Simulation reduction as constraint

R. Gentilini, C. Piazza, and A. Policriti
Dipartimento di Matematica e Informatica Università di Udine,
Via Le Scienze, 206. 33100 UDINE – ITALY
{gentilin, piazza, policrit}@dimi.uniud.it

Abstract

We encode the simulation problem into a membership/inclusion problem over sets. This allows us to exploit the notion of rank and to use constraint solving procedures to incrementally compute the maximum simulation.

1 Introduction

In this paper we deal with the problem of algorithmically determining the so-called *simulation* relation on a labeled graph (Kripke structure). The simulation relation is one of the most important tools used in the formal (automatic) verification of temporal properties, whenever such an activity is to be performed on prohibitively large Kripke structure (cf. [9, 3]). In practice, the quotient of the input Kripke structure with respect to the equivalence relation associated to the simulation relation is used in order to (often significantly) reduce the size set states'space.

Several polynomial-time algorithms for computing similarity quotients have been proposed: the ones presented in [1], [4], and [5] achieve time complexities of the orders $O(|N|^6|E|)$, $O(|N|^4|E|)$, and $O(|E|^2)$, respectively. The best time complexity for algorithms computing the maximal simulation is the one presented in [9] and [2] ($O(|N||E|)$). An (slow) algorithm improving the space complexity of the problem (an extremely significant parameter in the context of verification) is [3]. In [8] an algorithm faster than [3] and improving on the space complexity of [9] is proposed.

Our approach to the problem of simulation computation here has two aims: on the one hand we want to provide a framework in which developing algorithms competitive with the best ones in the literature, on the other hand we want to show how it is extremely natural to view such algorithms as constraint solving procedures to be easily implemented in a constraint logic programming scheme.

We consider the problem from an original point of view, namely we encode the simulation problem into a membership/inclusion problem over sets. In this way we not only re-formulate the problem, but we also naturally get to a procedure that works on a *layered* version of the input structure. The layering of the structure is determined using a—somehow yet unsatisfactory—notion of rank (ultimately suggested by the set-theoretic encoding); the use of constraint solvers over sets in order to compute the solutions allows then an *incremental* simplified computation (cf. Sections 4 and 5).

The circle of ideas presented was already used in [6] and [10] for *bisimulation* computation, allowing to obtain an algorithm whose complexity is in some case linear and in general matches the best in the literature for that problem. Our prospective task in the context of simulation is to propose optimizations such as those proposed for bisimulation. A major stumbling block to this end is the existence of a suitable notion of rank (cf. Section 4).

The paper is organized as follows: in Section 2 we introduce the simulation problem; in Section 3 we present our set theoretical encoding of the problem; the notion of rank which allows to incrementally compute the encoding is defined in Section 4, while the formalization of the incremental computation using $CLP(\mathcal{SET})$ and $\{\log\}$ is described in Section 5; in Section 6 some complexity issues are considered; in Section 7 we briefly discuss the symbolic representation of our encoding; some conclusions are drawn in Section 8.

2 The Simulation Problem

Given a graph with labels on the nodes the *simulation problem* consists in finding an equivalence relation between the nodes of the graph which is the maximal that satisfies certain conditions.

Definition 1 (Labelled Graphs) *Let Σ be a finite alphabet. A* labeled graph *is a triple $G = \langle N, E, \ell\rangle$, where $\langle N, E\rangle$ is a direct graph, and $\ell : N \to \Sigma$ is a labeling function.*

In this paper we assume that the nodes in G are an initial segment of the natural numbers, i.e. $N = \{1, \ldots, k\}$.

Definition 2 (Simulation) *Let $G = \langle N, E, \ell\rangle$ be a labeled graph. A relation $\leq \subseteq N \times N$ is said to be a* simulation *over G iff:*

(label) $n \leq m \Rightarrow \ell(n) = \ell(m)$*;*

(forw) $n \leq m \wedge nEn' \Rightarrow \exists m'(mEm' \wedge n' \leq m')$.

Lemma 3 *Given a graph G there exists a maximum simulation $\leq_s$ over G.*

Definition 4 (Maximum Simulation Equivalence) *Given a graph G, the* maximum simulation equivalence $\equiv_s$ *is defined as*

$$n \equiv_s m \Leftrightarrow n \leq_s m \wedge m \leq_s n.$$

Lemma 5 *Given a graph G the maximum simulation equivalence $\equiv_s$ over G is an equivalence relation.*

Given a graph G, we consider the problem of computing $\equiv_s$. All of the algorithms mentioned in the introduction ([1], [4], [5], [2], [9]) obtain the similarity quotient as a by-product of the computation of the similarity relation on the set of states N. Their space complexity is then limited from below by $O(|N|^2)$.

Recently Bustan and Grumberg proposed an algorithm which gives in output the quotient structure with respect to $\equiv_s$ and the simulation relation among the classes of $M = N/_{\equiv_s}$, without computing the entire simulation on N. The space requirements of this algorithm (often most critical, especially in the field of verification) depends on the size of M and is lower than the ones of the algorithms in [1], [4], [5], [2], and [9]. In more detail, the so-called *Partitioning Algorithm*, described in [3], uses only $|M|^2 + |N|\log(|M|)$ space. The time complexity of the *Partitioning Algorithm* is instead rather heavy: it is $O(|M|^4(|E| + |M|^2) + |M|^2|N|(|N| + |M|^2))$.

The algorithm in [8] combines ideas from [9] and [3], together with suitable data structures representing intermediate values for the reduced labeled graph, to obtain $\equiv_s$ in time $O(|M|^2|E|)$ and space $O(|M|^2 \log |M| + |N| \log |M|)$.

3 Encoding the Simulation Problem by sets

In [10] the authors show how the Bisimulation Problem can be naturally seen as the problem of establishing equality between sets under the so-called *anti-foundation* axioms. This is done thinking of nodes in the Kripke structure as sets and agreeing that an arc from m to n corresponds to the membership of the node $A(n)$ associated to n, to the node $A(m)$ associated to m:

$$mEn \quad \text{if and only if} \quad A(n) \in A(m).$$

Under these assumptions bisimilar nodes are exactly those that have the same associated sets.

The set-theoretic point of view allows, then, to exploit natural set-theoretic notions (as the notion of rank of a set) to optimize the algorithmic task of reducing the Kripke structure by eliminating bisimilar nodes.

If we try to extend the above outlined approach to the task of reducing a Kripke structure by eliminating *similar* nodes, a natural idea would be, again, to associate sets to nodes and represent arcs using the membership. The problem with this approach is the following: a moment's thought shows that it can be the case that $m \geq_s n$ without having the set associated to m including the set associated to n[1]:

$$m \geq_s n \quad \not\Rightarrow \quad A(m) \supseteq A(n).$$

Hence, we can easily build examples in which $m \geq_s n$ and $n \geq_s m$ but either $A(m) \not\supseteq A(n)$ or $A(n) \not\supseteq A(m)$. This implies that the fact that $m \equiv_s n$ cannot be checked considering equality between the associated sets.

[1] e.g. m' is the only child of m and both m' and n' with $m' \geq_s n'$ are the children of n

The basic idea behind our set-encoding for simulation, is to assign *two* sets $a(n)$ and $A(n)$ to each node n and to use the membership relation to represent $\leq_s$ as follows:

$$a(n) \in A(m) \quad \text{if and only if} \quad n \leq_s m.$$

The A-sets can now be used to establish similarity between nodes, however, since $A(m)$ is the collection of the a-encoding relative to nodes the simulated by m, it does not necessarily correspond to the collection of encoding of children of m and the above mentioned consideration does not apply.

More specifically, recalling that by our convention a node $n \in N$ is a natural number, we assign to n the set:

$$a(n) \quad = \quad \{\emptyset\}^n \quad = \quad \underline{n},$$

where $\{\emptyset\}^1 = \{\emptyset\}$, $\{\emptyset\}^2 = \{\{\emptyset\}\}$, and so on.

We determine the A-encoding as solution of a collection of (set-theoretic) constraints keeping into account the above idea as well as the initial labeling on G. Consider, for each node $n \in N$, a set variable $\mathtt{A_n}$ and the constraint $C(n)$ defined as follows:

$$\begin{aligned} C(n) \quad = \quad & \{a(n)\} \subseteq \mathtt{A_n} \subseteq \{a(m) \mid m \in N \wedge \ell(m) = \ell(n)\} \wedge \\ & \bigwedge_{\ell(m)=\ell(n)} (a(m) \in \mathtt{A_n} \leftrightarrow \{a(r) \mid mEr\} \subseteq \bigcup_{nEs} \mathtt{A_s}). \end{aligned}$$

Finally, let Sys_G be the constraint obtained as conjunction of all the constraint $C(n)$

$$Sys_G \quad = \quad \bigwedge_{n \in N} C(n).$$

We prove the correctness of our approach by showing that:

1. each solution Sol of Sys_G determines a simulation $\leq_{Sol}$;
2. the maximal simulation relation $\leq_s$ is determined by a solution SM of Sys_G, i.e $\leq_s \equiv \leq_{SM}$;
3. if $\leq_{Sol1} \subseteq \leq_{Sol2}$, then $Sol1 \sqsubseteq Sol2$, where $\sqsubseteq$ is a partial order naturally induced over the set of solutions of Sys_G (cf. Definition 8).

Lemma 6 *Let $G = \langle N, E, \ell \rangle$ be a graph and Sys_G be the system of constraint obtained from G as described above. Consider a solution $Sol = \{A(n) \mid n \in N\}$ of Sys_G and let*

$$n \leq_{Sol} m \quad \leftrightarrow_{\mathtt{def}} \quad a(n) \in A(m).$$

The relation $\leq_{Sol}$ is a simulation over G.

The following lemma is a sort of converse of the above and shows that the maximum simulation $\leq_s$ over G determines a solution of Sys_G.

Lemma 7 *Let $G = \langle N, E, \ell \rangle$ be a graph and $\leq_s$ be the maximum simulation relation over G. Let*

$$SM \quad = \quad \{A(n) = \{a(m) \mid m \leq_s n\} \mid n \in N\}.$$

SM is a solution of Sys_G. Moreover,

$$\leq_{SM} = \leq_s,$$

where $\leq_{SM}$ is the simulation defined from SM as in Lemma 6.

The following definition introduces an ordering among the A-encoding (i.e. the solutions of the system Sys_G defined above) that will turn out to be the key to choose the encoding associated to the maximal simulation.

Definition 8 *Let $G = \langle N, E, \ell \rangle$ be a graph and $\mathcal{S}$ be the set of solutions of Sys_G. Consider $Sol_1 = \{A_1(n) \mid n \in N\}$ and $Sol_2 = \{A_2(n) \mid n \in N\}$ two elements of $\mathcal{S}$, we define the partial order $\sqsubseteq$ over $\mathcal{S}$ as follows:*

$$Sol_1 \sqsubseteq Sol_2 \quad \leftrightarrow \quad \forall n \in N(A_1(n) \subseteq A_2(n))).$$

The following lemma shows that the inclusion relation among simulation relations corresponds to the above ordering between A-encoding, thereby ensuring that the maximal A-encoding does correspond to the maximal simulation relation.

Lemma 9 *If $\leq_{Sol_1} \subseteq \leq_{Sol_2}$, then $Sol_1 \sqsubseteq Sol_2$.*

Theorem 10 *Let G be a graph. There exists a $\sqsubseteq$-maximum solution of Sys_G. Moreover, if SM is the $\sqsubseteq$-maximum solution of Sys_G, then $\leq_{SM}$ is the maximum simulation over G.*

Example 11 *From the graph G in Figure 1 we obtain:*

$$\begin{array}{lll} a(1) = \{\emptyset\}^1 = \underline{1} & a(2) = \{\emptyset\}^2 = \underline{2} & a(3) = \{\emptyset\}^3 = \underline{3} \\ a(4) = \{\emptyset\}^4 = \underline{4} & a(5) = \{\emptyset\}^5 = \underline{5} & a(6) = \{\emptyset\}^6 = \underline{6} \end{array}$$

$$\begin{array}{c} \{a(1)\} \subseteq \mathtt{A}_1 \subseteq \{a(1), a(3), a(5)\} \wedge (a(1) \in \mathtt{A}_1 \leftrightarrow \emptyset \subseteq \emptyset) \wedge \\ (a(3) \in \mathtt{A}_1 \leftrightarrow \{a(2), a(4)\} \subseteq \emptyset) \\ \dots \\ \{a(6)\} \subseteq \mathtt{A}_6 \subseteq \{a(2), a(4), a(6)\} \wedge \\ (a(2) \in \mathtt{A}_6 \leftrightarrow \{a(1)\} \subseteq \mathtt{A}_3) \wedge \\ (a(4) \in \mathtt{A}_6 \leftrightarrow \{a(5)\} \subseteq \mathtt{A}_3) \wedge \\ (a(6) \in \mathtt{A}_6 \leftrightarrow \{a(3)\} \subseteq \mathtt{A}_3) \end{array}$$

The maximum solution of Sys_G is

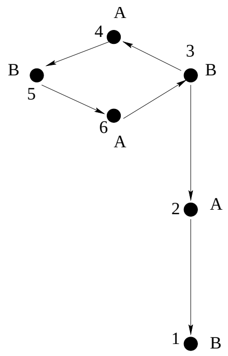

Figure 1: Example.

$$\begin{aligned} A(1) &= \{a(1)\} \\ A(2) &= \{a(2)\} \\ A(3) &= A(5) = \{a(1), a(3), a(5)\} \\ A(4) &= A(6) = \{a(4), a(6)\}. \end{aligned}$$

We conclude this section with a remark that will be our starting point, in the next section, in trying to optimize the work we have done up to this point with a rank-like argument of the sort of those used in [6, 10] in the case of bisimulation.

Remark 12 *The pair of codes associated to the nodes using the above outlined technique is, in general, redundant in the following sense: different nodes have always different a-encoding, while similar nodes, for example, could obviously share the same a-encoding. Clearly, a simplification (decrease of the a's) based on the knowledge of the simulation relation would not make sense since it is possible only* a-posteriori. *However, as we will see in the next section, an elimination of redundancies becomes viable when performed after assigning a rank to the nodes.*

A nice consequence we obtain by removing the redundancy in the a-encoding is that:

$$\begin{aligned} a(n) = a(m) \quad &\Leftrightarrow \quad n \equiv_s m \\ A(n) \subseteq A(m) \quad &\Leftrightarrow \quad n \leq_s m \end{aligned}$$

i.e. the a-encoding represents the relation $\equiv_s$, while the A-encoding represents the relation $\leq_s$.

4 Determining the encoding by ranks

In order to make use of the possibility of reducing the number of different a's using the information conveyed by the A's, in this section we describe a strategy based on a *layering* of the input graph roughly corresponding to the layering induced by the notion of *rank* among sets.

The possible presence of cycles in the graph naturally calls for the introduction of the following, preliminary, definition. For simplicity we introduce the notion below in the case on un-labeled graphs, the extension to the labeled case is obvious.

Definition 13 (Strongly Connected Components) *Given $G = \langle N, E\rangle$, let $G^{scc} = \langle N^{scc}, E^{scc}\rangle$ be the graph obtained as follows:*

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

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

Using G^{scc} we can easily define a notion of rank suitable for our purposes:

Definition 14 (Rank) *Let $G = \langle N, E\rangle$. The* rank *of a node n of G is defined as:*

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

On the ground of the above definition the computation of the a's and the A's can now be carried out rank by rank as in the following example.

Example 15 *Consider the graph in Figure 2*

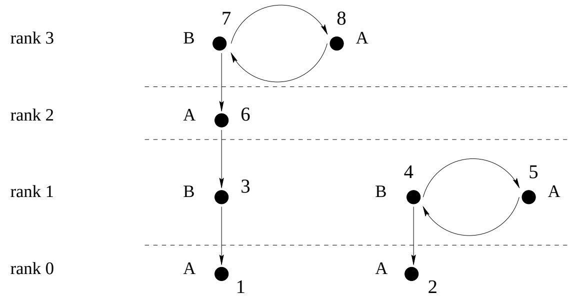

Figure 2: Example.

First considering only the nodes at rank 0 we obtain

$$a(1) = \underline{1} \quad a(2) = \underline{2}$$

and

$$\{a(1)\} \subseteq \mathtt{A}_1 \subseteq \{a(1), a(2)\} \wedge (a(1) \in \mathtt{A}_1 \leftrightarrow \emptyset \subseteq \emptyset) \wedge (a(2) \in \mathtt{A}_1 \leftrightarrow \emptyset \subseteq \emptyset)\wedge$$
$$\{a(1)\} \subseteq \mathtt{A}_2 \subseteq \{a(1), a(2)\} \wedge (a(1) \in \mathtt{A}_2 \leftrightarrow \emptyset \subseteq \emptyset) \wedge (a(2) \in \mathtt{A}_2 \leftrightarrow \emptyset \subseteq \emptyset)$$

whose maximum solution is

$$A(1) = A(2) = \{a(1), a(2)\}.$$

This means that 1 *and* 2 *are similar, hence we can re-assign to them the same* a*-encoding:*

$$a(1) = a(2) = \underline{1}.$$

As second step we consider the nodes at rank 1 *that must be considered together with the nodes at rank* 0*. The following are the corresponding values of the* a*'s:*

$$a(1) = a(2) = \underline{1} \quad a(3) = \underline{2} \quad a(4) = \underline{3} \quad a(5) = \underline{4}$$

from which we obtain the constraint

$$\{a(1)\} \subseteq \mathtt{A}_1 \subseteq \{a(1), a(5)\} \wedge (a(5) \in \mathtt{A}_1 \leftrightarrow \{a(4)\} \subseteq \emptyset) \wedge \ldots$$

whose solution is

$$\begin{array}{ll} A(1) = A(2) = \{a(1)\} & A(3) = \{a(3)\} \\ A(4) = \{a(3), a(4)\} & A(5) = \{a(5), a(1)\}. \end{array}$$

Since there are no two nodes n, m *such that* $a(n) \neq a(m)$ *and* $A(n) = A(m)$*, we do not have to re-assign the* a*-encodings.*

In the third step we consider all the nodes at ranks $0, 1, 2$*:*

$$a(1) = a(2) = \underline{1} \quad a(3) = \underline{2} \quad a(4) = \underline{3} \quad a(5) = \underline{4} \quad a(6) = \underline{5},$$

and

$$\{a(1)\} \subseteq \mathtt{A}_1 \subseteq \{a(1), a(6)\} \wedge (a(6) \in \mathtt{A}_1 \leftrightarrow \{a(3)\} \subseteq \emptyset) \wedge \ldots$$

whose solution is

$$\begin{array}{lll} A(1) = A(2) = \{a(1)\} & A(3) = \{a(3)\} & A(4) = \{a(3), a(4)\} \\ A(5) = \{a(5), a(1), a(6)\} & A(6) = \{a(6), a(1)\}. & \end{array}$$

Notice that here we have added $a(6)$ *to* $A(5)$*.*

Similarly during the fourth step we consider all the nodes at ranks $0, 1, 2, 3$*:*

$$\begin{array}{llll} a(1) = a(2) = \underline{1} & a(3) = \underline{2} & a(4) = \underline{3} & \\ a(5) = \underline{4} & a(6) = \underline{5} & a(7) = \underline{6} & a(8) = \underline{7}, \end{array}$$

and

$$\ldots \wedge (a(7) \in \mathtt{A}_3 \leftrightarrow \{a(6), a(8)\} \subseteq \mathtt{A}_1) \wedge \ldots$$

whose solution is

$$\begin{array}{ll} A(1) = A(2) = \{a(1)\} & A(3) = \{a(3)\} \\ A(4) = A(7) = \{a(3), a(4), a(7)\} & \\ A(5) = A(8) = \{a(5), a(1), a(6), a(8)\} & A(6) = \{a(6), a(1)\}. \end{array}$$

Hence, we can re-assign the a-encoding as follows:

$$a(1) = a(2) = \underline{1} \quad a(3) = \underline{2} \quad a(4) = a(7) = \underline{3} \quad a(5) = a(8) = \underline{4} \quad a(6) = \underline{5},$$

and obtain

$$\begin{array}{ll} A(1) = A(2) = \{\underline{1}\} & A(3) = \{\underline{2}\} \\ A(4) = A(7) = \{\underline{2}, \underline{3}\} & \\ A(5) = A(8) = \{\underline{4}, \underline{1}, \underline{5}\} & A(6) = \{\underline{5}, \underline{1}\}. \end{array}$$

As illustrated in the above example, the computation proceeds by computing the a's and the A's (with the technique outlined in Section 3) and then by simplifying the choices for the a's using the information conveyed by the A's. This simplification turns out to be useful in subsequent steps (greater ranks) of the computation. The fact that the A-values computed must be, in general, re-calculated in subsequent steps, is a consequence of the fact that, again in general, there are similar nodes at different ranks (see, for example, nodes 5 and 8 in Figure 2).

An important feature of the technique presented above, is that even though the A-values must be re-calculated at each step, their value can only increase. In fact, the A-values to be modified are those that must be updated by adding a-values of nodes at the current rank which turn out to be smaller w.r.t. $\leq_s$ (e.g., while processing nodes at rank 2, since $6 \leq_s 5$ the set $A(5)$ must get $a(6)$).

An algorithm based on the ideas presented in this section and naturally implementing the above incremental strategy is described in the next section.

We conclude this section by observing that the reader may consider the need of taking into account at each step the whole set of already determined A-values, a *weakness of our proposed notion of rank*. This is true and, as a matter of fact, to date we do not have a notion of rank allowing us to proceed in the computation of the encoding at rank k ignoring all the encoding of nodes at rank less than k. With respect to this problem we observe that:

1. a *good* notion of rank for the computation of the *bisimulation* relation (but not working for *simulation*) is available (cf. [6]);

2. any proposed notion of rank to be used in the computation of the *simulation* relation must be sensible to the labeling: if all the nodes have the same label, two nodes are similar if and only if they have the same rank as defined in [6].

5 Incremental computation of the simulation relation

Given a graph G, it is extremely natural to use a constraint solver like the interpreter {log}, which is based on the constraint logic programming language $CLP(\mathcal{SET})$ (see [7]), in order to determine the $\sqsubseteq$-maximum solution of Sys_G. In particular, the version of the interpreter {log} we refer to in this paper is available at `http://www.dimi.uniud.it/~piazza/LOG` and can be *consulted* using

SICStus Prolog. We show in this section how to use {log} in the incremental computation of the solution of Sys_G.

Assuming that at step i we have considered the nodes at ranks $0, 1, 2, \ldots, i-1$ obtaining the partial solution

$$a(1) = \underline{1} \quad \ldots \quad a(k) = \underline{k'}$$

$$A(1) = \{a(1), \ldots\} \quad \ldots \quad A(k) = \{a(k), \ldots, \},$$

with

$$a(h) = a(j) \Leftarrow A(h) = A(j)$$

(i.e. we have re-assigned the a-encodings) and

$$max\{a(1), \ldots, a(k)\} = \underline{m},$$

then at step $i+1$ we have to:

- consider all the nodes at rank $0, 1, 2, \ldots, i$;
- assign to the (say) s nodes at rank i an a-encoding

  $$a(k+1) = m+1 \quad \ldots \quad a(k+s) = m+s;$$

- if h is a node at rank $0, 1, 2, \ldots, i-1$, then consider the constraint

  $$A(h) \subseteq \mathtt{A_h} \subseteq A(h) \cup \{a(j) \mid rank(j) = i+1 \wedge \ell(j) = \ell(h)\} \wedge$$

  $$\bigwedge_{rank(j)=i+1 \wedge \ell(j)=\ell(h)} (a(j) \in \mathtt{A_h} \leftrightarrow \{a(t) \mid jEt\} \subseteq \bigcup_{hEv} \mathtt{A_v});$$

- if h is a node at rank i, then consider the constraint

  $$\{a(h)\} \subseteq \mathtt{A_h} \subseteq \{a(j) \mid rank(j) \leq i+1 \wedge \ell(j) = \ell(h)\} \wedge$$

  $$\bigwedge_{rank(j) \leq i+1 \wedge \ell(j)=\ell(h)} (a(j) \in \mathtt{A_h} \leftrightarrow \{a(t) \mid jEt\} \subseteq \bigcup_{hEv} \mathtt{A_v});$$

- if

  $$A(1) = \{a(1), \ldots\} \quad \ldots \quad A(k+s) = \{a(k+s), \ldots\}$$

  is the $\sqsubseteq$-maximum solution of the conjunction of these constraints, and

  $$A(h) = A(j) \wedge a(h) < a(j),$$

  then re-assign to j the a-encoding $a(h)$, i.e. put

  $$a(j) = a(h)$$

  and replace it also in the solution;

- repeat the last step until it is possible to apply it.

As already explained in Section 4, the computation is incremental in the sense that once we put $a(j)$ in $A(h)$ we never remove it. As a matter of fact, during the step $i+1$, relatively to the nodes h of rank less than i, we only try to add to $A(h)$ the a-encodings of the nodes of rank i. The fact that the computation is incremental is a consequence of the fact that if j is a node at rank i, then all the nodes reachable from j have rank at most i, and these are the only nodes that may cause $a(j)$ to be added to $A(h)$. Moreover, relatively to the nodes h at rank i we start by putting in $A(h)$ at least $a(h)$, since it is always true that a node simulates itself.

Example 16 *Consider the graph in Figure 3.*

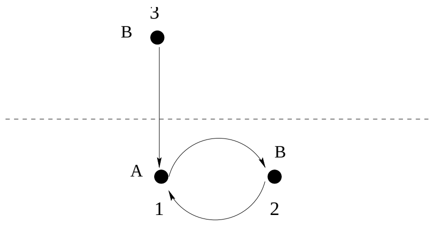

Figure 3: Example.

During the first step we consider the nodes at rank 0. *We put*

$$a(1) = \underline{1} \quad a(2) = \underline{2}$$

and we consider the constraint

$$\{\underline{1}\} \subseteq \mathtt{A}_1 \subseteq \{\underline{1}\} \wedge \{\underline{2}\} \subseteq \mathtt{A}_2 \subseteq \{\underline{2}\}$$

whose maximum solution is

$$A(1) = \{\underline{1}\} \quad A(2) = \{\underline{2}\}.$$

During the second step we consider the nodes at rank 0 *and* 1. *We put*

$$a(1) = \underline{1} \quad a(2) = \underline{2} \quad a(3) = \underline{3}$$

and we consider the constraint

$$\begin{array}{c} \{\underline{1}\} \subseteq \mathtt{A}_1 \subseteq \{\underline{1}\} \wedge \\ \{\underline{2}\} \subseteq \mathtt{A}_2 \subseteq \{\underline{2},\underline{3}\} \wedge (\underline{3} \in \mathtt{A}_2 \leftrightarrow \{\underline{1}\} \subseteq \mathtt{A}_1) \wedge \\ \{\underline{3}\} \subseteq \mathtt{A}_3 \subseteq \{\underline{3},\underline{2}\} \wedge (\underline{2} \in \mathtt{A}_3 \leftrightarrow \{\underline{1}\} \subseteq \mathtt{A}_1) \end{array}$$

whose maximum solution is

$$A(1) = \{\underline{1}\} \quad A(2) = A(3) = \{\underline{2},\underline{3}\}$$

Since $A(2) = A(3)$ *and* $a(2) < a(3)$ *we put* $a(3) = \underline{2}$ *obtaining*

$$a(1) = \underline{1} \quad a(2) = a(3) = \underline{2}$$

$$A(1) = \{\underline{1}\} \quad A(2) = A(3) = \{\underline{2}\}.$$

Below we discuss the (limited) fragment of $CLP(\mathcal{SET})$ needed to solve our constraints.

A $CLP(\mathcal{SET})$ *-term* is a term of the following language

$$t ::= \{\} \mid n \mid X \mid \{t_1 \mid t_2\},$$

where $\{\}$ is the empty-set, $n \in \mathbb{N}$, X is a variable (from now on all capital letters will represent variables), $\{t_1 \,|\, t_2\}$ stands for $\{t_1\} \cup t_2$ (i.e. the elements of $\{t_1 \,|\, t_2\}$ are t_1 and all the elements of t_2). Moreover, it is possible to write the term $\{t_1 \mid \{t_2 \mid \ldots \{t_n \mid \{\}\} \ldots\}\}$ as

$$\{t_1, t_2, \ldots, t_n\},$$

and $t_1, t_2, \ldots, t_n$ are all its elements.

A $CLP(\mathcal{SET})$ *-constraint* is composed by conjunctions (written &) and disjunctions (written `or`) of literals of the form[2]

$$\begin{gathered} t_1 = t_2 \quad t_1 \,\mathtt{neq}\, t_2 \quad t_1 \,\mathtt{in}\, t_2 \qquad t_1 \,\mathtt{nin}\, t_2 \\ \mathtt{un}(t_1, t_2, t_3) \quad \mathtt{nun}(t_1, t_2, t_3) \\ \mathtt{sub}(t_1, t_2) \quad \mathtt{nsub}(t_1, t_2) \quad \mathtt{betw}(t_1, t_2, t_3) \end{gathered}$$

whose semantics can be intuitively described as follows

$$\begin{array}{lcl} t_1 = t_2 & \Leftrightarrow & \forall t_3 (t_3 \in t_1 \leftrightarrow t_3 \in t_2) \\ t_1 \,\mathtt{neq}\, t_2 & \Leftrightarrow & t_1 \neq t_2 \\ t_1 \,\mathtt{in}\, t_2 & \Leftrightarrow & t_1 \in t_2 \\ t_1 \,\mathtt{nin}\, t_2 & \Leftrightarrow & t_1 \notin t_2 \\ \mathtt{un}(t_1, t_2, t_3) & \Leftrightarrow & t_1 \cup t_2 = t_3 \\ \mathtt{nun}(t_1, t_2, t_3) & \Leftrightarrow & t_1 \cup t_2 \neq t_3 \\ \mathtt{sub}(t_1, t_2) & \Leftrightarrow & t_1 \subseteq t_2 \\ \mathtt{nsub}(t_1, t_2) & \Leftrightarrow & t_1 \not\subseteq t_2 \\ \mathtt{betw}(t_1, t_2, t_3) & \Leftrightarrow & t_1 \subseteq t_2 \subseteq t_3 \end{array}$$

Example 17 *The formula*

$$\underline{1} \in \mathtt{A}_2 \leftrightarrow \emptyset \subseteq \emptyset$$

can be written as $CLP(\mathcal{SET})$ *constraint using the following syntax*

$$(1 \,\mathtt{in}\, A_2 \,\&\, \mathtt{sub}(\{\}, \{\})) \;\mathtt{or}\; (1 \,\mathtt{nin}\, A_2 \,\&\, \mathtt{nsub}(\{\}, \{\})).$$

[2]In the standard version of $CLP(\mathcal{SET})$presented in [7] and in the standard interpreter {log} available at `http://prmat.math.unipr.it/~gianfr/setlog.Home.html` there are no predicates `sub, nsub` and `betw`, which have easily been added in the version of the interpreter used here: they are only shorthand for standard $CLP(\mathcal{SET})$ -constraints.

The interpreter {log} gives us all the solutions of the $CLP(\mathcal{SET})$ *-constraint. In this case the collection of solutions is given parametrically as*

$$\mathtt{A}_2 = \{1 \mid \mathtt{N}\}$$

which means that 1 *is an element of* $\mathtt{A}_2$ *and in* $\mathtt{A}_2$ *there can be arbitrarily many other elements represented by (the set variable)* $\mathtt{N}$.

Now we show an example of the incremental computation of the simulation using $CLP(\mathcal{SET})$ -constraint and {log}.

Example 18 *Consider the graph in Figure 4.*

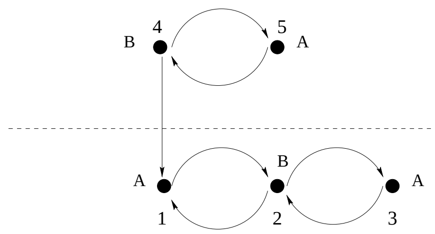

Figure 4: Example.

First we have to consider all the nodes at rank 0. *The a-encoding is*

$$a(1) = \underline{1} \quad a(2) = \underline{2} \quad a(3) = \underline{3}$$

and the corresponding constraint is

$$\begin{gathered}\{\underline{1}\} \subseteq \mathtt{A}_1 \subseteq \{\underline{1}, \underline{3}\} \wedge (\underline{3} \in \mathtt{A}_1 \leftrightarrow \{\underline{2}\} \subseteq \mathtt{A}_2) \wedge \\ \{\underline{2}\} \subseteq \mathtt{A}_2 \subseteq \{\underline{2}\} \wedge \\ \{\underline{3}\} \subseteq \mathtt{A}_3 \subseteq \{\underline{1}, \underline{3}\} \wedge (\underline{1} \in \mathtt{A}_3 \leftrightarrow \{\underline{2}\} \subseteq \mathtt{A}_2),\end{gathered}$$

which written as $CLP(\mathcal{SET})$ *-constraint is*

$$\begin{gathered}\mathtt{betw}(\{1\}, \mathtt{A}_1, \{1,3\}) \,\&\, ((3 \,\mathtt{in}\, \mathtt{A}_1 \,\&\, \mathtt{sub}(\{2\}, \mathtt{A}_2)) \;\mathtt{or}\; (3 \,\mathtt{nin}\, \mathtt{A}_1 \,\&\, \mathtt{nsub}(\{2\}, \mathtt{A}_2))) \,\&\\ \mathtt{betw}(\{2\}, \mathtt{A}_2, \{2\}) \,\&\\ \mathtt{betw}(\{3\}, \mathtt{A}_3, \{1,3\}) \,\&\, ((1 \,\mathtt{in}\, \mathtt{A}_3 \,\&\, \mathtt{sub}(\{2\}, \mathtt{A}_2)) \;\mathtt{or}\; (1 \,\mathtt{nin}\, \mathtt{A}_3 \,\&\, \mathtt{nsub}(\{2\}, \mathtt{A}_2))).\end{gathered}$$

In this case {log} gives us only the solution

$$\begin{aligned}\mathtt{A}_1 &= \{1,3\},\\ \mathtt{A}_2 &= \{2\},\\ \mathtt{A}_3 &= \{3,1\}.\end{aligned}$$

Since $\mathtt{A}_1 = \mathtt{A}_3$ *and* $a(1) < a(3)$ *we re-assign*

$$a(3) = a(1) = \underline{1},$$

obtaining

$$a(1) = a(3) = \underline{1} \quad a(2) = \underline{2}$$

$$A(1) = A(3) = \{\underline{1}\} \quad A(2) = \{\underline{2}\}.$$

At the second step we start with the a-encoding

$$a(1) = a(3) = \underline{1} \quad a(2) = \underline{2} \quad a(4) = \underline{3} \quad a(5) = \underline{4},$$

and the constraint

$$\begin{gathered}
\{\underline{1}\} \subseteq \mathtt{A}_1 \subseteq \{\underline{1}, \underline{4}\} \wedge (\underline{4} \in \mathtt{A}_1 \leftrightarrow \{\underline{3}\} \subseteq \mathtt{A}_2) \wedge \\
\{\underline{2}\} \subseteq \mathtt{A}_2 \subseteq \{\underline{2}, \underline{3}\} \wedge (\underline{3} \in \mathtt{A}_2 \leftrightarrow \{\underline{4}, \underline{1}\} \subseteq \mathtt{A}_1) \wedge \\
\{\underline{3}\} \subseteq \mathtt{A}_3 \subseteq \{\underline{3}, \underline{2}\} \wedge (\underline{2} \in \mathtt{A}_3 \leftrightarrow \{\underline{1}\} \subseteq \mathtt{A}_1 \cup \mathtt{A}_4) \wedge \\
\{\underline{4}\} \subseteq \mathtt{A}_4 \subseteq \{\underline{4}, \underline{1}\} \wedge (\underline{1} \in \mathtt{A}_4 \leftrightarrow \{\underline{2}\} \subseteq \mathtt{A}_3),
\end{gathered}$$

which written as $CLP(\mathcal{SET})$ *-constraint becomes*

$$\begin{gathered}
\mathtt{betw}(\{1\}, \mathtt{A}_1, \{1,4\})\ \&\ ((4\ \mathtt{in}\ \mathtt{A}_1\ \&\ \mathtt{sub}(\{3\}, \mathtt{A}_2))\ \mathtt{or}\ (4\ \mathtt{nin}\ \mathtt{A}_1\ \&\ \mathtt{nsub}(\{3\}, \mathtt{A}_2)))\ \& \\
\mathtt{betw}(\{2\}, \mathtt{A}_2, \{2,3\})\ \& \\
((3\ \mathtt{in}\ \mathtt{A}_2\ \&\ \mathtt{sub}(\{4,1\}, \mathtt{A}_1))\ \mathtt{or}\ (3\ \mathtt{nin}\ \mathtt{A}_2\ \&\ \mathtt{nsub}(\{4,1\}, \mathtt{A}_1)))\ \& \\
\mathtt{betw}(\{3\}, \mathtt{A}_3, \{3,2\})\ \&\ \mathtt{un}(\mathtt{A}_1, \mathtt{A}_4, \mathtt{X})\ \& \\
((2\ \mathtt{in}\ \mathtt{A}_3\ \&\ \mathtt{sub}(\{1\}, \mathtt{X}))\ \mathtt{or}\ (2\ \mathtt{nin}\ \mathtt{A}_3\ \&\ \mathtt{nsub}(\{1\}, \mathtt{X})))\ \& \\
\mathtt{betw}(\{4\}, \mathtt{A}_4, \{4,1\})\ \&\ ((1\ \mathtt{in}\ \mathtt{A}_4\ \&\ \mathtt{sub}(\{2\}, \mathtt{A}_3))\ \mathtt{or}\ (1\ \mathtt{nin}\ \mathtt{A}_4\ \&\ \mathtt{nsub}(\{2\}, \mathtt{A}_3))).
\end{gathered}$$

In this case $\{log\}$ *gives us the two solutions*

$$\begin{aligned}
\mathtt{A}_1 &= \{1\}, \\
\mathtt{A}_2 &= \{2\}, \\
\mathtt{A}_3 &= \{3,2\}, \\
\mathtt{A}_4 &= \{4,1\}
\end{aligned}$$

$$\begin{aligned}
\mathtt{A}_1 &= \{1,4\}, \\
\mathtt{A}_2 &= \{2,3\}, \\
\mathtt{A}_3 &= \{3,2\}, \\
\mathtt{A}_4 &= \{4,1\}
\end{aligned}$$

The second solution is the maximum one and using it to re-assign the a-encoding we obtain

$$a(1) = a(3) = a(5) = \underline{1} \quad a(2) = a(4) = \underline{2},$$

$$A(1) = A(3) = A(5) = \{\underline{1}\} \quad A(2) = A(4) = \{\underline{2}\}.$$

Notice that, even though the A's are singletons here, this is not true in general (cf. Examples 11 *and* 15*).*

6 Complexity Issues

In this section we analyse the worst case complexity of the constraint problem encoding the simulation problem assuming that all the nodes in the graph are at the same rank.

First of all we recall from [7] that it is possible to encode NP-complete problems in $CLP(\mathcal{SET})$ -constraints: given an instance of the 3-SAT problem in k propositional variables it is possible to write a $CLP(\mathcal{SET})$ -constraint in k variables which has a solution if and only if the answer to the instance of 3-SAT is positive. This implies that, in general, the interpreter {log} can take more than polynomial time to produce a solution for a constraint.

Given a graph $G = \langle N, E, \ell \rangle$ we obtain a constraint Sys_G in $|N|$ variables (the variables $\mathtt{A_n}$'s). In the case of the constraints of the form Sys_G the interpreter {log}, as it is currently implemented, uses a sort of *generate and test* strategy: it starts initializing all the $\mathtt{A_n}$'s to N, it tests if all the literals in Sys_G are satisfied, if this is not the case it removes one element from one of the $\mathtt{A_n}$'s, and it tests if now all the literals in Sys_G are satisfied, and so on, until the solution is found. Since the number of possible assignments to the variables $\mathtt{A_n}$'s is $(2^{(|N|-1)})^{|N|}$, in the worst case we have a non-polynomial complexity.

However, it is possible to optimize the interpreter {log} for the constraints of the form Sys_G. Using counters and data-structures similar to the ones used in the algorithm proposed in [9], it would be possible to force {log} to solve the constraints of the form Sys_G in time $O(|N||E|)$, but this seems to be an unnatural solution which does not directly come from the set theoretical encoding we propose.

Let us analyse some possible optimizations suggested from the set encoding, which allow to design a constraint-solver for the constraints of the form Sys_G and which works in polynomial time. We can imagine that the constraint-solver works in this way:

- the $\mathtt{A_n}$'s are all initialized to N;
- each time that $Succ(m) = \{a(r) \mid mEr\} \not\subseteq \bigcup_{nEs} \mathtt{A_s} = \mathtt{B_n}$ it removes $a(m)$ from $\mathtt{A_n}$.

In the worst case the constraint-solver performs $|N|^2$ steps. The cost of one step for a single $C(n)$ is given by:

- the cost to compute $\mathtt{B_n}$;
- for each m the cost to perform the check $S(m) \subseteq \mathtt{B_n}$

Using lists to represent the $S(m)$'s and arrays of length $|N|$ for the $\mathtt{A_n}$'s and the $\mathtt{B_n}$'s we obtain that the total cost to find the maximal solution of Sys_G is $O(|N|^3|E|)$. Using lists without repetitions to represent the $S(m)$'s, the $\mathtt{A_n}$'s, and the $\mathtt{B_n}$'s we obtain that the cost is $O(|N|^4|E|)$.

7 Simulation reduction and symbolic computation

The simulation's set-encoding proposed in the previous sections can be naturally combined with the so-called symbolic representation of Kripke structures. In this section we will briefly suggest how to

- obtain OBDD representations of the a's and the A's associated to the nodes of the graph
- use the OBDD representation in order to refine $\{a(n) \mid n \in N\}$: the set of a-encoding assigned to the graph's vertexes.

We will use the notion of rank introduced in Definition 14 in order to get incrementally to the symbolic representation of our simulation's set-encoding.

Consider the computation associated to a fixed rank r and suppose having associated a variable $x_{a(n)}$ to each element of $\{a(n) \mid rank(n) \leq r\}$. We proceed assigning to each member of $\{n \mid rank(n) = r\}$ a distinct a-encoding, $a(n)$, together with a new variable $x_{a(n)}$. The A's relative to nodes having rank equal to r can be then represented by the OBDD of the boolean function

$$\bigwedge_{a(m) \in A(n)} x_{a(m)}$$

Such OBDD representation of the A-encoding turns out to be useful in the process of simplifying our choices for the a's: the a-encoding of two nodes m and n must be the same if and only if the OBDD's for m and n are equal. Once refined the set $\{a(n) \mid rank(n) = r\}$ we can finally give a more compact symbolic representation (i.e. associated to a boolean function with less variables) of the A's at rank r.

8 Conclusions and open problems

A novel method to compute the simulation relation, based on the encoding of the problem into a collection of set-theoretic constraint, is presented. The method allows for a natural and simple implementation in $CLP(\mathcal{SET})$ in which the solution of the constraint is incrementally determined.

As for the technique outlined in Section 5, it is easy to imagine that the whole incremental search for the final solution could be automated. To this end it would be necessary to integrate our procedure with an algorithm for the computation of the rank that at each step builds the constraint, passes it to {log} and, using the maximum solution, builds the constraint for the next step.

The main open problem we would like to mention, is the search for a notion of rank suitable for the computation of the simulation relation which does not need to keep into account, at rank k, the encoding of nodes at rank less than k. Provided such a notion of rank is not computationally heavy to determine, it would improve the heart of our algorithm, allowing us to significantly diminish the amount of data necessary at each step of the computation.

Finally, notice that the approach outlined in this paper is easily paired with a symbolic representation of the labeled structures and its subsets (cf. Section 7). In particular, it allows to combine the determination of the OBDD-representation *toghether* with the simulation reduction. This positive by-product was already obtained for the bisimulation relation with a similar approach in [10].

References

[1] B. Bloom. *Ready Simulation, Bisimulation, and the Semantics of CCS-like Languages*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, August 1989.

[2] Bard Bloom and Robert Paige. Transformational design and implementation of a new efficient solution to the ready simulation problem. *Science of Computer Programming*, 24(3):189–220, June 1995.

[3] D. Bustan and O. Grumberg. Simulation based minimization. In D.A. McAllester, editor, *Proc. 17th Int'l Conference on Automated Deduction (CADE'00)*, volume 1831 of *LNCS*, pages 255–270. Springer, 2000.

[4] R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench: A Semantics Based Tool for the Verification of Concurrent Systems. *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, January 1993.

[5] R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. In K.G. Larsen and A. Skou, editors, *Proceedings of Computer Aided Verification (CAV'91)*, volume 575 of *LNCS*, pages 48–58. Springer, 1992.

[6] 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.

[7] A. Dovier, C. Piazza, E. Pontelli, and G. Rossi. Sets and constraint logic programming. *Transaction on Programming Languages and Systems*, 22(5):861–931, 2000.

[8] R. Gentilini, C. Piazza, and A. Policriti. Simulation as stable coarsest partition problem. Technical Report 04-01, Dep. of Computer Science, University of Udine, Italy, 2001. http://www.dimi.uniud.it/~piazza/simul.ps.gz.

[9] 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.

[10] C. Piazza and A. Policriti. Ackermann encoding, bisimulation, and obdd's. In *Proceedings of Workshop on Verification and Computational Logic (VCL'01)*, 2001.