Simulation Reduction as Constraint

R. Gentilini a,1 C. Piazza b,2 A. Policriti a,3

a Dip. di Matematica e Informatica, Univ. di Udine.
Via delle Scienze 206, 33100 Udine (Italy).
b Dip. di Informatica, Univ. Ca’ Foscari di Venezia.
Via Torino 155, 30173 Mestre – Venezia (Italy).

Abstract

The simulation relation is largely used in Model Checking where it allows to reduce Kripke structures, on which verification takes place, while preserving significant fragments of Temporal Logic. 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. 4

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 structures (cf. [12,3]). In practice, the quotient with respect to simulation of the input Kripke structure can be used to (often significantly) reduce the size of the model, while preserving important fragments of temporal logics such as $\forall CTL^*$ and $\Box \mu – calculus ([14,3])$.

Several polynomial-time algorithms to compute similarity quotients have been proposed. The ones presented in in [1], [4], and [5] determine the simulation over a Kripke structure with $|N|$ states and $|E|$ transitions using

4 A preliminary version of this work has been presented to the workshop SAVE 2001.
1 Email: gentilin@dimi.uniud.it
2 Email: piazza@dimi.uniud.it
3 Email: policrit@dimi.uniud.it

©2002 Published by Elsevier Science B. V.
A simulation procedure having a time complexity of $O(|N|E)$ was independently defined in [2] and in [12]. The space required by each of the above mentioned routines is limited from below by $|N|^2$. Recently [6], [3], and [10] improved the space complexity of the problem which is an extremely significant parameter in the context of verification. In particular, the procedure in [6] interleaves the computation of the simulation relation with the determination of the bisimulation quotient. Both the algorithm in [3] and the one in [10] instead, get to the simulation quotient $S$ by successively refining a partition coarser than $S$.

We consider the problem from a perspective whose starting point is an encoding of the simulation problem into a membership/inclusion problem. In this way we naturally get to a procedure that allows us to fruitfully combine an important space-saving tool as the simulation-reduction with constraint logic programming (CLP) methodology. Moreover, the set-theoretic framework ultimately suggests the use of the (set) notion of rank to define a layering on the input structure suitable for driving the simulation computation. Taking advantage of such a layering, a constraint solver embedding sets allows to define incrementally a simulation over a labelled graph (cf. Sections 4 and 5). Indeed, both a constraint solver involving flat sets (like Conjunto [11]) and a constraint solver allowing nested sets (like the interpreter $\{\log\}$ [7,9]) can be used to this purpose. In this paper we describe the proposed incremental simulation computation within $\{\log\}$.

Similar ideas were already used in [8] and [15] for bisimulation computation, allowing to obtain an algorithm whose complexity is in some case linear and in general matches the best ones 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 (technical) stumbling block to this end is the existence of a suitable notion of rank, in Section 4 we comment and discuss this problem.

Even though it is not the central point of this paper, in Section 7 we briefly present a technique that, exploiting again the layering of the graph, allows to reuse subtree associated to nodes at different ranks in the OBDD-representation of the Kripke structure. The technique is currently under study and was already proposed in the context of bisimulation in [15].

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 $CLP(SETS)$ 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 2.1 [Labelled Graphs] Let $\Sigma$ 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 2.2 [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:

1. (label) $n \leq m \Rightarrow \ell(n) = \ell(m)$;
2. (forw) $n \leq m \land nE n' \Rightarrow \exists m'(mEm' \land n' \leq m')$.

Lemma 2.3 Given a graph $G$ there exists a maximum simulation $\leq_s$ over $G$.

Definition 2.4 [Maximum Simulation Equivalence] Given $G = \langle N, E, \ell \rangle$, the maximum simulation equivalence $\equiv_s \subseteq N \times N$ is defined as: $n \equiv_s m \iff n \leq_s m \land m \leq_s n$.

Lemma 2.5 The relation $\equiv_s$ is an equivalence relation.

We consider the problem of computing $\equiv_s$ over $G = \langle N, E, \ell \rangle$. All of the algorithms mentioned in the introduction ([1], [4], [5], [2], [12]) obtain the similarity quotient, $S = N/ \equiv_s$, as a by-product of the computation of the similarity relation on $N$. Their space complexity is then limited from below by $O(|N|^2)$.

The simulation algorithms in [6], [3], and [10] do not compute the entire simulation relation over $N$ and hence improve the space complexity of the problem. The procedure in [3] successively refines a partition which is coarser than $S = N/ \equiv_s$ and determines the simulation relation over $S$: it uses only $|S|^2 + |N| \log(|S|)$ space, but requires $O(|S|^4(|E| + |S|^2) + |S|^3 |N| (|N| + |S|^2))$ time. The algorithm in [10] combines ideas from [12] and [3], together with suitable data structures representing intermediate values for the reduced labeled graph, to obtain $\equiv_s$ in time $O(|S|^2 |E|)$ and space $O(|S|^2 + |N| \log |S|)$. In [6] the computation of the simulation relation is interleaved with the determination of the bisimulation quotient $B$, achieving a time complexity of $O(|B||E|)$ and a space complexity of $O(|B|^2 + |N| \log |B|)$. Since $B$ is a partition finer than $|S|$ the space requirements of [3] and [10] are lower than the ones of [6].


3 Encoding the Simulation Problem by Sets

In [15] 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 $m$: $m \sim n$ if and only if $A(n) \in A(m)$.

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 $m \geq n$ without having the set associated to $m$ including the set associated to $n$:\footnote{e.g. $m'$ is the only child of $m$, while $m'$ and $n'$ with $m' \geq n'$ are the children of $n$}

$m \geq n \not\Rightarrow A(m) \supseteq A(n)$. Hence, we can easily build examples in which $m \equiv n$ cannot be checked considering equality between the associated sets.

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 $\leq$ as follows: $a(n) \subseteq A(n)$ if and only if $n \leq m$. 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 the nodes simulated by $m$, it does not necessarily correspond to the collection of encoding relative to children of $m$ and the above mentioned consideration does not apply.

Recalling that by our convention $n \in N$ is a natural number, we assign to $n$ the set: $a(n) = \{\emptyset\}^n = n$, 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 $A_n$ and the constraint $C(n)$:

$$C(n) = \{a(n)\} \subseteq A_n \subseteq \{a(m) \mid m \in N \land \ell(m) = \ell(n)\} \land \bigwedge_{\ell(m) = \ell(n)} (a(m) \in A_n \leftrightarrow \{a(r) \mid mEr\} \subseteq \bigcup_{n \in E_s} A_r).$$

Let $Sys_G$ be the constraint conjunction of all the $C(n)$’s: $Sys_G = \bigwedge_{n \in N} C(n)$.

We prove the correctness of our approach by showing that:

1. each solution $Sol$ of $Sys_G$ determines a simulation $\leq_{Sat}$;
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 \( Sol1 \subseteq Sol2 \), where \( \subseteq \) is a partial order naturally induced over the set of solutions of \( Sys_G \) (cf. Definition 3.3).

**Lemma 3.1** Let \( G = \langle N, E, \ell \rangle \) 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 \( n \leq_{sol} m \Leftrightarrow a(n) \in A(m) \). 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 3.2** Let \( G = \langle N, E, \ell \rangle \) and \( \leq_s \) be the maximum simulation over \( G \). Let \( SM = \{ A(n) = \{ a(m) \mid m \leq_s n \} \mid n \in N \} \). \( SM \) is a solution of \( Sys_G \). Moreover, \( \leq_{SM} = \leq_s \), where \( \leq_{SM} \) is the simulation defined from \( SM \) as in Lemma 3.1.

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 3.3** Let \( G = \langle N, E, \ell \rangle \) be a graph and \( S \) be the set of solutions of \( Sys_G \). Consider \( Sol1 = \{ A1(n) \mid n \in N \} \) and \( Sol2 = \{ A2(n) \mid n \in N \} \) two elements of \( S \), we define the partial order \( \subseteq \) over \( S \) as follows:

\[
Sol1 \subseteq Sol2 \Leftrightarrow \forall n \in N (A1(n) \subseteq A2(n)).
\]

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 3.4** If \( \leq_{Sol1} \subseteq \leq_{Sol2} \), then \( Sol1 \subseteq Sol2 \).

**Theorem 3.5** Let \( G \) be a graph. There exists a \( \subseteq \)-maximum solution of \( Sys_G \). Moreover, if \( SM \) is the \( \subseteq \)-maximum solution of \( Sys_G \), then \( \leq_{SM} \) is the maximum simulation over \( G \).
Example 3.6 From the graph $G$ in Figure 1 we obtain:

$$\{a(1)\} \subseteq A_1 \subseteq \{a(1), a(3), a(5)\} \land$$

$$(a(1) \in A_1 \leftrightarrow \emptyset \subseteq \emptyset) \land$$

$$(a(3) \in A_1 \leftrightarrow \{a(2), a(4)\} \subseteq \emptyset)$$

$$\ldots$$

$$\{a(6)\} \subseteq A_6 \subseteq \{a(2), a(4), a(6)\} \land$$

$$(a(2) \in A_6 \leftrightarrow \{a(1)\} \subseteq A_3) \land$$

$$(a(4) \in A_6 \leftrightarrow \{a(5)\} \subseteq A_3) \land$$

$$(a(6) \in A_6 \leftrightarrow \{a(3)\} \subseteq A_3)$$

The maximum solution of $Sys_G$ is $A(1) = \{a(1)\}$, $A(2) = \{a(2)\}$, $A(3) =$

![Fig. 1. Example.](image)

$A(5) = \{a(1), a(3), a(5)\}$, $A(4) = A(6) = \{a(4), a(6)\}$.

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 [8,15] in the case of bisimulation.

Remark 3.7 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:

$$a(n) = a(m) \iff n \equiv_s m \quad A(n) \subseteq A(m) \iff n \leq_s m$$
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 4.1** [Strongly Connected Components] Given $G = \langle N, E \rangle$, let $G^{\text{scc}} = \langle N^{\text{scc}}, E^{\text{scc}} \rangle$ be the graph defined as: $N^{\text{scc}} = \{c : c$ is a strongly connected component in $G\}$; $E^{\text{scc}} = \{\langle c_1, c_2 \rangle : c_1 \neq c_2$ and $\exists n_1 \in c_1, n_2 \in c_2 (\langle n_1, n_2 \rangle \in E)\}$. Given a node $n \in N$, we refer to the node of $G^{\text{scc}}$ associated to the strongly connected component of $n$ as $c(n)$.

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

**Definition 4.2** [Rank] Let $G = \langle N, E \rangle$. The rank of a node $n$ of $G$ is:

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

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 4.3** Consider the graph in Figure 2.

First considering only the nodes at rank 0 we obtain $a(1) = 1$, $a(2) = 2$.  

![Fig. 2. Example.](image-url)
and

\[ \{a(1)\} \subseteq A_1 \subseteq \{a(1), a(2)\} \land (a(1) \in A_1 \leftrightarrow \emptyset \subseteq \emptyset) \land (a(2) \in A_1 \leftrightarrow \emptyset \subseteq \emptyset) \land \]

\[ \{a(1)\} \subseteq A_2 \subseteq \{a(1), a(2)\} \land (a(1) \in A_2 \leftrightarrow \emptyset \subseteq \emptyset) \land (a(2) \in A_2 \leftrightarrow \emptyset \subseteq \emptyset) \]

whose maximum solution is \( A(1) = A(2) = \{a(1), a(2)\} \). Since 1 and 2 are similar, we re-assign to them the same \( a \)-encoding: \( a(1) = a(2) = 1 \).

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: \( a(1) = a(2) = 1, a(3) = 2, a(4) = 3, a(5) = 4 \). From which we obtain

\[ \{a(1)\} \subseteq A_1 \subseteq \{a(1), a(5)\} \land (a(5) \in A_1 \leftrightarrow \{a(4)\} \subseteq \emptyset) \land \ldots \]

whose solution is \( A(1) = A(2) = \{a(1)\}, A(3) = \{a(3)\}, A(4) = \{a(3), a(4)\}, A(5) = \{a(5), a(1)\} \). 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: \( a(1) = a(2) = 1, a(3) = 2, a(4) = 3, a(5) = 4, a(6) = 5 \), and

\[ \{a(1)\} \subseteq A_1 \subseteq \{a(1), a(6)\} \land (a(6) \in A_1 \leftrightarrow \{a(3)\} \subseteq \emptyset) \land \ldots \]

whose solution is \( A(1) = A(2) = \{a(1)\}, A(3) = \{a(3)\}, A(4) = \{a(3), a(4)\}, A(5) = \{a(5), a(1), a(6)\}, A(6) = \{a(6), a(1)\} \). 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: \( a(1) = a(2) = 1, a(3) = 2, a(4) = 3, a(5) = 4, a(6) = 5, a(7) = 6, a(8) = 7 \), and \( \ldots \land (a(7) \in A_3 \leftrightarrow \{a(6), a(8)\} \subseteq A_1) \land \ldots \) whose solution is \( A(1) = A(2) = \{a(1)\}, A(3) = \{a(3)\}, A(4) = A(7) = \{a(3), a(4), a(7)\}, A(5) = A(8) = \{a(5), a(1), a(6), a(8)\}, A(6) = \{a(6), a(1)\} \). Hence, we can re-assign the \( a \)-encoding as follows:

\( a(1) = a(2) = 1, a(3) = 2, a(4) = a(7) = 3, a(5) = a(8) = 4, a(6) = 5 \)

and obtain \( A(1) = A(2) = \{1\}, A(3) = \{2\}, A(4) = A(7) = \{2, 3\}, A(5) = A(8) = \{4, 1, 5\}, A(6) = \{5, 1\} \).

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, in general, there are similar nodes at different ranks (see 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. [8]);
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, then two well-founded nodes (see [8]) are similar if and only if they have the same rank, while all the non-well-founded nodes are similar.

5 Incremental Computation of the Simulation Relation

Given a graph $G$, it is extremely natural to use a constraint solver embedding sets, in order to determine the $\sqsubseteq$-maximum solution of $S_{ysG}$. Both a CLP framework managing the entire class of hereditarily finite sets (like \{log\} [9]) and a CLP framework involving only flat sets (like Conjunto [11]) could be used to this purpose. In fact all the sets involved in the definition and in the solution to $S_{ysG}$ are finite and not nested sets. In this section we use the interpreter \{log\}, which based on the constraint logic programming language $CLP(SET)$ [9], to show how to compute incrementally a solution to $S_{ysG}$ within a CLP setting. In particular, the version of \{log\} we refer to in this paper is available at http://www.dimi.uniud.it/~piazza/LOG and can be consulted with SICStus.

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

$$a(1) = 1 \ldots a(k) = k' \quad (\ast)$$

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

with $a(h) = a(j) \iff A(h) = A(j)$ (i.e. we have re-assigned the $a$-encodings) and $\max\{a(1), \ldots, a(k)\} = \mathbf{m}$, 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

\[ a(k+1) = m+1 \ldots a(k+s) = m+s; \quad (***) \]

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

\[ A(h) \subseteq \mathbb{A}_h \subseteq A(h) \cup \{ a(j) \mid \text{rank}(j) = i + 1 \land \ell(j) = \ell(h) \}\wedge \bigwedge_{\text{rank}(j)=i+1\land\ell(j)=\ell(h)} (a(j) \in \mathbb{A}_h \leftrightarrow \{ a(t) \mid j \in \text{Ev} \} \subseteq \bigcup_{h \in \text{Ev}} \mathbb{A}_h); \]

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

\[ \{ a(h) \} \subseteq \mathbb{A}_h \subseteq \{ a(j) \mid \text{rank}(j) \leq i + 1 \land \ell(j) = \ell(h) \}\wedge \bigwedge_{\text{rank}(j)\leq i+1\land\ell(j)=\ell(h)} (a(j) \in \mathbb{A}_h \leftrightarrow \{ a(t) \mid j \in \text{Ev} \} \subseteq \bigcup_{h \in \text{Ev}} \mathbb{A}_h); \]

• if $A(1) = \{ a(1), \ldots \} \ldots A(k+s) = \{ a(k+s), \ldots \}$ is the $\subseteq$-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.

Below we discuss the (limited) fragment of $CLP(SET)$ needed to solve our constraints. A $CLP(SET)$-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), $\{ t_1 \mid t_2 \}$ stands for $\{ t_1 \} \cup t_2$ (i.e. the elements of $\{ t_1 \mid t_2 \}$ 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 $CLP(SET)$-constraint is composed by conjunctions (written $\&$) and
disjunctions (written or) of literals of the form\footnote{In the standard version of \textit{CLP(S\textsc{et})} presented in [9] and in the standard interpreter \{log\} available at \url{http://prmat.math.unipr.it/~gianfr/setlog.Home.html} there are no predicates \texttt{sub}, \texttt{nsub} and \texttt{betw}. These predicates can be straightforward built disposing of \texttt{un} as \( t_1 \subseteq t_2 \) if \( \text{un}(t_1,t_2,t_2) \). Thus, they have easily been added in the version of \{log\} used here: they are only shorthand for \textit{CLP(S\textsc{et})} -constraints.}

\[
t_1 = t_2 \text{ t}_1 \text{ neq} t_2 \text{ t}_1 \text{ in} t_2 \text{ t}_1 \text{ nin} t_2 \text{ un}(t_1,t_2,t_3) \text{ nun}(t_1,t_2,t_3) \text{ sub}(t_1,t_2) \text{ nsub}(t_1,t_2) \text{ betw}(t_1,t_2,t_3)
\]

whose semantics can be intuitively described as follows

\[
t_1 = t_2 \iff t_3 \subseteq t_1 \iff t_4 \subseteq t_2 \quad t_1 \text{ neq} t_2 \iff t_1 \neq t_2
\]
\[
t_1 \text{ in} t_2 \iff t_1 \subseteq t_2 \quad t_1 \text{ nin} t_2 \iff t_1 \nsubseteq t_2
\]
\[
\text{un}(t_1,t_2,t_3) \iff t_1 \cup t_2 = t_3 \quad \text{nun}(t_1,t_2,t_3) \iff t_1 \cup t_2 \neq t_3
\]
\[
\text{sub}(t_1,t_2) \iff t_1 \subseteq t_2 \quad \text{nsub}(t_1,t_2) \iff t_1 \nsubseteq t_2
\]
\[
\text{betw}(t_1,t_2,t_3) \iff t_1 \subseteq t_2 \subseteq t_3
\]

Now we show an example of the incremental computation of the simulation using \textit{CLP(S\textsc{et})}-constraint and \{log\}.

**Example 5.1** Consider the graph in Figure 3.

\[
\begin{align*}
\text{betw}([1],A_1,[1,3]) & \land ([3 \in A_1 \iff [2] \subseteq A_2) \land [2] \subseteq A_2 \subseteq [2] \land \\
[3] & \subseteq A_3 \subseteq [1,3] \land (1 \in A_3 \iff [2] \subseteq A_2),
\end{align*}
\]

which written as \textit{CLP(S\textsc{et})}-constraint is

\[
\text{betw}([1],A_1,[1,3]) \land (\text{in} A_1 \land \text{sub}([2],A_2)) \lor (\text{nin} A_1 \land \text{nsub}([2],A_2)) \land \\
\text{betw}([2],A_2,[2]) \land \\
\text{betw}([3],A_3,[1,3]) \land (\text{in} A_3 \land \text{sub}([2],A_2)) \lor (\text{nin} A_3 \land \text{nsub}([2],A_2)).
\]

Fig. 3. Example.
In this case \( \{ \log \} \) gives us only the solution \( A_1 = \{1, 3\} \), \( A_2 = \{2\} \), \( A_3 = \{3, 1\} \). Since \( A_1 = A_3 \) and \( a(1) < a(3) \) we re-assign \( a(3) = a(1) = 1 \), obtaining
\[
a(1) = a(3) = 1 \quad a(2) = 2 \quad A(1) = A(3) = \{1\} \quad A(2) = \{2\}.
\]

At the second step we start with the \( a \)-encoding \( a(1) = a(3) = 1 \), \( a(2) = 2 \), \( a(4) = 3 \), \( a(5) = 4 \), and the constraint
\[
\{1\} \subseteq A_1 \subseteq \{1, 4\} \land (4 \in A_1 \leftrightarrow \{3\} \subseteq A_2) \land \\
\{2\} \subseteq A_2 \subseteq \{2, 3\} \land (3 \in A_2 \leftrightarrow \{4, 1\} \subseteq A_1) \land \\
\{3\} \subseteq A_3 \subseteq \{3, 2\} \land (2 \in A_3 \leftrightarrow \{1\} \subseteq A_1 \cup A_4) \land \\
\{4\} \subseteq A_4 \subseteq \{4, 1\} \land (1 \in A_4 \leftrightarrow \{2\} \subseteq A_3),
\]

which written as \( CLP(SET) \)-constraint becomes
\[
\text{betw}(\{1\}, A_1, \{1, 4\}) \& ((4 \in A_1 \& \text{sub}(\{3\}, A_2)) \lor (4 \in A_1 \& \text{nsub}(\{3\}, A_2))) \& \\
\text{betw}(\{2\}, A_2, \{2, 3\}) \& \\
((3 \in A_2 \& \text{sub}(\{4, 1\}, A_1)) \lor (3 \in A_2 \& \text{nsub}(\{4, 1\}, A_1))) \& \\
\text{betw}(\{3\}, A_3, \{3, 2\}) \& (\text{un}(A_1, A_4, X) \& \\
((2 \in A_3 \& \text{sub}(\{1\}, X)) \lor (2 \in A_3 \& \text{nsub}(\{1\}, X))) \& \\
\text{betw}(\{4\}, A_4, \{4, 1\}) \& ((1 \in A_4 \& \text{sub}(\{2\}, A_3)) \lor (1 \in A_4 \& \text{nsub}(\{2\}, A_3))).
\]

In this case \( \{ \log \} \) gives us the two solutions
\[
A_1 = \{1\}, \quad A_2 = \{2\}, \quad A_3 = \{3, 2\}, \quad A_4 = \{4, 1\}; \\
A_1 = \{1, 4\}, \quad A_2 = \{2, 3\}, \quad A_3 = \{3, 2\}, \quad A_4 = \{4, 1\}.
\]

The second solution is the maximum one and using it to re-assign the \( a \)-encoding we obtain \( a(1) = a(3) = a(5) = 1 \), \( a(2) = a(4) = 2 \), \( A(1) = A(3) = A(5) = \{1\} \), \( A(2) = A(4) = \{2\} \). Notice that, even though the \( A \)'s are singletons here, this is not true in general (cf. Examples 3.6 and 4.3).

### 6 Complexity Issues

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

First of all we recall from [9] that it is possible to encode NP-complete problems in \( CLP(SET) \)-constraints: given an instance of the 3-SAT problem in \( k \) propositional variables it is possible to write a \( CLP(SET) \)-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 and this should came as no surprise, given the expressivity of the class of constraint decided by \{\log\}.

Given a graph \(G = (N, E, \ell)\) we obtain a constraint \(\text{Sys}_G\) in \(|N|\) variables (the variables \(A_n\)’s). In the case of the constraints of the form \(\text{Sys}_G\) the interpreter \{\log\}, as it is currently implemented, uses a sort of generate and test strategy: it starts initializing all the \(A_n\)’s to \(N\), it tests if all the literals in \(\text{Sys}_G\) are satisfied, if this is not the case it removes one element from one of the \(A_n\)’s, and it tests if now all the literals in \(\text{Sys}_G\) are satisfied, and so on, until the solution is found. Since the number of possible assignments to the variables \(A_n\)’s is \(2^{(|N|-1)}|N|\), in the worst case we have a non-polynomial complexity. Let us analyse some possible optimizations suggested from the set encoding, which allow to design a constraint-solver for the constraints of the form \(\text{Sys}_G\) and which works in polynomial time. We can imagine that the constraint-solver works in this way:

- the \(A_n\)’s are all initialized to \(N\);
- if \(\text{Succ}(m) = \{a(r) \mid mEr\} \not\subseteq \bigcup_{n \in N} A_n = B_n\), it removes \(a(m)\) from \(A_n\).

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 \(B_n\);
- for each \(m\) the cost to perform the check \(S(m) \subseteq B_n\)

Using lists to represent the \(S(m)\)’s and arrays of length \(|N|\) for the \(A_n\)’s and the \(B_n\)’s we obtain that the total cost to find the maximal solution of \(\text{Sys}_G\) is \(O(|N|^3|E|)\). Using lists (without repetitions) to represent not only the \(S(m)\)’s, but also the \(A_n\)’s, and the \(B_n\)’s we obtain that the cost is \(O(|N|^4|E|)\).

An \textit{ad hoc} constraint-solver, designed and implemented just for simulation reduction computation, would deal with a more restricted class of constraints and certainly guarantee better performances.

7 Simulation and Symbolic Computation

The simulation’s set-encoding proposed in the previous sections can be naturally combined with the symbolic representation of Kripke structures based on the use of OBDD’s (see [13]). OBDD’s can be used both to perform the incremental computation and to obtain a compact representation of the quotiented structure.

In order to use OBDD’s in the incremental computation our idea is to:

- consider the binary representation of the \(a\)-encoding;
- determine the OBDD representations of the \(A\)’s associated to the nodes;
- use the OBDD representations of the \(A\)’s in order to refine \(\{a(n) \mid n \in N\}\) (the set of \(a\)-encoding assigned to the graph’s vertexes) by collapsing \(a\)-
values when the corresponding $A$-values are equal.

Let us assume that at step $i$ we have the partial solution as described in $(\ast)$ in Section 5 with the $a$-encoding represented in binary notation and the $A$’s represented as OBDD’s. We proceed assigning at each node $h$ at rank $i$ a new binary number $a(h)$ as described in $(\ast\ast)$. We compute the $A(h)$’s relative to all the nodes at rank at most $i$ using the OBDD:

$$\bigvee_{a(j) \in A(h)} a(j)$$

to represent $A(h)$. 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 $h$ and $j$ whose rank is at most $i$ must be the same if and only if the OBDD’s for $h$ and $j$ are equal. Once refined the set \{ $a(h) \mid \text{rank}(h) \leq i$ \} we can give a more compact symbolic representation (i.e. associated to a boolean function with less variables) of the $A$’s at rank at most $i$.

At the end of step $i$, when we already have refined the set \{ $a(h) \mid \text{rank}(h) \leq i$ \} of binary numbers, if $h$ is a node at rank $i$ such that $a(h) > m$ (see Section 5 $(\ast\ast)$), then we consider the OBDD

$$E(h) = \bigvee_{v \in \text{Big}(h)} a(v)$$

where $\text{Big}(h)$ the set of the successors of $h$ which are not simulated by any other successor of $h$ is formally defined as

$$\text{Big}(h) = \{ v \mid hEv \land a(v) \not\in \bigcup_{hEv' \land v' \neq v} A(v') \}.$$  

The OBDD representation of the quotient structure $G/ \equiv_s$ is given by the collection of all the OBDD’s of the form $E(h)$ that we compute. It is important to notice that the computation of the quotient structure is fully incremental in the sense that once we have computed $E(h)$ we never recompute it. This is due to the facts that: if a node $j$ is a successor of a node $h$, then \text{rank}(j) \leq \text{rank}(h); if a node $h$ has rank $i$, then after the end of the $i$th iteration $a(h)$ is never modified. Moreover, we avoid to compute $E(h)$ whenever $h$ is similar to a node $j$ and we have already computed $E(j)$.

**Example 7.1** Consider the graph in Figure 2. At the end of the computation relative to the nodes at rank 1 we have the following situation:
\[ a(1) = a(2) = 001, \ a(3) = 010, \ a(4) = 011, \ a(5) = 100, \ a(6) = 101; \]
\[
E(1) = E(2) = \bot \quad A(1) = A(2) = \bot \\
E(3) = \neg x_3 \land \neg x_2 \land x_1 \quad A(3) = E(3) \\
E(4) = x_3 \land \neg x_2 \land \neg x_1 \quad A(4) = E(4) \lor (\neg x_3 \land \neg x_2 \land x_1) \\
E(5) = \neg x_3 \land x_2 \land x_1 \quad A(5) = E(5) \\
E(6) = \neg x_3 \land x_2 \land \neg x_1 \quad A(6) = E(6).
\]

We begin the computation at rank 2 with \( m = 101 \) and \( a(7) = 110, \ a(8) = 111 \). We obtain that
\[
A'(4) = A'(7) = A(4) \lor (x_3 \land x_2 \land \neg x_1) \quad A'(8) = A'(5) = A(5) \lor (x_3 \land x_2 \land x_1).
\]

From this we deduce that we can reassign \( a(7) = a(4) = 011 \) and \( a(8) = a(5) = 100 \). Since all the \( a \)-encoding obtained at rank 2 are smaller than \( m \) we do not have to compute any new \( E \): we immediately have \( E(7) = E(4) \) and \( E(8) = E(5) \).

### 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 \( CLP(SET) \) 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 \( together \) with the simulation reduction. This positive by-product was obtained for the bisimulation relation with a similar approach in [15].
References


