT).
{raffaella.gentilini}@dmi.unipg.it, {piazza|policriti}@dimi.uniud.it

Abstract. We consider the algorithmic problem of computing the maximal simulation preorder (and quotient) on acyclic labelled graphs. The acyclicity allows to exploit an inner structure on the set of nodes, that can be processed in stages according to a set-theoretic notion of rank. This idea, previously used for bisimulation computation, on the one hand improves on the performances of the ensuing procedure and, on the other hand, gives to the solution an orderly iterative flavour making the algorithmic idea more explicit. The computational complexity achieved is good as we obtain the best performing algorithm for simulation computation on acyclic graphs, in both time and space.

This is a pre-copyedited, author-produced PDF of an article accepted for publication in Journal of Logic and Computation following peer review. The version of record – Raffaella Gentilini, Alberto Policriti, and Carla Piazza *Rank and simulation: the well-founded case* Journal of Logic and Computation(2015) 25(6):1331-1349, first published on line December 3 2013– is available online at:
`http://m.logcom.oxfordjournals.org/content/25/6/1331`.

1 Introduction

The simulation preorder [29] is a behavioural refinement relation on labelled graphs, suitable to characterize situations in which some form of collapse among states is possible without a significant loss in expressive power. For this reason simulation is widely used as a formal tool supporting the design and the automated reasoning on complex systems. In particular, simulations play a role in two tasks that are often crucial to guarantee the success of a formal method in system design or in computer aided verification: system *refinement* and system *abstraction* [25]. In this context, the behaviour of a system or a set of programs implementing a collection of cooperating units, is naturally modelled as a (labelled) graph, whose nodes describe the possible states and arrows represent actions. Given one such specification of a system as a labelled graph G_1, the simulation preorder provides a formal tool for checking whether G_1 is *correctly* implemented (or refined) by the concrete system G_2. Moreover, the equivalence relation induced by simulation can be used as an abstraction tool to cope with the intricacies buried in the modelling activity and to control the sheer size of the obtained structures. In particular, space requirements underlay the notorious state-explosion problem

in *model checking* [11], a fully automatic (and quite efficient in time) formal method for verifying finite-state systems[1] with respect to temporal logics specifications. In this context, the tool of abstraction/refinement is also crucial to verify by means of model checking *infinite state systems*, such as e.g. hybrid automata [25, 22, 4].

Simulation and Logic Abstraction methods for model checking are required to be preservative with respect to the logic language used for specifying the properties of the system. An abstraction method is said to be *weakly preservative* for a temporal logic $\mathcal{L}$ if whenever a property p of $\mathcal{L}$ is true in the abstract structure, p holds also in the concrete model. An abstraction method is said to be *strongly preservative* for a temporal logic $\mathcal{L}$ if both true and false $\mathcal{L}$-properties are preserved from the abstract structure to the concrete model. Grumberg et al. [31] proved that the simulation preorder is weakly preservative for ACTL* and ACTL, the universal fragments of the branching temporal logics CTL and CTL* [10], as well as for the universal fragment of the μ-calculus. In [28], it was shown that the simulation equivalence strongly preserves both the universal and the existential fragment of the μ-calculus. As a consequence, it strongly preserves its sublogics ACTL*, ECTL*, ECTL and ACTL, widely used for model checking. The latter preservation results combined with the existence of a number of polynomial algorithms for computing (the maximal) simulation on a labelled graph [25, 6, 20, 36], explains the appealing of simulation-based abstraction methods in model checking, also w.r.t. other popular behavioural refinement relations such as language equivalence and bisimulation [34]. In fact, language equivalence provides strong preservation of linear temporal properties and large reductions, however its complexity is exponential, whereas the complexity of bisimulation and simulation is polynomial [32, 15, 16]. On the one hand, bisimulation has the advantage (w.r.t. simulation and language equivalence) of preserving more expressive logics. On the other hand, this can also be seen as a disadvantage, since the abstract structure is required to be so close to the original model that the reductions allowed are far less powerful.

Our Contribution In this paper we consider the problem of *simulation* preorder computation and its relationship with a notion of *rank*. The notion of rank we use is defined on the nodes of the underlying graph, a graph that we assume to be *acyclic*. Directed acyclic graphs (DAGs) are commonly used to model data in a wide variety of applications, ranging from business process modelling [13, 5], biological and biomedical ontologies [17], semantic WEB-schemas and XML documents [1]. Abstraction methods based on the notions of bisimulation and simulation have been naturally considered in these contexts to support the automated reasoning on the corresponding massive data-sets. For example, grouping together bisimilar/similar nodes in a XML data-set is the first step in many approaches to the construction of indexing data structures for efficient XPath query evaluation [35, 27, 24].

As far as the complexities are concerned, we propose a simulation algorithm that has optimal performances with respect to both time and space on acyclic graphs, outperforming [36, 37, 21]. More specifically, our algorithm uses $\mathcal{O}(|E||V_{\equiv_S}|)$ time and

[1] The labelled graphs used to model the system under verification are called *Kripke structures* in the context of Model Checking

$O(|V_{\equiv_S}|^2 + |V|\log(|V_{\equiv_S}|))$ bits to compute a simulation preorder on a given acyclic graph G with $|V|$ nodes and $|E|$ edges, where $|V_{\equiv_S}|$ denotes the size of the maximum simulation (equivalence) on G. The time/space improvement with respect to [36, 37, 21] is a direct consequence of a computing strategy proceeding by ranks. At any (refinement-)step only arcs actually contributing to reach stability are traversed and, as a consequence, only a minimal amount of space is allocated. Possible approaches to extend our results to the general cyclic case are trughly discussed, although a satisfactory extension is left as an open problem.

State of the Art Among the algorithms for computing the simulation preorder, the most well known ones are by Henzinger, Henzinger and Kopke [25], Bustan and Grumberg [6], Tan and Cleaveland [12], Gentilini, Piazza, and Policriti [21, 39], and Ranzato and Tapparo [36, 37]. Given a (labelled) graph G with $|V|$ nodes and $|E|$ edges, let $|V_{\equiv_S}|$ be the size of the maximum simulation (equivalence) on G. The algorithm by Ranzato and Tapparo [36] runs in $\mathcal{O}(|E||V_{\equiv_S}|)$ time and $\mathcal{O}(|E||V||V_{\equiv_S}|)$ space. It is the best up-to-date simulation procedure as far as time complexity is concerned. On the other hand, the algorithm in [21] (that originally had a minor flow, subsequently corrected in [39]) has the best up-to-date space complexity—$\mathcal{O}(|V_{\equiv_S}|^2 + |V|\log(|V_{\equiv_S}|))$—and runs in $\mathcal{O}(|E||V_{\equiv_S}|^2)$ time. In [37], Ranzato and Tapparo proposed a new simulation algorithm featuring an improvement w.r.t. the space-complexity of their previous procedure, while slightly worsening the time-performance (of a cubic factor w.r.t. $|V_{\equiv_S}|$).

2 Simulation as Coarsest Partition Pair Problem

We start by introducing basic notations and definitions. In particular, we recall the formal definition of maximum simulation problem and its encoding as a partitioning problem.

Preliminary to that, we introduce the notion of *preorder*, which is "almost" a partial order, as it is not required to be neither symmetric nor anti-symmetric.

Definition 1 (Preorder). *Let V be a set and $Q \subseteq V \times V$ a binary relation over V. Q is said to be a* preorder *over V if and only if Q is reflexive and transitive.*

The simplest structures over which it is reasonable to use preorders are *labelled graphs*, i.e., directed graphs whose nodes are equipped with labels, to be interpreted and represented as equivalence classes.

Definition 2 (Labelled Graph). *A triple $G = \langle V, E, \Sigma\rangle$ is said to be a (finite)* labelled graph *if and only if $G^- = \langle V, E\rangle$ is a (finite) graph and Σ is a partition over V. We say that two nodes $v_1, v_2 \in V$ have the same* label *if they belong to the same Σ-class.*

An equivalent way to define labelled graphs is to use a labelling function $\ell : V \to L$, where L is a finite set of labels (inducing of a partition Σ_L of V). Given a node $v \in V$ we will use $[v]_\Sigma$ (or $[v]$, if Σ is clear from the context) to denote the Σ- class to which v belongs.

Example 1. A *Kripke Structure* is a labelled graph and, vice-versa, each connected labelled 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 3 (Simulation). *Let $G = \langle V, E, \Sigma \rangle$ be a labelled graph. A relation $\leq \subseteq V \times V$ is said to be a* simulation *over G if and only if:*

1. $v \leq u \rightarrow [v]_\Sigma = [u]_\Sigma$;
2. $(v \leq u \wedge vEv_1) \rightarrow \exists u_1 (uEu_1 \wedge v_1 \leq u_1)$.

In this case we also say that u simulates *v.*
We say that u and v are sim-equivalent *($u \equiv_s v$) if there exist two simulations $\leq_1$ and $\leq_2$, such that $u \leq_1 v$ and $u \leq_2 v$.*

Notice that a simulation might be neither reflexive nor transitive: e.g. the *empty* relation is always a simulation. However the reader can easily verify that given an arbitrary simulation its reflexive and transitive closure is always a simulation. A simulation $\preceq_S$ over G is said to be *maximal* if for all the simulations $\leq$ over G it holds $\leq \subseteq \preceq_S$. Given a labelled graph $G = \langle V, E, \Sigma \rangle$ there always exists a unique maximal simulation $\preceq_S$ over G. Moreover $\preceq_S$ is a preorder [29].

Example 2. Consider the labelled graph $G = \langle V, E, \Sigma \rangle$ depicted in Figure 1, where $V = \{x, y, z\}$, $E = \{(x, y), (x, z), (y, z)\}$, and $\Sigma = \{\alpha = \{x, y\}, \beta = \{z\}\}$. The maximum simulation preorder on G is given by $I \cup \{(y, x)\}$, where I denotes the identity relation over V.

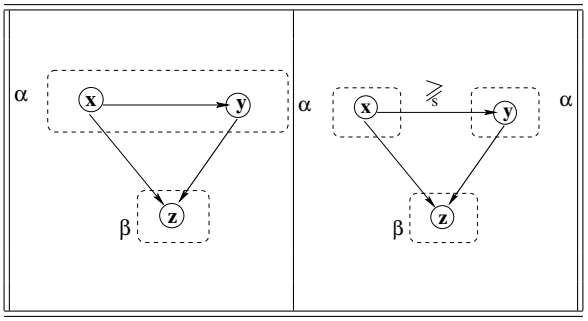

Fig. 1. A labelled graph G (left) and the maximum simulation on G (right).

In order to illustrate our general strategy, we recall here that a *bisimulation* relation is a (sort of) symmetric simulation. As for simulation, there always exists a unique maximum bisimulation relation over a graph, denoted by $\equiv_B$, which is an equivalence relation. One of the key ingredients at the basis of efficient algorithms for finding the maximum bisimulation over a graph consists in redescribing such problem as a *coarsest partitioning problem* (see, e.g., [32, 16]).

Given a labelled graph G, the *simulation problem* consists in determining the maximum simulation preorder on G, and can be elegantly encoded in terms of a *Generalized Coarsest Partition Problem (GCPP)* [21]. Such a formulation is the key step suggesting the space efficient procedure presented in [21, 39] and relies on the fundamental notions of *partition pair* (PP), PP *refinement* and PP *stability*. In this paper we will use a notion of partition pair which slightly generalizes the one introduced in [21]. Moreover, we will consider a different definition of stability and exploit it to introduce the *Coarsest Partition Pair Problem (CPPP)* (cfr. Remark 1). We will show that also in this formulation our partitioning problem is equivalent to the maximum simulation one.

A partition pair consists of a partition of a set V together with a reflexive relation over the classes of such partition. Intuitively, in the case of simulation, the partition represents the simulation equivalence relation, while the reflexive relation is the simulation preorder.

Definition 4 (Partition Pairs). *Let V be a set. A partition pair on V is a pair $\langle \Delta, D \rangle$, where Δ is a partition of V and D is a reflexive relation on Δ.*

Example 3. Consider the set $V = \{1, 2, 3, 4\}$. A partition pair on V is, for instance, $\langle \Delta, D \rangle = \langle \{\{1, 2\}, \{3\}, \{4\}\}, I \cup \{(\{3\}, \{4\}), (\{4\}, \{3\})\} \rangle$, where I is the identity relation over Δ.

Given a set V, each preorder relation $\preceq_P$ on V induces a corresponding partition pair $\langle V_{\equiv_P}, P \rangle$, where $\equiv_P$ is the equivalence relation $\equiv_P = \{(u, v) \mid u \preceq_P v \wedge v \preceq_P u\}$, and $P = \{(\alpha, \beta) \in V_{\equiv_P} \mid \exists u \in \alpha, \exists v \in \beta.(u \preceq_P v)\}$. In particular, given a labelled graph $G = \langle V, E, \Sigma \rangle$, we denote by $\langle V_{\equiv_S}, S \rangle$ the partition pair on V corresponding to the maximum simulation preorder $\preceq_S$ of G.

The notion of refinement establishes a partial order on partition pairs. A partition pair $\langle \Pi, P \rangle$ is said to be *finer* than $\langle \Delta, D \rangle$ if Π splits the blocks of Δ (i.e. Π is *finer* than Δ as classically said among partitions), while P is obtained by removing just *some* of the relationships induced by D on finer blocks. More formally:

Definition 5 (Refinement). *Let $\langle \Delta, D \rangle, \langle \Pi, P \rangle$ be two partition pairs on V:*

$$\langle \Pi, P \rangle \sqsubseteq \langle \Delta, D \rangle \Leftrightarrow \Pi \text{ is finer than } \Delta \text{ and } P \subseteq D(\Pi)$$

where $D(\Pi)$ denotes the relation on Π induced by $D \subseteq \Delta \times \Delta$, i.e.:

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

Given two sets of nodes $A, B \subseteq V$ we write $A \rightarrow_\exists B$ to denote the fact that there exists a node $a \in A$ "reaching" a node $b \in B$, i.e. such that $(a, b) \in E$. Similarly, $A \rightarrow_\forall B$ denotes the fact that all nodes in A reach some node in B. Moreover, given a binary relation D we use the notation $D(\alpha)$ for the set $\{\beta \mid (\alpha, \beta) \in D\}$.

Provided the above notation, Definition 6 introduces the crucial concept of *stability*.

Definition 6 (Stability). *Let $G = \langle V, E, \Sigma \rangle$ be a (labelled) graph, let $\langle \Delta, D \rangle$ be a partition pair on V. $\langle \Delta, D \rangle$ is said* stable *w.r.t. the transition relation of the graph E iff:*

$$\forall \alpha, \beta \in \Delta(\alpha \rightarrow_\exists \beta \Rightarrow \bigcup\{\delta \mid \delta \in D(\alpha)\} \rightarrow_\forall \bigcup\{\delta \mid \delta \in D(\beta)\} \qquad (1)$$

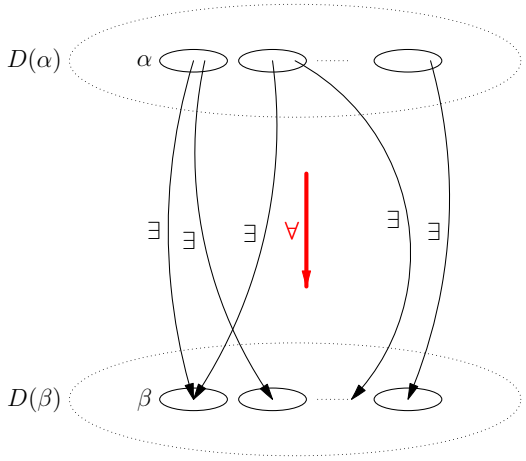

Fig. 2. Pictorial representation of the notion of stability.

In Figure 2 we provide a pictorial representation of our notion of stability, which is further exemplified wihin Example 4.

Example 4. Consider $G = \langle V, E, \Sigma \rangle$, where $V = \{1, 2, 3, 4\}$, $E = \{(1, 3), (2, 4)\}$, and $\Sigma = \{\{1, 2, 3, 4\}\}$ and let $\langle \Delta, D \rangle$ be, for instance, the partition pair on V defined by $\Delta = \{\{1, 2\}, \{3, 4\}\}$ and $D = I \cup \{(\{3, 4\}, \{1, 2\})\}$. $\langle \Sigma, I \rangle$ is not sable with respect to Condition (1), while $\langle \Delta, D \rangle$ is stable with respect to Condition (1).

We are now ready to use our notion of stability to code the simulation problem into a coarsest partition pair problem. Lemma 1 finally proves that such a coding is correct.

Definition 7 (Coarsest Partition Pair Problem (CPPP)). *Let $G = \langle V, E, \Sigma \rangle$ be a labelled graph and consider the identity relation I on Σ. The Coarsest Partition Pair Problem asks to determine the coarsest partition pair $\langle \Pi, P \rangle \sqsubseteq \langle \Sigma, I \rangle$ stable with respect to E.*

Lemma 1 (CPPP as Simulation Problem). *Let $G = \langle V, E, \Sigma \rangle$ be a labelled graph. The coarsest partition pair problem is well defined and admits as unique solution the partition pair $\langle V_{\equiv_S}, S \rangle$, corresponding to the maximum simulation preorder on G.*

Proof. We show that the unique solution to the CPPP is the partition pair $\langle V_{\equiv_S}, S \rangle \sqsubseteq \langle \Sigma, I \rangle$ corresponding to the maximum simulation preorder $\preceq_S$ on $G = \langle V, E, \Sigma \rangle$. We start by proving that $\langle V_{\equiv_S}, S \rangle$ is stable w.r.t. E. Let $\alpha, \beta \in V_{\equiv_S}$ and assume that $\alpha \rightarrow_\exists \beta$. Then, there exist two nodes $s \in \alpha$, $s' \in \beta$ such that $s \rightarrow s'$. Consider an

arbitrary node $p \in \bigcup\{\delta \mid \delta \in S(\alpha)\}$. Since $s \preceq_S p$ and $s \to s'$, there exists a node p' such that $p \to p'$ and $s' \preceq_S p'$. Hence, $p' \in \bigcup\{\delta \mid \delta \in S(\beta)\}$. Our arbitrary choice of $p \in \bigcup\{\delta \mid \delta \in S(\alpha)\}$ guarantees that $\bigcup\{\delta \mid \delta \in S(\alpha)\} \to_\forall \bigcup\{\delta \mid \delta \in S(\beta)\}$, i.e., $\langle V_{\equiv_S}, S\rangle$ is stable w.r.t E.

To conclude our thesis assume, by contradiction, that there exists a partition pair $\langle \Pi, P\rangle \sqsubseteq \langle \Sigma, I\rangle$ stable w.r.t. E and such that $\langle \Pi, P\rangle \not\sqsubseteq \langle V_{\equiv_S}, S\rangle$. Consider the relation $\leq_{\langle \Pi,P\rangle} \subseteq V \times V$, where $\leq_{\langle \Pi,P\rangle} = \{(s, s') \mid ([s]_\Pi, [s']_\Pi) \in P\}$. By our assumption stating that $\langle \Pi, P\rangle \not\sqsubseteq \langle V_{\equiv_S}, S\rangle$, we have that $\leq_{\langle \Pi,P\rangle} \not\subseteq \preceq_S$. Hence, a contradiction follows if we can prove that $\leq_{\langle \Pi,P\rangle}$ is a simulation on $G = \langle V, E, \Sigma\rangle$. In fact, in that case the relation $\leq_{\langle \Pi,P\rangle} \cup \preceq_S$ would be a simulation relation strictly including the maximum simulation preorder $\preceq_S$.

To prove that $\leq_{\langle \Pi,P\rangle}$ is a simulation on $G = \langle V, E, \Sigma\rangle$, let $(s, s') \in \leq_{\langle \Pi,P\rangle}$. By $\langle \Pi, P\rangle \sqsubseteq \langle \Sigma, I\rangle$, we have that $[s]_\Sigma = [s']_\Sigma$. Consider p such that $s \to p$. Then $[s]_\Pi \to_\exists [p]_\Pi$. Since $\langle \Pi, P\rangle$ is stable w.r.t. E we have that $s' \in \bigcup\{\delta \mid \delta \in P([s]_\Pi)\}$ has an edge to a node $p' \in \bigcup\{\delta \mid \delta \in P([p]_\Pi)\}$, i.e. to a node p' such that $(p, p') \in \leq_{\langle \Pi,P\rangle}$. □

Remark 1. Notice that the definition of stability used in this paper is different from the one used in [21], that is:

$$\forall \alpha, \beta, \alpha' \in \Delta(\alpha \to_\exists \beta \wedge \alpha' \in D(\alpha) \Rightarrow \exists \beta'(\beta' \in D(\beta) \wedge \alpha' \to_\forall \beta')). \qquad (2)$$

It is easy to see that in the general case these two notions of stability are not equivalent. As a matter of fact, the notion introduced by Definition 6 can be satisfied also without having a class $\alpha' \in D(\alpha)$ and a class $\beta' \in D(\beta)$ such that $\alpha' \to_\forall \beta'$, as shown in Example 5.

Example 5. Consider for instance the graph $G = \langle V, E, \Sigma\rangle$, where $V = \{1, 2, 3, 4\}$, $E = \{(1, 3), (2, 4)\}$, and $\Sigma = \{\{1, 2, 3, 4\}\}$, and the partition pair $\langle \Delta, D\rangle$ on V defined in Example 3. $\langle \Delta, D\rangle$ is stable with respect to Condition (1), while it is not stable with respect to Condition (2).

Regardless of the above considerations, it is true that if $\langle \Delta, D\rangle$ is such that D is acyclic, then Conditions (1) and (2) are equivalent. Notice also that, differently from [21], here we do not impose acyclicity on D. However, since we will start working with an acyclic relation (i.e. a combination of rank/label-induced partitions), at each step of our computation we will still have to deal with acyclic relations only.

3 The Set-Theoretic Notion of Rank

In the context of Set Theory the notion of rank has a long history and can be found in many early works (see, e.g., [30, 40]). In particular Mirimanoff, while working on Russell's and Burali-Forti's paradoxes, introduced a distinction between *ordinary* (well-founded) and *extraordinary* (non-well-founded) sets: a set X is ordinary if every membership descending chain in X is finite; it is extraordinary, otherwise. He defined the notion of rank of a ordinary set as the least ordinal above the ranks of its members.

A collection of ordinary sets can be considered a set if and only if it has a rank. As a consequence the collection WF of all ordinary sets is not a set.

We believe we can say that the real impact of the notion of rank, however, is revealed by the work of von Neumann, who introduced the cumulative hierarchy of hereditary well-founded sets, i.e., well-founded sets built using only well-founded sets. In such a hierarchy, V_α denotes the set of all sets having ranks less than α. Hence, if we denote by ω the smallest infinite ordinal, V_ω is the set of *hereditary finite* well-founded sets. In [2] Ackermann went further in this direction proposing an encoding of V_ω into natural numbers. Both von Neumann's and Ackermann's constructions are at the basis of the development of algorithms for decidability in Set Theory[2].

When dealing with the combinatorics of Sets, (directed) graphs emerge as a natural representation. For example, hereditary finite sets can be conveniently represented by finite graphs as follows: each node v of the graph represents a (finite) set whose elements are the nodes reachable from v through an edge. Hence, directed acyclic graphs model well-founded hereditary finite sets, while cyclic graphs are necessary for the non-well-founded case. An important feature (drawback?) of such a representation is the fact that, in general, it is *redundant*: two different graphs can represent the same set. Consider $G_1 = \langle \{u, v\}, \{(u, v)\} \rangle$ and $G_2 = \langle \{a, b, c\}, \{(a, b), (a, c)\} \rangle$ both v, b, c represent $\emptyset$, while both u and a represent $\{\emptyset\}$. Introducing (and computing) an equivalence relation on the nodes of a graph representing a set, is a natural way to eliminate such redundancy. Two interesting aspects of such an approach deserve some comment: on the one hand, an efficient computation of such an equivalence relation would be welcome and, on the other hand, we should consider whether this approach would be viable even in the case of non-well-founded sets. If we consider well-founded sets only, the most efficient quotient-computation procedure passes through the introduction of a notion of rank. If we consider a non-well-founded set theory which includes the Anti-Foundation Axiom (AFA), two graphs represent the same set if and only if they are bisimilar and the minimal representation for a set is exactly the one found quotienting w.r.t. the maximum bisimulation relation. Such a strong connection between the notion of bisimulation (which applies to the well-founded case as well) and set equality, makes natural to consider the use of a notion of rank to drive the maximum bisimulation computation. In particular, rewriting the notion of rank on directed acyclic graphs we obtain the following definition.

Definition 8 (Rank). *Let $G = \langle V, E \rangle$ be an acyclic graph, let $v \in V$. The* rank *of the node v is defined as:*

$$\mathit{rank}\,(v) = \begin{cases} 0 & \text{if } E(v) = \emptyset; \\ \max\{1 + \mathit{rank}\,(v') \mid (v, v') \in E\} & \text{otherwise.} \end{cases}$$

In other words, the above notion of rank is the maximum length of path conducing to a sink.

[2] The interested reader can refer to, e.g., [40, 23, 7, 8], for a complete treatment of both fundamental results on Set Theory as well as for an extensive discussion of their relationships with computational issues.

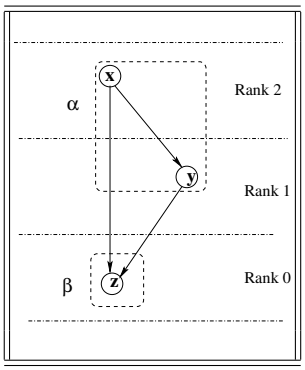

Fig. 3. The rank on the labelled graph G.

Example 6. Consider the labelled graph in Figure 1, described in Example 2. In such a graph, node x has rank 2, node y has rank 1 and node z has rank 0, as illustrated in Figure 3.

Given an acyclic graph, the ranks of its nodes can be computed in linear time with respect to the size of the graph both in the explicit [16] and in the symbolic case [14].

There are two fundamental properties of the notion of rank with respect to bisimulation which are at the basis of the linear time bisimulation subroutine for acyclic graphs described in [16, 14].

Lemma 2 ([16]). *Let $G = \langle V, E, \Sigma \rangle$ be an acylic labelled graph, $\equiv_B$ be the maximum bisimulation relation over G, and $u, v \in V$:*

$$u \equiv_B v \Rightarrow \mathit{rank}\,(u) = \mathit{rank}\,(v) \tag{3}$$

$$u \equiv_B v \Leftrightarrow [u]_\Sigma = [v]_\Sigma \text{ and } \{[u']_{\equiv_B} \mid (u, u') \in E\} = \{[v']_{\equiv_B} \mid (v, v') \in E\} \tag{4}$$

The above result ultimately guarantees that we can proceed computing $\equiv_B$ (inductively) by ranks and we can keep complexity under control as, basically, each edge needs to be processed only once.

Unfortunately the notion of simulation does not have an immediate set theoretic counter-part as in the case of bisimulation. The relation $\preceq_S$ is somehow close to set-theoretic notion of inclusion $\subseteq$, but in order to obtain such a correspondence it is necessary to *enrich* each set with all the elements that its original elements can simulate.

Lemma 3 (Rank and Simulation). *Let $G = \langle V, E, \Sigma \rangle$ be an acylic labelled graph, $\preceq_S$ be the maximum simulation preorder over G, $\equiv_S$ be the maximum simulation equivalence over G, and $u, v \in V$:*

$$u \preceq_S v \Rightarrow \mathit{rank}\,(u) \leq \mathit{rank}\,(v) \tag{5}$$

$$u \preceq_S v \Leftrightarrow [u]_\Sigma = [v]_\Sigma \text{ and } \{[u'']_{\equiv_S} \mid \exists u' \in V((u, u') \in E \wedge u'' \preceq_S u')\} \subseteq \{[v'']_{\equiv_S} \mid \exists v' \in V((v, v') \in E \wedge v'' \preceq_S v')\} \tag{6}$$

Proof. As far as Condition (5) is concerned, assume by contradiction that there exist two nodes $s \in V$, $s' \in V$ such that $s \preceq_S s'$ and $\mathit{rank}(s) > \mathit{rank}(s')$. Let $r = \mathit{rank}(s) > \mathit{rank}(s')$. Since $\mathit{rank}(s) = r$, we can determine a sequence of r nodes $s_1, \ldots, s_r$ such that $s \to s_1 \wedge (\bigwedge_{1 \leq i < r} s_i \to s_{i+1})$. By definition of simulation, there exists a corresponding sequence of r nodes $s'_1, \ldots, s'_r$ such that

$$s' \to s'_1 \wedge s_1 \preceq_S s'_1 \wedge \left(\bigwedge_{1 \leq i < r} s_i \to s_{i+1} \wedge s_{i+1} \preceq_S s'_{i+1} \right)$$

Such a sequence of nodes witnesses the fact that $\mathit{rank}(s') \geq r$, which contradicts our hypothesis.

We now prove Condition (6). We start by proving direction ($\Rightarrow$). By definition of simulation, we have that $u \preceq_S v$ implies $[u]_\Sigma = [v]_\Sigma$. Let $[u'']_{\equiv_S}$ be such that there exists u' with $(u, u') \in E$ and $u'' \preceq_S u'$. Since $u \preceq_S v$ there exists v' such that $(v, v') \in E$ and $u' \preceq_S v'$. Since $\preceq_S$ is transitive we get that $u'' \preceq_S v'$, i.e., the thesis. As far as direction ($\Leftarrow$) is concerned, if $(u, u') \in E$, then since $u' \preceq_S u'$ we have that $[u']_{\equiv_S} \in \{[u'']_{\equiv_S} \mid \exists u' \in V((u, u') \in E \wedge u'' \preceq_S u')\}$. So, from our hypothesis it follows that $[u']_{\equiv_S} \in \{[v'']_{\equiv_S} \mid \exists v' \in V((v, v') \in E \wedge v'' \preceq_S v')\}$. This means that there exist $\overline{u}$ and v' such that $u' \equiv_S \overline{u}$, $(v, v') \in E$, and $\overline{u} \preceq_S v'$. Since $\preceq_S$ is transitive, we get $(v, v') \in E$ and $u' \preceq_S v'$. Our hypothesis guarantees that $[u]_\Sigma = [v]_\Sigma$, so we can conclude $u \preceq_S v$. □

Notice that, while in the case of bisimulation the initial labels on the nodes can be omitted introducing new nodes and egdes (see [16]), the same cannot be done in the case of simulation (see [21]).

The above properties will be the starting point for our rank-based simulation algorithm. In fact, to obtain an efficient algorithm we, basically, need to:

- process the edge (u, u') when we are considering the nodes at rank $\mathit{rank}(u')$;
- avoid to explicitly computing the sets $\{[u'']_{\equiv_S} \mid \exists u' \in V((u, u') \in E \wedge u'' \preceq_S u')\}$ and, instead, exploit both $\equiv_S$ and $\preceq_S$ to *implicitly* evaluate set inclusions.

4 An Optimal Simulation Algorithm on Acyclic Graphs

In this section, we introduce our optimal (w.r.t. both time and space) simulation algorithm on acyclic graphs. Such a procedure relies on solving the coarsest partition pair problem (i.e. computing the maximum simulation preorder) proceeding by *ranks*. As a consequence of Lemma 3, the notion of rank, introduced in Definition 8, allows one to perform a preliminary partition in the given labelled graph and to drive the successive computation. Condition (5) of Lemma 3 in terms of partition pairs can be restated as follows.

Lemma 4. *Let $G = \langle V, E, \Sigma \rangle$ be an acyclic labelled graph. Then:*

$$\langle V_{\equiv_S}, S \rangle \sqsubseteq \langle V_{\equiv_R}, R \rangle$$

where $\langle V_{\equiv_S}, S\rangle$ *is the partition pair on* V *encoding the maximum simulation on* G*, and* $\langle V_{\equiv_R}, R\rangle$ *is the partition pair on* V *corresponding to the rank-labelling preorder* $\preceq_R = \{(u,v) \mid \mathit{rank}\,(u) \leq \mathit{rank}\,(v)\}$.

Proof. This is an immediate consequence of Lemma 3. □

As a consequence of the above circle of ideas, we get the algorithm presented in Table 1, to be used on acyclic graphs.

Algorithm RANKSIM consists, mainly, of two phases: A preprocessing stage realized through lines 1–11 and a main loop consisting of lines 12–28.

The preprocessing performs the rank-partitioning of the given labelled graph. Such an initial partition is then explored, in turn, proceeding from the lowest to the highest rank, corresponding to successive executions of the main loop at line 12. At each iteration i of the main loop (corresponding to rank $i-1$) the transitions targeting nodes having rank $i-1$ are employed to refine the current partition pair, in order to get us closer to the validity of the stability property. In particular, a class α at rank at least i is split using a class β at rank $i-1$ when there are both elements of α reaching elements of β and elements of α *not* reaching any class which simulates β. Similarly, a class γ simulating α can be split if it contains elements reaching classes simulating β as well as elements which do not.

Notice that even if in our definition of partition pair we do not require to have an acyclic relation, all the partition pairs generated in our algorithm are acyclic. As a matter of fact, a cyclic partition pair would represent a wrong split and would require a subsequent union to converge to the correct output.

The following subsection further illustrates the execution of our rank-based simulation algorithm on some concrete example.

4.1 The Algorithm at Work on Some Examples

In this subsection we consider two concrete labelled graphs, and we show how the algorithm RANKSIM of Table 1 computes the maximum simulation on them.

We start by employing as input for our rank-based simulation algorithm the labelled graph $G = \langle V, E, \Sigma\rangle$ depicted in Figure 1 and illustrated within Example 2. In this simple case, the algorithm RANKSIM terminates upon the execution of the first loop at line 3, that uses the information given by the rank to refine the initial partition pair $\langle \Sigma, I\rangle$, where $\Sigma = \{\alpha = \{x,y\}, \beta\{z\}\}$, to the partition pair $\langle \Pi, P\rangle$, where $\Pi = \{\alpha = \{x\}, \alpha_1 = \{y\}, \beta = \{z\}\}$ and $P = I \cup \{(\alpha_1, \alpha)\}$. In fact, in this case the preprocessing provided by the loop at line 3 is sufficient to solve the simulation problem.

As a more involved example, consider the labelled graph depicted in Figure 4, where the three labelling letters α, β, δ induce the initial partition pair $\langle \Sigma, I\rangle$, with $\Sigma = \{\alpha = \{m,n\}, \beta = \{o,p\}, \delta = \{q,r\}\}$. The rank labelling performed by the loop at line 3 refines the partition pair $\langle \Sigma, I\rangle$ to the partition pair $\langle \Pi, P\rangle$, where $\Pi = \{\alpha = \{m,n\}, \beta = \{o,p\}, \delta = \{q\}, \delta_1 = \{r\}\}$ and $P = I \cup \{(\delta_1, \delta)\}$. The new class δ_1 generated by such a preprocessing loop is then used within the second iteration of the main loop at line 12 (corresponding to the analysis of the classes at rank 1) to split the higher ranked class α. In particular, such a split refines the partition pair $\langle \Pi, P\rangle$ to the partition

Algorithm 1: RANKSIM

input : $G = \langle V, E, \Sigma \rangle$,$I := \{(\alpha, \alpha) \mid \alpha \in \Sigma\}$

output: $\langle V_{\equiv_S}, S \rangle$: partition pair encoding the maximum simulation preorder $\preceq_S$ on G.

begin

  /* Initial refinement & rank-ordering of the classes. */

  $notRanked := V; rankMax := -1; sim := I;$

  repeat

    forall $\alpha \in \Sigma \mid \alpha \subseteq notRanked \wedge \alpha \nsubseteq pre(notRanked)$ **do**

      if $\alpha \neq \alpha \setminus pre(notRanked)$ **then**

        $\alpha_1 := \alpha \setminus pre(notRanked); \alpha := \alpha \setminus \alpha_1; rank(\alpha_1) := rankMax+1;$
        $\Sigma := \Sigma \cup \{\alpha_1\}; sim := sim \cup \{(\gamma, \alpha_1) \mid (\gamma, \alpha) \in sim\} \cup \{(\alpha_1, \alpha)\}$

      else

        $rank(\alpha) := rankMax + 1$

    $notRanked := pre(notRanked); rankMax := rankMax + 1$

  until $pre(notRanked) = \emptyset$

  /* Process Σ by rank & refine $\langle \Sigma, sim \rangle$ to establish the stability prop. $\forall(\alpha, \beta).(\alpha \rightarrow_\exists \beta \Rightarrow sim(\alpha) \rightarrow_\forall sim(\beta))$. */

  for $rk = 1$ **to** $rankMax$ **do**

    foreach $\beta \in \Sigma \mid rank(\beta) = rk - 1$ **do**

      foreach $\alpha \mid pre(\beta) \cap \alpha \neq \emptyset$ **do**

        if $\alpha \nsubseteq pre(sim(\beta))$ **then**

          $\alpha_1 := \alpha \setminus pre(sim(\beta)); rank(\alpha_1) := rank(\alpha); \alpha := \alpha \setminus \alpha_1;$
          $\Sigma := \Sigma \cup \alpha_1$
          $sim := sim \cup \{(\alpha_1, \delta) \mid (\alpha, \delta) \in sim\}$
          $\cup \{(\delta, \alpha_1) \mid (\delta \neq \alpha, \alpha) \in sim\}$

        foreach $\gamma \mid \gamma \nsubseteq pre(sim(\beta)) \wedge \gamma \in sim(\alpha)$ **do**

          $\gamma_1 := \gamma \setminus pre(sim(\beta))$

          if $\gamma_1 \neq \gamma$ **then**

            $\gamma := \gamma \setminus \gamma_1; rank(\gamma_1) := rank(\gamma); \Sigma := \Sigma \cup \gamma_1$ $sim :=$
            $sim \cup \{(\gamma_1, \delta) \mid (\gamma, \delta) \in sim\} \cup \{(\delta, \gamma_1) \mid (\delta \neq \gamma, \gamma) \in sim\}$

          $sim := sim \setminus \{(\alpha, \gamma_1)\}$

end

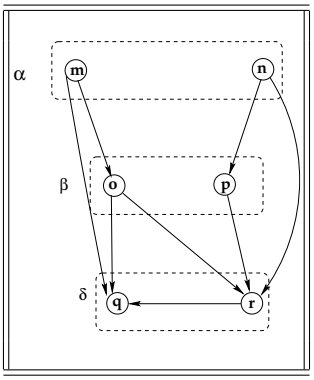

Fig. 4. A labelled graph G_1.

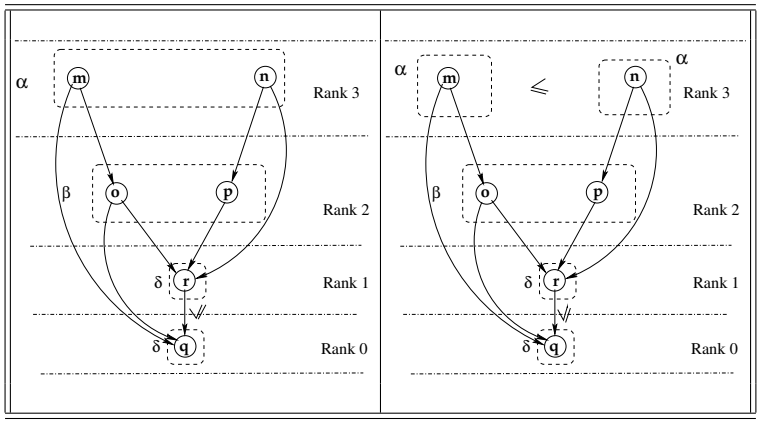

Fig. 5. Rank-Based Simulation at work on the labelled graph G_1

pair $\langle \Pi', P' \rangle$, where $\Pi' = \{\alpha = \{m\}, \alpha_1 = \{n\}, \beta = \{o, p\}, \delta = \{q\}, \delta_1 = \{r\}\}$ and $P = I \cup \{(\delta_1, \delta), (\alpha, \alpha_1)\}$.

5 Correctness and Complexity Results

In this section, we prove the correctness of our simulation algorithm on acyclic graphs and we establish its complexity.

5.1 Correctness

Lemma 5, below, shows that the preprocessing step correctly computes the partition pair induced by the rank labelling of the graph.

Lemma 5. *The loop at line* 3 *in the algorithm* RANKSIM$(\langle V, E, \Sigma \rangle, I)$ *terminates computing the partition pair* $\langle \Pi, P \rangle$ *and the variable rankMax, where:*

- *rankMax*$= \max\{\textit{rank}\,(v) \mid v \in V\}$,
- $\Pi \sqsubseteq \Sigma$ *is the coarsest partition finer than the rank-labelling partition* $V_{\equiv_R}$,
- $P = \{(\alpha, \beta) \mid \textit{rank}\,(\alpha) \leq \textit{rank}\,(\beta) \wedge \exists \gamma \in \Sigma(\alpha \subseteq \gamma \wedge \beta \subseteq \gamma)\}$

Proof. This can be easily proved by induction on the number of loop iterations. □

Lemma 6 and Theorem 1 define crucial invariants and prove their validity throughout the execution of the main loop at line 12. Such invariants imply the correctness of our simulation algorithm.

Lemma 6. *Consider the overall execution of the for-loop at line* 12 *guarded by the variable* $1 \leq rk \leq rankMax$*. Whenever a class* γ *is involved either in a split or in a refinement of its simulator set* $\bigcup\{\beta \mid (\gamma, \beta) \in sim\}$*, the following statement holds:*

$$\textit{rank}\,(\gamma) \geq rk$$

Proof. Denote by $\langle \Sigma_i, sim_i \rangle$ the partition pair in input to the i-th iteration of the for-loop at line 12. Moreover, if γ is a class processed within the i-th execution of the for-loop at lines 12, let γ_i denote the unique class $\gamma_i \in \Sigma_i$ such that $\gamma_i \supseteq \gamma$.

Given the above notations, consider the i-th execution of the for-loop at lines 12 (for which $rk = i$) and let γ be a class in a partition pair processed within such an iteration. Assume that either $\gamma \subset \gamma_i$, or $\bigcup\{\beta \mid (\gamma, \beta) \in sim\} \neq \bigcup\{\beta_i \mid (\gamma_i, \beta_i) \in sim_i\}$. Then, there exists a pair of classes $\{\alpha, \beta\} \subseteq \Sigma_i$ such that $\textit{rank}\,(\beta) = i - 1$, $\alpha \rightarrow_\exists \beta$ and $\gamma_i \in sim_i(\alpha)$. $\textit{rank}\,(\beta) = i - 1 \wedge \alpha \rightarrow_\exists \beta$ implies $\textit{rank}\,(\alpha) \geq i$. Lemma 5 allows then to conclude that $\textit{rank}\,(\gamma) = \textit{rank}\,(\gamma_i) \geq i$. □

Theorem 1. *The following invariants hold at the beginning of each iteration of the for-loop at line* 12*, within the algorithm* RANKSIM$(\langle V, E, \Sigma \rangle, I)$*.*

1. *For each node* $v \in V$ *such that* $\textit{rank}\,(v) < rk$*:*

$$[v]_{\equiv_S} = \alpha \in \Sigma \wedge \bigcup\{u \mid v \preceq_S u\} = \bigcup\{\beta \mid (\alpha, \beta) \in sim\}$$

2. *For each node* $v \in V$ *such that* $\textit{rank}\,(v) \geq rk$*:*

$$[v]_{\equiv_S} \subseteq \alpha \in \Sigma \wedge \bigcup\{u \mid v \preceq_S u\} \subseteq \bigcup\{\beta \mid (\alpha, \beta) \in sim\}$$

3. *For each pair of classes* $\alpha, \beta \in \Sigma$*:*

$$(\alpha \rightarrow_\exists \beta \wedge \textit{rank}\,(\beta) < rk - 1) \Rightarrow \bigcup\{\gamma \mid (\alpha, \gamma) \in sim\} \rightarrow_\forall \bigcup\{\gamma \mid (\beta, \gamma) \in sim\}$$

Proof. By induction on the number of iterations of the for-loop at line 11.
Base $(rk = 1)$. The first two items in our statement follow directly from Lemma 4 and Lemma 5, while the third item holds trivially since no class $\beta \in \Sigma$ has a rank strictly lower than $rk - 1 = 0$.
Inductive Step $(1 < rk)$. Given $i \geq 1$, denote by $\langle \Sigma_i, sim_i \rangle$ the partition pair in input to the i-th execution of the for-loop at line 12. Given $\gamma \in \Sigma_{i>1}$, denote by γ_{i-1} the only class in Σ_{i-1} such that $\gamma_{i-1} \supseteq \gamma$.

1. In order to prove our inductive step for item (3), consider $\langle \Sigma_{i=rk>1}, sim_{i=rk>1} \rangle$ and let $\alpha \in \Sigma_i$ such that $\alpha \to \beta \wedge \mathit{rank}(\beta) < i-1$. By $\mathit{rank}(\beta) < i-1$ and Lemma 6 we have $\beta_{i-1} = \beta$ and $\bigcup\{\delta \mid (\beta,\delta) \in sim_i\} = \bigcup\{\delta \mid (\beta_{i-1},\delta) \in sim_{i-1}\} = sim\beta$. Hence, if $\mathit{rank}(\beta) < (i-1)-1$ we can conclude our thesis exploiting the inductive hypothesis for which $\bigcup\{\delta \mid (\alpha_{i-1},\delta) \in sim_{i-1}\} \to_\forall sim\beta$. Otherwise, assume $\mathit{rank}(\beta) = i-1$ and suppose by contradiction that $(\bigcup\{\delta \mid (\alpha,\delta) \in sim_i\} \not\to_\forall sim\beta)$. Let $\alpha^* \supset \alpha$ be the superclass of α at the moment in which β gets selected at line 13 with $rk = i-1$. Within the execution of the most internal foreach-loop at line 14, if α^* contains some state that does not reach $sim\beta$, then α^* gets split into the two subclasses: $\alpha_1^* := \alpha^* \setminus sim\beta$ and $\alpha^* := \alpha^* \setminus \alpha_1$. By $\alpha \to_\exists \beta \subseteq sim\beta$, we have that the statement $\alpha \subseteq \alpha^*$ is true both before and after such a split. Moreover, the loop at line 19 processes each class $\gamma \in sim_{i-1}^*(\alpha^* \supseteq \alpha)$. If γ contains some state that does not reach $sim\beta$, then γ gets split into the two subclasses $\gamma_1 := \gamma \setminus sim\beta$ and $\gamma := \gamma \setminus \alpha_1$, and γ_1 is removed from the simulators of $\alpha^* \supseteq \alpha$ (line 23). Hence, once β have been considered at line 13 within the $i-1$-th execution of the for-loop at line 12, we have that each node belonging to a class simulating α^* has a successor in $sim\beta$. Each subsequent refinement of α^* or its set of simulators will maintain this property, and thus we get to the contradiction of our assumption $(\bigcup\{\delta \mid (\alpha,\delta) \in sim_i\} \not\to_\forall sim\beta$
2. We now proceed proving the inductive step for item (1). Let $i = rk > 1$ and consider the $\langle \Sigma_i, sim_i \rangle$. Let $v \in V$ such that $\mathit{rank}(v) < rk-1$. Then, item (1) holds by Lemma 6 and by inductive hypothesis. Let v such that $\mathit{rank}(v) = rk-1$. Then, by inductive hypothesis on item (2) we have:

$$[v]_{\equiv_S} \subseteq \alpha \in \Sigma \wedge \bigcup\{u \mid v \preceq_S u\} \subseteq \bigcup\{\beta \mid (\alpha,\beta) \in sim\} \tag{7}$$

   Moreover, the inductive step already proved on item (3) ensures that:

$$(\alpha \to_\exists \beta \wedge \mathit{rank}(\beta) < rk-1) \Rightarrow \bigcup\{\gamma \mid (\alpha,\gamma) \in sim\} \to_\forall \bigcup\{\gamma \mid (\beta,\gamma) \in sim\} \tag{8}$$

   Since $\mathit{rank}(\alpha) = rk-1$, any class reached by α has rank strictly lower than $rk-1$. Hence, Conditions (7) and (8) guarantee that:

$$[v]_{\equiv_S} = \alpha \in \Sigma \wedge \bigcup\{u \mid v \preceq_S u\} = \bigcup\{\beta \mid (\alpha,\beta) \in sim\}$$

   completing our inductive step for item (1).
3. Let $i = rk > 1$ and consider v such that $rank(v) \geq rk$. If $rank(v) > rk$, then $[v]_{\Sigma_i} = [v]_{\Sigma_{i+1}} \subseteq \alpha$ by inductive hypothesis. If $rank(v) = rk$, there are two cases to consider. In the first case, $[v]_{\Sigma_i} = [v]_{\Sigma_{i+1}}$ (i.e., the class of v gets not split within the i-th iteration of the loop at line 13) and we are done. In the second case, $[v]_{\Sigma_i}$ gets split into α, α_1 within the i-th execution of the loop at line 13. By contradiction, assume there exist two states $u \in \alpha, u' \in \alpha_1$ such that $u \equiv_S u'$. By definition of α, α_1, u has a successor into a class β of rank $r < rk$, and u' has no successor in any class $\beta' \in sim(\beta)$. By inductive hypothesis, this implies that there exists z such that $(u,z) \in E$ and u' has no successor z' such that $z \leq_S z'$, contradicting our hypothesis $u \equiv_S u'$.

□

5.2 Complexity

We finally establish the complexity of our simulation algorithm on acyclic graphs. In particular, Theorem 2 shows that our procedure uses $\mathcal{O}(|E||V_{\equiv_S}|)$ time and requires $O(|V_{\equiv_S}|^2 + |V|\log(|V_{\equiv_S}|))$ bits to compute a simulation preorder on a given acyclic labelled graph $G = \langle V, E, \Sigma\rangle$. Therefore, it has optimal performances w.r.t. both time and space on acyclic graphs, outperforming [36, 37, 21].

Theorem 2. *The algorithm* RANKSIM$(G = \langle V, E, \Sigma\rangle, I)$ *performs* $O(|V_{\equiv_S}||E|)$ *steps and uses* $O(|V_{\equiv_S}|^2 + |V|\log(|V_{\equiv_S}|))$ *bits to compute the solution to the coarsest partition pair problem* $\langle V_{\equiv_S}, S\rangle$.

Proof. Let $r = \max\{\mathit{rank}\,(v) \mid v \in V\}$. The cost of the while-loop at line 3 is $O(r * |E|) = O(|V_{\equiv_S}||E|)$.

The cost of the loop at line 12, excluded the execution of the innermost if-statement at line 21, is:

$$O(\Sigma_{1=1}^{r}\Sigma_{\beta\in V_{\equiv_S},\mathit{rank}\,(\beta)=i-1}(|pre(\beta)| * |\Sigma_{i+1}| + |pre(\bigcup\{\delta \mid (\beta,\delta) \in sim\}|) =$$

$$= O(|V_{\equiv_S}||E|)$$

In fact, consider a class β such that $\mathit{rank}\,(\beta) = i - 1$. It is possible to distinguish with marks the classes that reach (resp. do not reach/reach with all their nodes) the set $\bigcup\{\delta \mid (\beta,\delta) \in sim\}$ at the cost $O(pre(\bigcup\{\delta \mid (\beta,\delta) \in sim\})$. Moreover, the same cost allows one to appropriately mark each node in $pre(\bigcup\{\delta \mid (\beta,\delta) \in sim\}$. Then, fixed β, $\mathit{rank}\,(\beta) = i-1$, the cost of executing lines 14–24 without considering the innermost if-statement, is $O(|pre(\bigcup\{\delta \mid (\beta,\delta) \in sim\}| + \Sigma_{\alpha\in pre(\beta)}|sim(\alpha)|)$.

The innermost if-statement at lines 21–23 is executed only upon the creation of a new class γ_1 and cost globally $O(|V_{\equiv_S}||E|)$. In fact, each execution of lines 21–23 for the creation of the new classes $\gamma_1, \gamma \setminus \gamma_1$ from γ, requires only to scan the nodes in γ and the classes in $sim(\gamma), sim^{-1}(\gamma)$.

As far as space complexity is concerned we refer to bit complexity without considering the space required by the graph G, since it is never modified (see, e.g., [33, 6]). In particular, Σ is stored through an array of length $|V|$ associating to each node its class. Hence, it requires $O(|V|\log(|V_{\equiv_S}|))$ bits. $notRanked$ is a $|V|$ array of bits, labeling with 1 the nodes which do not have a rank. $rank$ is stored in a $|V|$ array of lists, where the ith list keeps the classes at rank i. Hence it requires $O(|V| + |V_{\equiv_S}|\log(|V_{\equiv_S}|))$ bits. Finally, the relation sim is stored in a bit matrix whose size grows up to $O(|V_{\equiv_S}|^2)$ bits. □

6 Back To The Notion of Rank

In order to generalize our algorithm to the cyclic case we could follow two opposite approaches. On the one hand, we could extend the notion of rank ensuring that:

$$u \preceq_S v \Rightarrow \mathit{rank}\,(u) \leq \mathit{rank}\,(v) \tag{9}$$

$$(u, u') \in E \Rightarrow \mathit{rank}\,(u) \geq \mathit{rank}\,(u') \tag{10}$$

These would allow us to both start from the rank partition and to compute the simulation equivalence at rank i using only the information computed at rank at most i. On the other hand, we could decide to drop Condition (9) in order to use a notion of rank as fine as possible with respect to Condition (10). In this case we would need to both split and merge classes, but we could consider at each step relatively small sets of nodes.

In more detail, the first policy imposes us to put all the nodes which reach a cycle at the same rank.

Example 7. Consider the labelled graph $G = \langle V, E, \Sigma \rangle$ such that $V = \{a, b, c\}$, $E = \{(a, a), (c, b), (c, c)\}$, and $\Sigma = \{V\}$. We have that $a \equiv_S c$.

More in general, cycles allow to simulate acyclic chains of arbitrary length and for this reason we cannot discriminate *a priori* among nodes which reach cycles. Hence, the only possible extension of the notion of rank which satisfies both (9) and (10) and which does not exploit node labelling is the following:

$$\mathit{rank}^*(v) = \begin{cases} 0 & \text{if } E(v) = \emptyset, \\ \max\{1 + \mathit{rank}^*(v') \mid (v, v') \in E\} & \text{if } v \text{ does not reach cycles,} \\ +\infty & \text{otherwise.} \end{cases}$$

In this case we would "only" have to extend our algorithm to deal with rank $+\infty$. Unfortunately, this is the most complex case and it could include a large set of nodes.

The second policy allows us to use a notion of rank considering each strongly connected component separately. In particular, given a graph G let G^{scc} be the acyclic graph of its strongly connected components and consider:

$$\mathit{rank}_*(v) = \mathit{rank}\,(scc(v))$$

where $scc(v)$ is the strongly connected component of v. In this case the difficulty in developing a competitive simulation algorithm lies in the merging operations. In the graph of Example 7 nodes a and c are simulation equivalent, but they are at different rank_*.

We conclude this section briefly mentioning other cases of problems over graphs in which the set theoretic notion of rank and its extensions to the cyclic case have been used to find algorithmic solutions.

In Section 3 we already discussed the use of the classical notion of rank for bisimulation computation over acyclic graphs. In [16, 14] an extension of the notion of rank based on strongly connected components (different from rank_*) has been exploited for efficiently computing bisimulation in the general case. The computation proceed by ranks and at each steps only the nodes at a given rank are considered. However, when cycles are involved a single step can require a number of iterations which is more than linear with respect to its dimension.

In [26] a more abstract notion of rank is used for evaluating and comparing web structures. A set of web pages can be seen as a graph whose nodes are the pages, while the edges encode the links among pages. In their analysis the authors exploit a notion of rank which is slightly different from our. In particular, they introduce a hierarchy of ranks. At the lower level of such hierarchy rank_0 is the length of the shortest path

from a node to a leaf (a node without outgoing edges). At the next level $\mathit{rank}_1(v)$ is the set $\{rank_0(v') \mid (v, v') \in E\}$, and so on. The authors observe that for each graph G there exists k such that two nodes have the same rank_k if and only if they are bisimilar. Unfortunately, in the case of simulation this notion of rank is not useful. Consider for instance the graph $G = \langle V, E, \Sigma \rangle$ such that $V = \{a, b, c\}$, $E = \{(a, a), (b, b), (b, c)\}$, and $\Sigma = \{V\}$. It holds that $a \equiv_S b$, but for each k $\mathit{rank}_k(a) \neq \mathit{rank}_k(b)$. In particular, for $k > 0$ we have $\mathit{rank}_k(a) = \{\infty\}^k$, while $\mathit{rank}_k(b)$ has two elements which are both different from $\{\infty\}^{k-1}$.

In [19] the authors consider Relational and XML Databases proposing translations from the first to the second ones. Two problems over graphs emerged from this analysis. The maximum density problem requires to extract a forest with maximum number of edges (or equivalently minimum number of roots) from a given graph. It can be solved in linear time using rank_*. The maximum depth problem asks for a forest with maximum depth, where the depth of a forest is the sum of the depths of its nodes. In the acyclic case also this problem can be solved in linear time using the notion of rank. In the general case this second problem is NP-complete. However, again rank_* can be used to compute in linear time an approximated solution. The approximation degree depends on the structure of the graph.

7 Conclusions and Future Work

The algorithm presented in this paper is strongly based on a characterization and statement of the maximum simulation quotient and preorder computation, as a coarsest partition *pair* problem. Such a view was proved useful and exploited also in the context of maximum bisimulation computation (see [15, 16]) and, ultimately, links this problem with set theory. Not surprisingly, the notion of rank is a standard tool in well-founded set theory, used to give "structure" to the universe of sets (see, e.g., [38]). As a matter of fact, for example, the algorithm for bisimulation computation presented in [15, 16] has a linear time complexity in the acyclic case, while in the general case it allows one to focus the computation on subgraphs of the given graph.

The rank-based simulation algorithm we presented here has very good space/time performances on acyclic graphs. However, its generalization to the (possibly) cyclic case is still an open problem. One reason for such difficulties lies in the fact that, when moving from acyclic to cyclic graphs the notion of simulation does not work "compositionally": loops allow to simulate paths of arbitrary length.

Further direction for future work are given by the open problem concerning the possibility of developing a linear time simulation algorithm for the acyclic case. Again, there seems to be an intrinsic higher complexity in the notion of simulation with respect to bisimulation which prevents one to easily achieve the linear time complexity. In particular, even though with Lemma 3 we were able to give a set-theoretic "flavour" to the notion of simulation, the induced characterization is just reminiscent of a sort of recursive inclusion. This is in contrast with the notion of bisimulation on acyclic graphs that corresponds *exactly* to equality on well-founded sets.

Finally, we could consider the algorithmic analysis of the quantitative extension of the notion of simulation recently introduced in [3] to deal with the containment problem

on quantitative languages [9, 18], ultimately applying to the verification of quantitative properties on complex systems.

References

1. E. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: from relations to semistructured data and XML*. Morgan Kaufmann, 2000.
2. W. Ackermann. Die widerspruchsfreiheit der allgemeinen mengenlehre. *Mathematische Annalen*, 114:305–315, 1937.
3. Guy Avni and Orna Kupferman. Making weighted containment feasible: a heuristic based on simulation and abstraction. In *Proceedings of the 23rd international conference on Concurrency Theory*, CONCUR'12, pages 84–99. Springer-Verlag, 2012.
4. K. Bauer, R. Gentilini, and K. Schneider. A uniform approach to three-valued semantics for -calculus on abstractions of hybrid automata. *International Journal on Software Tools for Technology Transfer*, 13(3):273–287, 2012.
5. M. Ben-Ari, T. Milo, and E. Verbin. Querying dag-shaped execution traces through views. In *WebDB: 12th International Workshop on the Web and Databases*, 2009.
6. D. Bustan and O. Grumberg. Simulation-based minimization. *ACM Trans. Comput. Logic*, 4(2):181–206, April 2003.
7. D. Cantone, A. Ferro, and E. Omodeo. *Computable set theory, Volume 1*. Clarendon Press, 1990.
8. D. Cantone, E. Omodeo, and A. Policriti. *Set Theory for Computing*. Springer, 2001.
9. Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Trans. Comput. Logic*, 11(4):23:1–23:38, 2010.
10. E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logics. *Logic of Programs*, pages 52–71, 1981.
11. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. Elsevier/MIT press, 2001.
12. R. Cleaveland and L. Tan. Simulation revisited. In T. Margaria and W. Yi, editors, *Proc. 7th Int'l Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, volume 2031 of *LNCS*, pages 480–495. Springer, 2001.
13. D. Deutch and T. Milo. A quest for beauty and wealth (or, business processes for database researchers). In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '11, pages 1–12. ACM, 2011.
14. A. Dovier, R. Gentilini, C. Piazza, and A. Policriti. Rank-based symbolic bisimulation: (and model checking). *Electronic Notes in Theoretical Computer Science*, 67:166–183, 2002.
15. A. Dovier, C. Piazza, and A. Policriti. A fast bisimulation algorithm. In G. Berry, H. Comon, and A. Finkel, editors, *Proceedings of Computer Aided Verification (CAV'01)*, volume 2102 of *LNCS*, pages 79–90. Springer, 2001.
16. A. Dovier, C. Piazza, and A. Policriti. An efficient algorithm for computing bisimulation equivalence. theoretical computer science. *Theor. Comput. Sci*, 311:221–256, 2004.
17. B. Smith et al. The obo foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature Biotechnology*, pages 1251–1255, 2004.
18. Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. Quantitative languages defined by functional automata. In *Proceedings of the 23rd international conference on Concurrency Theory*, CONCUR'12, pages 132–146. Springer-Verlag, 2012.
19. M. Franceschet, D. Gubiani, A. Montanari, and C. Piazza. From entity relationship to xml schema: A graph-theoretic approach. In *Proceedings of the 6th International XML Database Symposium on Database and XML Technologies*, pages 165–179. Springer-Verlag, 2009.

20. R. Gentilini, C. Piazza, and A. Policriti. Simulation as coarsest partition problem. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS '02, pages 415–430. Springer-Verlag, 2002.
21. R. Gentilini, C. Piazza, and A. Policriti. From bisimulation to simulation: Coarsest partition problems. *J. Autom. Reasoning*, 31(1):73–103, 2003.
22. R. Gentilini, K. Schneider, and B. Mishra. Successive abstractions of hybrid automata for monotonic ctl model checking. In *Proceedings of the international symposium on Logical Foundations of Computer Science*, LFCS '07, pages 224–240. Springer-Verlag, 2007.
23. J. Van Heijenoort. *From Frege to Godel: A Source Book in Mathematical Logic, 1879-1931*. Harvard University Press, 1977.
24. J. Hellings, G. Fletcher, and H. Haverkort. Efficient external-memory bisimulation on dags. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 553–564. ACM, 2012.
25. M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, pages 453–462. IEEE Computer Society Press, 1995.
26. I. Horie, K. Yamaguchi, and K. Kashiwabara. Higher-order rank analysis for web structure. In *Proceedings of the sixteenth ACM conference on Hypertext and hypermedia*, pages 98–106. ACM, 2005.
27. K. Kaneko and J. Zhu. Xsim: The first method for generating the simulation quotient of xml documents in a relational database. In *Proceedings of the 2012 International Conference on Future Information Technology and Management Science*, FITMSE '12, pages 53–61, 2012.
28. K. Laiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6(1):11–44, 1995.
29. R. Milner. A calculus of communicating systems. In G. Goos and J. Hartmanis, editors, *Lecture Notes on Computer Science*, volume 92. Springer, 1980.
30. D. Mirimanoff. Les antinomies de russell et de burali-forti et le problme fondamental de la thorie des ensembles. *Enseign. math.*, 19:37–52, 1917.
31. O. Grumberg and D.E. Long. Model checking and modular verification. *ACM Transactions on Programming Languages and systems*, 16(3):843–871, May 1994.
32. R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.
33. C. H. Papadimitriou. *Computational complexity*. Addison-Wesley Publishing Company, Inc., 1994.
34. D. Park. Concurrency on automata and infinite sequences. *Theoretical Computer Science*, pages 167–183, 1981.
35. P. Ramanan. Covering indexes for xml queries: bisimulation - simulation = negation. In *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '03, pages 165–176, 2003.
36. F. Ranzato and F. Topparo. A new efficient simulation equivalence algorithm. In *Proceedings of Logics in Computer Science (LICS'07)*, pages 171–180, 2007.
37. F. Ranzato and F. Topparo. Saving space in a time efficient simulation algorithm. In *Proceedings of Int. Conference on Application of Concurrency to System design (ACSD'09)*, pages 60–69, 2009.
38. J. E. Rubin. *Set Theory for the Mathematician*. New York: Holden-Day, 1967.
39. R. van Glabbeek and B. Ploeger. Correcting a space-efficien simulation algorithm. In *Proceedings of Int. Conference on Computer Aided Verification (CAV'08)*, pages 517–529, 2008.
40. J. von Neuman. *DCollected Works. Volume I: Logic, Theory of Sets and Quantum Mechanics*. Pergamon Press, 1961.