Three Simulation Algorithms for Labelled Transition Systems

Gérard Cécé

Three Simulation Algorithms for Labelled Transition Systems

Gérard Cécé

FEMTO-ST, UMR 6174,
1 cours Leprince-Ringuet, BP 21126,
25201 Montbliard Cedex France
Gerard.Cece@femto-st.fr

January 8, 2013

Abstract

Algorithms which compute the coarsest simulation preorder are generally designed on Kripke structures. Only in a second time they are extended to labelled transition systems. By doing this, the size of the alphabet appears in general as a multiplicative factor to both time and space complexities. Let Q denotes the state space, $\rightarrow$ the transition relation, Σ the alphabet and P_{sim} the partition of Q induced by the coarsest simulation equivalence. In this paper, we propose a base algorithm which minimizes, since the first stages of its design, the incidence of the size of the alphabet in both time and space complexities. This base algorithm, inspired by the one of Paige and Tarjan in 1987 for bisimulation and the one of Ranzato and Tapparo in 2010 for simulation, is then derived in three versions. One of them has the best bit space complexity up to now, $O(|P_{sim}|^2 + |\rightarrow|.\log|\rightarrow|)$, while another one has the best time complexity up to now, $O(|P_{sim}|.|\rightarrow|)$. Note the absence of the alphabet in these complexities. A third version happens to be a nice compromise between space and time since it runs in $O(b.|P_{sim}|.|\rightarrow|)$ time, with b a branching factor generally far below $|P_{sim}|$, and uses $O(|P_{sim}|^2.\log|P_{sim}| + |\rightarrow|.\log|\rightarrow|)$ bits.

1 Introduction

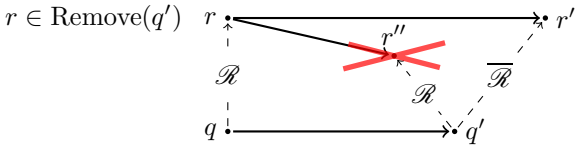
Simulation is a behavioral relation between processes [7]. It is mainly used to tackle the state-explosion problem that arises in *model checking* [5, 1] and to speed up the test of inclusion of languages [2]. It can also be used as a sufficient condition for the inclusion of languages when this test is undecidable in general [3]. The paper [5] gives a complete state of the art of the notion.

1.1 Last Ten Years

Let $T = (Q, \Sigma, \rightarrow)$ be a Labelled Transition System (LTS) with Q its set of states, Σ its alphabet and $\rightarrow \subseteq Q \times \Sigma \times Q$ its transition relation. A relation $\mathscr{S} \subseteq Q \times Q$ is a simulation

over T if for any transition $q_1 \xrightarrow{a} q_1'$ and any state $q_2 \in Q$ such that $(q_1, q_2) \in \mathscr{S}$, there is a transition $q_2 \xrightarrow{a} q_2'$ such that $(q_1', q_2') \in \mathscr{S}$. The simulation $\mathscr{S}$ is a bisimulation if $\mathscr{S}^{-1}$ is also a simulation. Given any preorder (reflexive and transitive relation) $\mathscr{R} \subseteq Q \times Q$ the purpose of this paper is to design efficient algorithms which compute the coarsest simulation over T included in $\mathscr{R}$.

In the context of Kripke structures, which are transition systems where only states are labelled, the most efficient algorithms are GPP, the one of Gentilini, Piazza and Policriti [5] (corrected by van Glabbeek and Ploeger [11]), for the space efficiency, and RT, the one of Ranzato and Tapparo [9], for the time efficiency. These two algorithms either use, for GPP, or extend, for RT, HHK the one of Henzinger, Henzinger and Kopke [6].



The starting idea of HHK, see the above figure, is to consider couples (q', r') that do not belong to $\mathscr{R}$ (thus (q', r') belongs to $\overline{\mathscr{R}}$ the complement of $\mathscr{R}$) and to propagate this knowledge backward by refining $\mathscr{R}$. For each state q' a set of states, $\text{Remove}(q')$, is maintained. This set is included in the complement of $\text{pre}(\mathscr{R}(q'))$, the set of states which have at least one outgoing transition leading to a state related to q' by $\mathscr{R}$. In the figure above, to illustrate that a state r belongs to $\text{Remove}(q')$ we depict that there is no state r'' reachable from r and such that (q', r'') belongs to $\mathscr{R}$. For a given state q', $\text{Remove}(q')$ is used as follows: for each couple $(q, r) \in \text{pre}(q') \times \text{Remove}(q')$, with $\text{pre}(q')$ the set of states leading, by a transition, to q', if (q, r) belongs to $\mathscr{R}$ then it is removed and $\text{Remove}(q)$ is possibly updated. The couple (q, r) is safely removed from $\mathscr{R}$ because by the definition of $r \in \text{Remove}(q')$ it is impossible that (q, r) belongs to a simulation included in $\mathscr{R}$. The algorithm HHK runs in (remember, for the moment transitions are not labelled) $O(|\!\rightarrow\!|.|Q|)$ time and uses $O(|Q|^2.\log|Q|)$ bits for all the Remove sets. Note that in order to achieve the announced time complexity the authors use a set of counters which plays the same role as this introduced by Paige and Tarjan [8] to lower the time complexity for the corresponding bisimulation problem from $O(|\!\rightarrow\!|.|Q|)$ to $O(|\!\rightarrow\!|.\log|Q|)$. In HHK the set of counters enable to lower the time complexity for the simulation problem from $O(|\!\rightarrow\!|.|Q|^2)$ to $O(|\!\rightarrow\!|.|Q|)$.

If one extends HHK to LTS, where transitions are labelled, there is a necessity to maintain a Remove set for each couple state-letter (q', a) because, now, $\text{Remove}_a(q')$ is included in the complement of $\text{pre}_a(\mathscr{R}(q'))$ and pre need to depend on the letters labelling the transitions. Then, any natural extension of HHK to LTS uses $O(|\Sigma|.|Q|^2.\log|Q|)$ bits for all the Remove's.

Let us come back to Kripke structures. The main difference between HHK in one hand and, GPP and RT in the other hand is that the last two do not encode the current relation $\mathscr{R}$ by a binary matrix of size $|Q|^2$ but by a partition-relation pair: a couple (P, R) with R a binary matrix of size $|P|^2$ and P the partition of Q issued from the equivalence relation $\mathscr{R} \cap \mathscr{R}^{-1}$. The difficulty of the proofs and the abstract interpretation framework put aside, RT is a thus a direct reformulation of HHK but with partition-relation pairs instead of mere relations between states. Note that in order to have sound refinements

of the relation R, blocks of P may first be split at each main iteration of the algorithm. The algorithm RT maintains for each block $B \in P$ a set Remove(B) included in the complement of pre($\mathscr{R}(B)$), the set of states which have at least one outgoing transition leading to the set of states related to B by $\mathscr{R}$. The algorithm runs in $O(|P_{sim}|.|{\rightarrow}|)$ time and uses $O(|P_{sim}|.|Q|.\log|Q|)$ bits for all the Remove's, with P_{sim} the partition associated to the coarsest simulation relation included in the initial preorder $\mathscr{R}_{init}$. In [4], Crafa and the authors of RT, reduced this space complexity to $O(|P_{sim}|.|P_{sp}|.\log|P_{sp}|)$ with P_{sp} a partition whose size is between these of P_{sim} and P_{bis}, the partition associated to the coarsest bisimulation included in $\mathscr{R}_{init} \cap \mathscr{R}_{init}^{-1}$. The goal, which has not been achieved, was a bit space complexity of $O(|P_{sim}|^2.\log|P_{sim}|)$. The bit space complexity of [4] is achieved at the cost of an increase of the time complexity comparable with $O(|P_{sim}|^2.|{\rightarrow}|)$. The algorithm GPP uses a partition-relation pair (P, R) too. It also proceed by iterative steps of one split of P and one refinement of R. A split step is done in a more global way than in RT, then a refinement step uses HHK on an abstract structure whose states are blocks of P. A refinement step is thus done in $O(|P_{sim}|.|{\rightarrow}|)$ time (remember, here states are blocks of P). As they prove it, there is at most $|P_{sim}|$ refinement steps. The entire algorithm is thus done in $O(|P_{sim}|^2.|{\rightarrow}|)$ time. Since states are blocks in this use of HHK, the encoding of all the Remove's uses $O(|P_{sim}|^2.\log|P_{sim}|)$ bits, which has not been taken into account in the announced bit space complexity of GPP: $O(|P_{sim}|^2 + |Q|.\log|P_{sim}|)$.

The paper [1] provides an adaptation for LTS of RT. It is also a very useful translation of RT from the context of abstract interpretation to a more classical algorithmic view on simulations. The algorithm of [1] runs in $O(|\Sigma|.|P_{sim}|.|Q| + |P_{sim}|.|{\rightarrow}|)$ time and uses $O(|\Sigma|.|P_{sim}|.|Q|.\log|Q|)$ bit space.

1.2 Our Contributions

We have mainly focused our attention on the $|\Sigma|$ factor which is present in both time and space complexities of the simulation algorithm in [1]. The major step was to realize that if the Remove$_a(B)$ set associated with a block $B \in P$ need to depend on a letter, a set of blocks not related to $R(B) = \{C \in P \mid (B, C) \in R\}$ does not depend on any letter. Therefore, instead of maintaining Remove$_a(B)$ we maintain NotRel(B) a set of blocks included in the complement of $R(B)$ and we compute Remove$_a(B)$ only when we need it. Therefore, we do not have to store it. The great by-product of doing this is that, for each block, we now maintain a set of blocks, encoded with $O(|P_{sim}|.\log|P_{sim}|)$ bits, and not a set of states encoded with $O(|Q|.\log|Q|)$ bits. Thus, we also achieve the main goal of [4].

In the next two sections we state the preliminaries and clarify our views regarding the underlying theory. Then, we propose our base algorithm. In the section which follows we derive this base algorithm in several versions. The first one runs in $O(\min(|P_{sim}|, b).|P_{sim}|.|{\rightarrow}|)$ time, with b a branching factor, of the underlying LTS, defined in Section 5.3, and uses $O(|P_{sim}|^2.\log|P_{sim}| + |{\rightarrow}|.\log|{\rightarrow}|)$ bit space. By adding a set of counters, like in [9, 1], we obtain a second version of the algorithm that runs in $O(|P_{sim}|.|{\rightarrow}|)$ time and uses $O(|P_{sim}|.|\operatorname{sl}({\rightarrow})|.\log|Q| + |{\rightarrow}|.\log|{\rightarrow}|)$ bit space, with (in common cases): $|P_{sim}| \leq |Q| \leq |\operatorname{sl}({\rightarrow})| \leq |{\rightarrow}| \leq |\Sigma \times Q|$. The adding space is used to store the counters and is the price to pay to obtain the best time complexity. This version of the algorithm becomes the best one, for LTS, regarding time efficiency. We then explain why

GPP does not have a bit space complexity of $O(|P_{sim}|^2 + |Q|.\log|P_{sim}|)$ but at least of $O(|P_{sim}|^2.\log|P_{sim}| + |Q|.\log|Q|)$. Then, we propose the third version of our base algorithm. It runs in $O(|P_{sim}|^2.|{\to}|)$ time and uses $O(|P_{sim}|^2 + |{\to}|.\log|{\to}|)$ bits. It is the best one regarding space complexity. Then, we detail the data structures that we use. We end the paper by some perspectives including a future work on bisimulation.

2 Preliminaries

Let Q be a set of elements. The number of elements of Q is denoted $|Q|$. A *binary relation* on Q is a subset of $Q \times Q$. In the remainder of this paper, we consider only binary relations, therefore when we write "relation" read "binary relation". Let $\mathscr{R}$ be a relation on Q. For all $q, q' \in Q$, we may write $q\,\mathscr{R}\,q'$, or q -·$\mathscr{R}$·-› q' in the figures, when $(q,q') \in \mathscr{R}$. We define $\mathscr{R}(q) \triangleq \{q' \in Q \,|\, q\,\mathscr{R}\,q'\}$ for $q \in Q$ and $\mathscr{R}(X) \triangleq \cup_{q\in X}\mathscr{R}(q)$ for $X \subseteq Q$. In the figures, we note X -·$\mathscr{R}$·-› Y when there is $(q,q') \in X \times Y$ with $q\,\mathscr{R}\,q'$. The *domain* of $\mathscr{R}$ is $\mathrm{dom}(\mathscr{R}) \triangleq \{q \in Q \,|\, \mathscr{R}(q) \neq \emptyset\}$. The complement of $\mathscr{R}$ is $\overline{\mathscr{R}} \triangleq \{(x,y) \in Q \times Q \,|\, (x,y) \notin \mathscr{R}\}$. Let $\mathscr{S}$ be another relation on Q, the composition of $\mathscr{R}$ by $\mathscr{S}$ is $\mathscr{S} \circ \mathscr{R} \triangleq \{(x,y) \in Q \times Q \,|\, y \in \mathscr{S}(\mathscr{R}(x))\}$. The relation $\mathscr{R}$ is said *reflexive* if for all $q \in Q$, we have $q\,\mathscr{R}\,q$. The relation $\mathscr{R}$ is said *reflexive on its domain* if for all $q \in \mathrm{dom}(\mathscr{R})$, we have $q\,\mathscr{R}\,q$. The relation $\mathscr{R}$ is said *antisymmetric* if $q\,\mathscr{R}\,q'$ and $q'\,\mathscr{R}\,q$ implies $q = q'$. The relation $\mathscr{R}$ is said *transitive* if $\mathscr{R} \circ \mathscr{R} \subseteq \mathscr{R}$.

A *preorder* is a reflexive and transitive relation. Let X be a set of subsets of Q, we note $\cup X \triangleq \cup_{B\in X} B$. A *partition* of Q is a set of non empty subsets of Q, called *blocks*, that are pairwise disjoint and whose union gives Q. A *partition-relation pair* over Q is a pair (P,R) such that P is a partition of Q and R is a reflexive relation on P. A partition-relation pair (P,R) is said antisymmetric if its relation R is antisymmetric. From a partition-relation pair (P,R) over Q we derive a relation $\mathscr{R}_{(P,R)}$ on Q such that: $\mathscr{R}_{(P,R)} = \cup_{(B,C)\in R} B \times C$.

Definition 1. *Let $\mathscr{R}$ be a relation on a set Q such that $\mathscr{R}$ is reflexive on its domain.*

- *For $q \in Q$, we define $[q]_{\mathscr{R}} \triangleq \{q' \in Q \,|\, q\,\mathscr{R}\,q' \wedge q'\,\mathscr{R}\,q\}$, for $X \subseteq Q$, we define $[X]_{\mathscr{R}} \triangleq \cup_{q\in X}[q]_{\mathscr{R}}$.*
- *A* block *of $\mathscr{R}$ is a non empty set of states B such that $B = [q]_{\mathscr{R}}$ for a $q \in Q$.*
- *$\mathscr{R}$ is said* block-definable*, or* definable by blocks *if: $\forall q, q' \in Q\,.\,(q,q') \in \mathscr{R} \Rightarrow [q]_{\mathscr{R}} \times [q']_{\mathscr{R}} \subseteq \mathscr{R}$.*

Let us remark that a preorder is reflexive and definable by blocks. The notion of definability by blocks will be useful since intermediate relations of our algorithms will be block-definable, but not necessarily preorders, even if we start from a preorder and finish with a preorder too.

Remark. Let (P,R) be an antisymmetric partition-relation pair over a set Q. Then: $P = \{[q]_{\mathscr{R}_{(P,R)}} \subseteq Q \,|\, q \in Q\}$ and $R = \{([q]_{\mathscr{R}_{(P,R)}}, [q']_{\mathscr{R}_{(P,R)}}) \subseteq P \times P \,|\, q\,\mathscr{R}_{(P,R)}\,q'\}$.

From a reflexive and block-definable relation $\mathscr{R}$ we derive an antisymmetric partition-relation pair $(P_{\mathscr{R}}, R_{\mathscr{R}})$ such that $P_{\mathscr{R}} = \{[q]_{\mathscr{R}} \subseteq Q \,|\, q \in Q\}$ and $R_{\mathscr{R}} = \{([q]_{\mathscr{R}}, [q']_{\mathscr{R}}) \subseteq P \times P \,|\, q\,\mathscr{R}\,q'\}$.

Remark. Let $\mathscr{R}$ be a reflexive and block-definable relation on Q. Then $\mathscr{R} = \bigcup_{(B,C)\in R_{\mathscr{R}}} B\times C$.

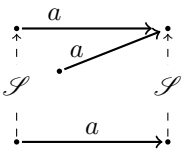
The two preceding remarks imply a duality between reflexive and block-definable relations, and antisymmetric partition-relation pairs. However, the notion of block-definable relation is somehow more general in the sense that we require its reflexibility on its domain, not necessarily on the whole state space Q.

Let $T = (Q, \Sigma, \to)$ be a triple such that Q is a finite set of elements called *states*, Σ is an *alphabet*, a finite set of elements called *letters* or *labels*, and $\to \subseteq Q \times \Sigma \times Q$ is a *transition relation* or *set of transitions*. Then, T is called a *Labelled Transition System (LTS)*. From T, given a letter $a \in \Sigma$, we define the two following relations: $\xrightarrow{a} \triangleq \{(q, q') \,|\, (q, a, q') \in \to\}$ and its reverse $\mathrm{pre}_{\xrightarrow{a}} \triangleq \{(q', q) \,|\, (q, a, q') \in \to\}$. When $\to$ is clear from the context, we simply note pre_a instead of $\mathrm{pre}_{\xrightarrow{a}}$. For $X, Y \subseteq Q$, we note $X \xrightarrow{a} Y$ to express that $X \cap \mathrm{pre}_a(Y) \neq \emptyset$. By abuse of notation, we also note $q \xrightarrow{a} Y$ for $\{q\} \xrightarrow{a} Y$. In the complexity analysis of the algorithms proposed in this paper, a new notion has emerged, that of *state-letter*. From T we define the set of state-letters $\mathrm{sl}(\to) \triangleq \{(q, a) \in Q \times \Sigma \,|\, \exists q' \in Q \,.\, q \xrightarrow{a} q' \in \to\}$. For $(q, a) \in \mathrm{sl}(\to)$, we simply note q_a instead of (q, a). If T is "normalized" (see first paragraph of Section 5) we have:

$$|Q| \leq |\,\mathrm{sl}(\to)| \leq |\to| \leq |\Sigma \times Q| \tag{1}$$

It is therefore more interesting to use $\mathrm{sl}(\to)$ instead of $\Sigma \times Q$.

The following definition of a simulation happens to be more effective than the classical one given in the introduction.



Definition 2. *Let $T = (Q, \Sigma, \to)$ be a LTS and $\mathscr{S}$ be a relation on Q. The relation $\mathscr{S}$ is a* simulation *over T if: $\forall a \in \Sigma \,.\, \mathscr{S} \circ \mathrm{pre}_a \subseteq \mathrm{pre}_a \circ \mathscr{S}$. For two states $q, q' \in Q$, we say "q' simulates q" if there is a simulation $\mathscr{S}$ over Q such that $q \,\mathscr{S}\, q'$.*

3 Underlying Theory

The first consequence of the definition of a simulation over a LTS $T = (Q, \Sigma, \to)$ is that states which have an outgoing transition labelled by a letter a can be simulated only by states which have at least one outgoing transition labelled by a. The next definition and lemma establish that we can restrict our problem of finding the coarsest simulation inside a preorder to the search of the coarsest simulation inside a preorder $\mathscr{R}$ that satisfies:

$$\forall a \in \Sigma \,.\, \mathscr{R}(\mathrm{pre}_a(Q)) \subseteq \mathrm{pre}_a(Q). \tag{2}$$

Definition 3. *Let $T = (Q, \Sigma, \to)$ be a LTS and $\mathscr{R}$ be a preorder on Q. We define* $\mathrm{InitRefine}(\mathscr{R}) \subseteq \mathscr{R}$ *such that:*

$$(q, q') \in \mathrm{InitRefine}(\mathscr{R}) \Leftrightarrow (q, q') \in \mathscr{R} \wedge \forall a \in \Sigma \,(q \in \mathrm{pre}_a(Q) \Rightarrow q' \in \mathrm{pre}_a(Q)).$$

Lemma 4. *Let $T = (Q, \Sigma, \to)$ be a LTS and $\mathscr{U} = \mathrm{InitRefine}(\mathscr{R})$ with $\mathscr{R}$ a preorder on Q. Then:*

1. *$\mathscr{U}$ is a preorder,*

2. *for all simulation* $\mathscr{S}$ *over* T: $\mathscr{S} \subseteq \mathscr{R} \Rightarrow \mathscr{S} \subseteq \mathscr{U}$,

3. $\forall X \subseteq Q \, \forall a \in \Sigma \, . \, \mathscr{U}(\mathrm{pre}_a(X)) \subseteq \mathrm{pre}_a(Q)$.

Proof.

1. Since $\mathscr{R}$ is a preorder and thus reflexive, $\mathscr{U}$ is also trivially reflexive. Now, let us suppose $\mathscr{U}$ is not transitive. There are three states $q_1, q_2, q_3 \in Q$ such that: $q_1 \mathscr{U} q_2 \wedge q_2 \mathscr{U} q_3 \wedge \neg\, q_1 \mathscr{U} q_3$. From the fact that $\mathscr{U} \subseteq \mathscr{R}$ and $\mathscr{R}$ is a preorder, we get $q_1 \mathscr{R} q_3$. With $\neg\, q_1 \mathscr{U} q_3$ and the definition of $\mathscr{U}$ there is $a \in \Sigma$ such that: $q_1 \in \mathrm{pre}_a(Q)$ and $q_3 \notin \mathrm{pre}_a(Q)$. But $q_1 \in \mathrm{pre}_a(Q)$ and $q_1 \mathscr{U} q_2$ implies $q_2 \in \mathrm{pre}_a(Q)$. With $q_2 \mathscr{U} q_3$ we also get $q_3 \in \mathrm{pre}_a(Q)$ which contradicts $q_3 \notin \mathrm{pre}_a(Q)$.

2. If this is not true there are two states $q_1, q_2 \in Q$ such that: $q_1 \mathscr{S} q_2 \wedge \neg\, q_1 \mathscr{U} q_2$. From $\mathscr{U} \subseteq \mathscr{R}$ we get $q_1 \mathscr{R} q_2$. With $\neg\, q_1 \mathscr{U} q_2$ and the definition of $\mathscr{U}$ there is $a \in \Sigma$ such that $q_2 \notin \mathrm{pre}_a(Q)$ and $q_1 \in \mathrm{pre}_a(Q)$. With $q_1 \mathscr{S} q_2$ we get $q_2 \in \mathscr{S} \circ \mathrm{pre}_a(Q)$. With the hypothesis that $\mathscr{S}$ is a simulation, we get $q_2 \in \mathrm{pre}_a \circ \mathscr{S}(Q)$ and thus $q_2 \in \mathrm{pre}_a(Q)$, since $\mathscr{S}(Q) \subseteq Q$, which contradicts $q_2 \notin \mathrm{pre}_a(Q)$.

3. This a direct consequence of the definition of $\mathscr{U}$.

□

The main idea to obtain efficient algorithms is to consider relations between blocks of states and not merely relations between states. Therefore, we need a characterization of the notion of simulation expressed over blocks.

Proposition 5. *Let* $T = (Q, \Sigma, \to)$ *be a LTS and* $\mathscr{S}$ *be a reflexive and block-definable relation on* Q. *The relation* $\mathscr{S}$ *is a simulation over* T *if and only if:*

$$\forall a \in \Sigma \, \forall q \in Q \, . \, \mathscr{S} \circ \mathrm{pre}_a([q]_{\mathscr{S}}) \subseteq \mathrm{pre}_a \circ \mathscr{S}([q]_{\mathscr{S}}).$$

Proof. If $\mathscr{S}$ is a simulation then, by definition, we have for any $X \subseteq Q$: $\forall a \in \Sigma \, . \, \mathscr{S} \circ \mathrm{pre}_a(X) \subseteq \mathrm{pre}_a \circ \mathscr{S}(X)$. This inclusion is thus also true for $X = [q]_{\mathscr{S}}$. In the other direction, if $\mathscr{S}$ is reflexive and block-definable then for any $q \in Q$ we get: $q \in [q]_{\mathscr{S}}$ and $\mathscr{S}(q) = \mathscr{S}([q]_{\mathscr{S}})$. We thus have:

$$\mathscr{S} \circ \mathrm{pre}_a(q) \subseteq \mathscr{S} \circ \mathrm{pre}_a([q]_{\mathscr{S}}) \subseteq \mathrm{pre}_a \circ \mathscr{S}([q]_{\mathscr{S}}) = \mathrm{pre}_a \circ \mathscr{S}(q)$$

which ends the proof. □

Now, suppose we have a reflexive and block-definable relation $\mathscr{R}$ and we want to remove from $\mathscr{R}$ all couples (q, r) not belonging in a simulation included in $\mathscr{R}$. If $\mathscr{R}$ is not already a simulation, from the last proposition, there are a letter a and a block B of $\mathscr{R}$ such that $\mathscr{R} \circ \mathrm{pre}_a(B) \not\subseteq \mathrm{pre}_a \circ \mathscr{R}(B)$. But we can assume that $\mathscr{R}$ satisfies (2). With $Q = \mathscr{R}(B) \cup \overline{\mathscr{R}}(B)$ we get: $\mathscr{R} \circ \mathrm{pre}_a(B) \subseteq \mathrm{pre}_a(\mathscr{R}(B) \cup \overline{\mathscr{R}}(B))$. This implies the existence of a non empty set $\mathrm{Remove} \triangleq \mathrm{pre}_a(\overline{\mathscr{R}}(B)) \setminus \mathrm{pre}_a(\mathscr{R}(B))$. Let $r \in \mathrm{Remove}$ and $q \in \mathrm{pre}_a(B)$. If $(q, r) \in \mathscr{R}$ we can safely remove (q, r) from $\mathscr{R}$. Why? Because, if we had $(q, r) \in \mathscr{S}$ with $\mathscr{S} \subseteq \mathscr{R}$ a simulation, with $q \in \mathrm{pre}_a(B)$, there would be $q' \in B$ such that $q \in \mathrm{pre}_a(q')$ and thus $r \in \mathscr{S} \circ \mathrm{pre}_a(q')$. But $\mathscr{S}$ being a simulation, this implies

$r \in \mathrm{pre}_a \circ \mathscr{S}(q')$ and thus $r \in \mathrm{pre}_a \circ \mathscr{R}(B)$ since $\mathscr{S} \subseteq \mathscr{R}$ and $q' \in B$. This contradicts $r \in \mathrm{Remove}$. To sum up, we can safely remove (q, r) from $\mathscr{R}$. But can we safely remove $C \times D$ from $\mathscr{R}$ with C the block of $\mathscr{R}$ containing q and D the block of $\mathscr{R}$ containing r? In general, the answer is no. However, the remainder of this section gives, and justifies, sufficient conditions to do so. We begin by the key definition of the paper.

Definition 6. *Let $T = (Q, \Sigma, \to)$ be a LTS and $\mathscr{R}$ be a reflexive and block-definable relation on Q. A* refiner *of $\mathscr{R}$ is a triple $(B, \mathscr{R}_1, \mathscr{R}_2)$ with $\mathscr{R}_1$ and $\mathscr{R}_2$ two relations on Q such that $\mathscr{R}_1$ is block-definable, B is a block of $\mathscr{R}_1$, $\mathscr{R}(B) \subseteq \mathscr{R}_1(B)$ and*

$$\forall a \in \Sigma \,.\, [\mathrm{pre}_a(\mathscr{R}_1(B) \cup \mathscr{R}_2(B))]_{\mathscr{R}} \cup \mathscr{R}(\mathrm{pre}_a(B)) \subseteq \mathrm{pre}_a(\mathscr{R}_1(B) \cup \mathscr{R}_2(B))$$

Let us fix the intuition for the reader. For the above discussion, we take $\mathscr{R}_1 = \mathscr{R}$ and $\mathscr{R}_2 = \overline{\mathscr{R}}$ which allows us to satisfy (under the assumption (2)) all the conditions of the definition of a refiner. However, if we use $\mathscr{R}_1$ and $\mathscr{R}_2$ like that, we will obtain algorithms whose time complexity is $O(|P_{sim}|^2.|{\to}|)$ for Kripke structures. To obtain algorithms in $O(|P_{sim}|.|{\to}|)$ time, still for Kripke structures, we have to consider in $\overline{\mathscr{R}}(B)$ only what is needed and thus to keep $\mathscr{R}_2$ smaller than $\overline{\mathscr{R}}$. The presence of the relation $\mathscr{R}_1$ is due to the management of the different letters of the alphabet for LTS. Note first, that constraining for all the letters in the alphabet the last condition of the definition of a refiner has made it independent of a particular letter. During a main iteration of the algorithm, we consider a relevant refiner. At this stage $\mathscr{R}_1 = \mathscr{R}$. Gradually, as we consider the letters involved in the transitions leading to $\mathscr{R}_2(B)$, $\mathscr{R}$ is refined and thus $\mathscr{R}(B)$ stays included in $\mathscr{R}_1(B)$ but may becomes smaller than $\mathscr{R}_1(B)$.

The first inclusion of the last condition of a refiner, $[\mathrm{pre}_a(\mathscr{R}_1(B) \cup \mathscr{R}_2(B))]_{\mathscr{R}} \subseteq \mathrm{pre}_a(\mathscr{R}_1(B) \cup \mathscr{R}_2(B))$, authorizes to split the blocks of P either with $\mathrm{pre}_a(\mathscr{R}_1(B))$ or with $\mathrm{Remove}_{a,ref} = \mathrm{pre}_a(\mathscr{R}_2(B)) \setminus \mathrm{pre}_a(\mathscr{R}_1(B))$ like in the next definition. The former induces algorithms that run in $O(|P_{sim}|^2.|{\to}|)$ time. The latter authorizes a run in $O(|P_{sim}|.|{\to}|)$ time. Let $\mathscr{R}'$ be the relation issued from the split. The second inclusion of the last condition of a refiner, $\mathscr{R}(\mathrm{pre}_a(B)) \subseteq \mathrm{pre}_a(\mathscr{R}_1(B) \cup \mathscr{R}_2(B))$, enables to soundly refine $\mathscr{R}'$ by $[\mathrm{pre}_a(B)]_{\mathscr{R}'} \times [\mathrm{Remove}_{a,ref}]_{\mathscr{R}'}$. The remainder of the section formalizes this approach.

Definition 7. *Let $T = (Q, \Sigma, \to)$ be a LTS, $\mathscr{R}$ be a reflexive and block-definable relation on Q, $ref = (B, \mathscr{R}_1, \mathscr{R}_2)$ be a refiner of $\mathscr{R}$ and $a \in \Sigma$ be a letter. We define:*

$$\mathrm{Remove}_{a,ref} \triangleq \mathrm{pre}_a(\mathscr{R}_2(B)) \setminus \mathrm{pre}_a(\mathscr{R}_1(B))$$

$$\mathrm{SplitDelete}_{a,ref}(\mathscr{R}) \triangleq \bigcup_{q \in \mathrm{Remove}_{a,ref}} \begin{array}{l}([q]_{\mathscr{R}} \setminus \mathrm{Remove}_{a,ref}) \times \\ ([q]_{\mathscr{R}} \cap \mathrm{Remove}_{a,ref})\end{array}$$

$$\mathrm{SplitRefine}_{a,ref}(\mathscr{R}) \triangleq \mathscr{R} \setminus \mathrm{SplitDelete}_{a,ref}(\mathscr{R})$$

$$\mathrm{Delete}_{a,ref}(\mathscr{R}) \triangleq \bigcup_{\substack{q \in \mathrm{Remove}_{a,ref} \\ q' \in \mathrm{pre}_a(B)}} \begin{array}{l}[q']_{\mathrm{SplitRefine}_{a,ref}(\mathscr{R})} \times \\ [q]_{\mathrm{SplitRefine}_{a,ref}(\mathscr{R})}\end{array}$$

$$\mathrm{Refine}_{a,ref}(\mathscr{R}) \triangleq \mathrm{SplitRefine}_{a,ref}(\mathscr{R}) \setminus \mathrm{Delete}_{a,ref}(\mathscr{R})$$

We should now prove that if a simulation $\mathscr{S}$ is included in $\mathscr{R}$ it is still included in $\text{Refine}_{a,ref}(\mathscr{R})$. Unfortunately, we do not know how to do that. However, if, instead of simply asking $\mathscr{S}$ to be included in $\mathscr{R}$, we ask $\mathscr{R}$ to be *$\mathscr{S}$-stable* (see next definition), everything works nicely.

Definition 8. *Let $\mathscr{R}$ and $\mathscr{S}$ be two relations on Q. The relation $\mathscr{R}$ is said* $\mathscr{S}$-stable *if $\mathscr{S} \circ \mathscr{R} \subseteq \mathscr{R}$.*

Obviously, if a reflexive relation $\mathscr{R}$ is $\mathscr{S}$-stable then $\mathscr{S}$ is included in $\mathscr{R}$. Intuitively, the $\mathscr{S}$-stability of $\mathscr{R}$ is required by the fact that $\mathscr{R}$ is no longer supposed to be a preorder but since it contains a transitive simulation (as we will see, the coarsest simulation included in a preorder is a preorder and thus transitive) it should be transitive "with" that simulation.

Theorem 9. *Let $T = (Q, \Sigma, \rightarrow)$ be a LTS, $\mathscr{R}$ be a reflexive and block-definable relation on Q, $\mathscr{S}$ be a simulation over T, $a \in \Sigma$ be a letter and $ref = (B, \mathscr{R}_1, \mathscr{R}_2)$ be a refiner of $\mathscr{R}$ such that $\mathscr{R}$ and $\mathscr{R}_1$ are $\mathscr{S}$-stable. Let $\mathscr{U} = \text{Refine}_{a,ref}(\mathscr{R})$. Then, $\mathscr{U}$ is a reflexive, block-definable and $\mathscr{S}$-stable relation. Furthermore, we have:* $[\text{pre}_a(\mathscr{R}_1(B))]_{\mathscr{U}} \cup \mathscr{U}(\text{pre}_a(B)) \subseteq \text{pre}_a(\mathscr{R}_1(B))$.

For the proof, we first need a lemma.

Lemma 10. *Let $T = (Q, \Sigma, \rightarrow)$ be a LTS and, $\mathscr{R}$ and $\mathscr{S}$ be two relations on Q such that $\mathscr{S}$ is a simulation over T and $\mathscr{R}$ is $\mathscr{S}$-stable. Then:*

$$\mathscr{S} \circ \text{pre}_a \circ \mathscr{R} \subseteq \text{pre}_a \circ \mathscr{R}$$

Said otherwise, $\text{pre}_a \circ \mathscr{R}$ *is $\mathscr{S}$-stable.*

Proof. Since $\mathscr{S}$ is a simulation, we have $\mathscr{S} \circ \text{pre}_a \subseteq \text{pre}_a \circ \mathscr{S}$ and thus: 1) $\mathscr{S} \circ \text{pre}_a \circ \mathscr{R} \subseteq \text{pre}_a \circ \mathscr{S} \circ \mathscr{R}$. With the hypothesis that $\mathscr{R}$ is $\mathscr{S}$-stable, we get: 2) $\text{pre}_a \circ \mathscr{S} \circ \mathscr{R} \subseteq \text{pre}_a \circ \mathscr{R}$. Inclusions 1) and 2) put together imply the claimed property. □

Proof of Theorem 9. The fact that $\mathscr{U}$ is reflexive and block-definable is an easy consequence of its definition: from a reflexive and block-definable relation, $\mathscr{R}$, we split some blocks, then we delete some relations between <u>different</u> blocks (after SplitRefine, we have: $[\text{pre}_a(B)]_{\text{SplitRefine}_{a,ref}(\mathscr{R})} \cap [\text{Remove}_{a,ref}]_{\text{SplitRefine}_{a,ref}(\mathscr{R})} = \emptyset$).

For the $\mathscr{S}$-stability of $\mathscr{U}$, let us first remark another direct consequence of the definitions of $\text{Remove}_{a,ref}$ and $\text{SplitRefine}_{a,ref}$:

$$q \in \text{Remove}_{a,ref} \Rightarrow [q]_{\text{SplitRefine}_{a,ref}(\mathscr{R})} \subseteq \text{Remove}_{a,ref} \tag{3}$$

If $\mathscr{U}$ is not $\mathscr{S}$-stable, there are three states $q_1, q_2, q_3 \in Q$ such that $q_1 \,\mathscr{U}\, q_2 \wedge q_2 \,\mathscr{S}\, q_3 \wedge \neg\, q_1 \,\mathscr{U}\, q_3$. We need the following property:

$$q_3 \in \text{Remove}_{a,ref} \Rightarrow q_2 \notin \text{pre}_a(\mathscr{R}_1(B)) \tag{4}$$

Suppose $q_3 \in \text{Remove}_{a,ref}$ and $q_2 \in \text{pre}_a(\mathscr{R}_1(B))$. Since B is a block of $\mathscr{R}_1$ and $\mathscr{R}_1$ is block-definable, for any $q \in B$ we have $q_3 \in \mathscr{S} \circ \text{pre}_a \circ \mathscr{R}_1(q)$. With the hypothesis that $\mathscr{R}_1$ is $\mathscr{S}$-stable, Lemma 10 implies $q_3 \in \text{pre}_a \circ \mathscr{R}_1(q)$ which contradicts $q_3 \in \text{Remove}_{a,ref}$.

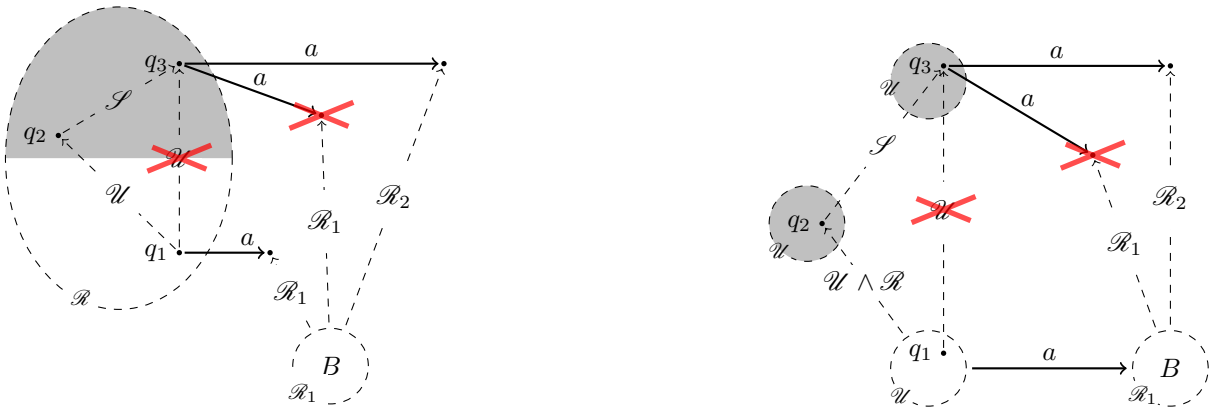


Figure 1: $\mathscr{S}$-stability of the refinement

Now, by construction, $\mathscr{U}$ is included in $\mathscr{R}$ and, by hypothesis, $\mathscr{R}$ is $\mathscr{S}$-stable, then from $q_1 \,\mathscr{U}\, q_2 \wedge q_2 \,\mathscr{S}\, q_3$, and thus $q_1 \,\mathscr{R}\, q_2 \wedge q_2 \,\mathscr{S}\, q_3$, we get: $q_1 \,\mathscr{R}\, q_3$. With $\neg\, q_1 \,\mathscr{U}\, q_3$, we necessarily have $(q_1, q_3) \in \text{SplitDelete}_{a,ref}(\mathscr{R})$ or $(q_1, q_3) \in \text{Delete}_{a,ref}(\mathscr{R})$. Let us consider the two cases (also depicted in Figure 1):

- $(q_1, q_3) \in \text{SplitDelete}_{a,ref}(\mathscr{R})$. This implies the existence of $q \in \text{Remove}_{a,ref}$ such that $q_1, q_3 \in [q]_{\mathscr{R}}$, $q_1 \notin \text{Remove}_{a,ref}$ and $q_3 \in \text{Remove}_{a,ref}$. Since $\mathscr{R}$ is reflexive and $\mathscr{S}$-stable, we have $\mathscr{S} \subseteq \mathscr{R}$. With the fact that, by construction, $\mathscr{U} \subseteq \mathscr{R}$, from $q_1 \,\mathscr{U}\, q_2 \wedge q_2 \,\mathscr{S}\, q_3$ we get $q_1 \,\mathscr{R}\, q_2 \wedge q_2 \,\mathscr{R}\, q_3$. With $q_1, q_3 \in [q]_{\mathscr{R}}$ and the fact that $\mathscr{R}$ is block-definable we get: $q_2 \in [q]_{\mathscr{R}}$. With $q \in \text{pre}_a(\mathscr{R}_2(B))$ and the fact that $(B, \mathscr{R}_1, \mathscr{R}_2)$ is a refiner of $\mathscr{R}$ we have, by definition: $q_2 \in \text{pre}_a(\mathscr{R}_1(B) \cup \mathscr{R}_2(B))$. With (4) we necessarily have $q_2 \in \text{pre}_a(\mathscr{R}_2(B)) \setminus \text{pre}_a(\mathscr{R}_1(B)) = \text{Remove}_{a,ref}$. With $q_1 \notin \text{Remove}_{a,ref}$ and $q_1, q_2 \in [q]_{\mathscr{R}}$ this would imply $(q_1, q_2) \in \text{SplitDelete}_{a,ref}(\mathscr{R})$ and would contradict the fact that $q_1 \,\mathscr{U}\, q_2$.

- $(q_1, q_3) \in \text{Delete}_{a,ref}(\mathscr{R})$. This implies the existence of $q_1' \in \text{pre}_a(B)$ and $q_3' \in \text{Remove}_{a,ref}$ such that $q_1 \in [q_1']_{\text{SplitRefine}_{a,ref}(\mathscr{R})}$ and $q_3 \in [q_3']_{\text{SplitRefine}_{a,ref}(\mathscr{R})}$. From (3) we get $q_3 \in \text{Remove}_{a,ref}$. From $q_1 \,\mathscr{R}\, q_2$, the fact that $\mathscr{R}$ is block-definable and $q_1 \in [q_1']_{\text{SplitRefine}_{a,ref}(\mathscr{R})}$, thus $q_1' \in [q_1]_{\mathscr{R}}$ since $\text{SplitRefine}_{a,ref}(\mathscr{R}) \subseteq \mathscr{R}$, we get $q_1' \,\mathscr{R}\, q_2$. With $q_1' \in \text{pre}_a(B)$ we have $q_2 \in \mathscr{R}(\text{pre}_a(B))$. With the fact that $(B, \mathscr{R}_1, \mathscr{R}_2)$ is a refiner of $\mathscr{R}$ we have, by definition, $q_2 \in \text{pre}_a(\mathscr{R}_1(B) \cup \mathscr{R}_2(B))$. With (4) we necessarily have $q_2 \in \text{Remove}_{a,ref}$. With $q_1' \in \text{pre}_a(B)$ and $q_1 \in [q_1']_{\text{SplitRefine}_{a,ref}(\mathscr{R})}$ this implies $(q_1, q_2) \in \text{Delete}_{a,ref}(\mathscr{R})$ and contradicts the fact that $q_1 \,\mathscr{U}\, q_2$.

Both cases lead to a contradiction. The relation $\mathscr{U}$ is thus $\mathscr{S}$-stable.

Let us now prove the last property:

- $[\text{pre}_a(\mathscr{R}_1(B))]_{\mathscr{U}} \subseteq \text{pre}_a(\mathscr{R}_1(B))$. Let $q \in [\text{pre}_a(\mathscr{R}_1(B))]_{\mathscr{U}}$. There is $q' \in \text{pre}_a(\mathscr{R}_1(B))$ such that $q \in [q']_{\mathscr{U}}$. Since $\mathscr{U} \subseteq \mathscr{R}$ and $(B, \mathscr{R}_1, \mathscr{R}_2)$ is a refiner of $\mathscr{R}$ then $q \in \text{pre}_a(\mathscr{R}_1(B) \cup \mathscr{R}_2(B))$. If $q \notin \text{pre}_a(\mathscr{R}_1(B))$ then $q \in \text{Remove}_{a,ref}$, which implies $(q', q) \in \text{SplitDelete}_{a,ref}(\mathscr{R})$ and contradicts $q' \,\mathscr{U}\, q$.

- $\mathscr{U}(\text{pre}_a(B)) \subseteq \text{pre}_a(\mathscr{R}_1(B))$. Let $q \in \mathscr{U}(\text{pre}_a(B))$. There is $q' \in \text{pre}_a(B)$ such that $q' \,\mathscr{U}\, q$. Since $\mathscr{U} \subseteq \mathscr{R}$ and $(B, \mathscr{R}_1, \mathscr{R}_2)$ is a refiner of $\mathscr{R}$ then $q \in \text{pre}_a(\mathscr{R}_1(B) \cup$

$\mathscr{R}_2(B)$). If $q \notin \mathrm{pre}_a(\mathscr{R}_1(B))$ then $q \in \mathrm{Remove}_{a,ref}$, which implies $(q', q) \in \mathrm{Delete}_{a,ref}(\mathscr{R})$ and contradicts $q' \mathscr{U} q$.

□

4 Base Algorithm

Function `Split`$(Remove, P)$

```
1  SplitCouples := ∅; Touched := ∅; BlocksInRemove := ∅;
2  forall r ∈ Remove do
3      Touched := Touched ∪ {r.block};
4  forall C ∈ Touched do
5      if C ⊆ Remove then
6          BlocksInRemove := BlocksInRemove ∪ {C};
7      else  //C must be split
8          D := C ∩ Remove; P := P ∪ {D};
9          BlocksInRemove = BlocksInRemove ∪ {D};
10         C := C \ Remove;
11         // Only C is modified, not C.Rel or C.NotRel;
12         D.Rel := copy(C.Rel);
13         D.NotRel := copy(C.NotRel);
14         forall q ∈ D do q.block := D;
15         SplitCouples := SplitCouples ∪ {(C, D)};
16 forall (C, D) ∈ SplitCouples, E ∈ P do
17     if C ∈ E.Rel then
18         E.Rel := E.Rel ∪ {D};
19 return (P, BlocksInRemove, SplitCouples)
```

Function `Init`(T, P_{init}, R_{init}) with $T = (Q, \Sigma, \rightarrow)$

```
1  P := copy(P_init); S := ∅;
2  forall a ∈ Σ do a.Remove := ∅;
3  forall B ∈ P do
4      B.Rel := {C ∈ P | (B, C) ∈ R_init};
5  forall q -a-> q' ∈ → do
6      a.Remove := a.Remove ∪ {q};
7  forall a ∈ Σ do
8      (P, BlocksInRemove, _) := Split(a.Remove, P);
9      forall C ∈ BlocksInRemove, D ∈ P do
10         if D ∉ BlocksInRemove then
11             C.Rel := C.Rel \ {D};
12 forall C ∈ P do
13     C.NotRel := ∪{D ∈ P | D ∉ C.Rel};
14     if C.NotRel ≠ ∅ then S := S ∪ {C};
15 return (P, S)
```

Function $\texttt{Sim}(T, P_{init}, R_{init})$ with $T = (Q, \Sigma, \to)$

1 $(P, S) := \texttt{Init}(T, P_{init}, R_{init})$;
2 **forall** $a \in \Sigma$ **do** $\{a.\mathsf{PreB} := \emptyset; a.\mathsf{Remove} := \emptyset\}$;
3 $alph := \emptyset$;
4 **while** $\exists B \in S$ **do**
5 $\quad S := S \setminus \{B\}$;
6 $\quad$ // Assert : $alph = \emptyset \wedge (\forall a \in \Sigma \,.\, a.\mathsf{PreB} = \emptyset \wedge a.\mathsf{Remove} = \emptyset)$
7 $\quad$ **forall** $r \xrightarrow{a} (B.\mathsf{NotRel}) \wedge r \notin \mathrm{pre}_a(\cup B.\mathsf{Rel})$ **do**
8 $\quad\quad alph := alph \cup \{a\}$;
9 $\quad\quad a.\mathsf{Remove} := a.\mathsf{Remove} \cup \{r\}$;
10 $\quad B.\mathsf{NotRel} := \emptyset$;
11 $\quad$ **forall** $q \xrightarrow{a} B \wedge a \in alph$ **do**
12 $\quad\quad a.\mathsf{PreB} := a.\mathsf{PreB} \cup \{q\}$;
13 $\quad$ **forall** $a \in alph$ **do**
14 $\quad\quad (P, BlocksInRemove, SplitCouples) := \texttt{Split}(a.\mathsf{Remove}, P)$;
15 $\quad\quad$ **forall** $(C, D) \in SplitCouples$ **do**
16 $\quad\quad\quad C.\mathsf{Rel} := C.\mathsf{Rel} \setminus \{D\}$; $C.\mathsf{NotRel} := C.\mathsf{NotRel} \cup D$;
17 $\quad\quad\quad S := S \cup \{C\}$;
18 $\quad\quad$ **forall** $\begin{array}{l} D \in BlocksInRemove, \\ C \in \{q.\mathsf{block} \in P \,\big|\, q \in a.\mathsf{PreB}\} \end{array}$ **do**
19 $\quad\quad\quad$ **if** $D \in C.\mathsf{Rel}$ **then**
20 $\quad\quad\quad\quad C.\mathsf{Rel} := C.\mathsf{Rel} \setminus \{D\}$; $C.\mathsf{NotRel} := C.\mathsf{NotRel} \cup D$;
21 $\quad\quad\quad\quad S := S \cup \{C\}$;
22 $\quad$ **forall** $a \in alph$ **do** $\{a.\mathsf{PreB} := \emptyset; a.\mathsf{Remove} := \emptyset\}$;
23 $\quad alph := \emptyset$;
24 $P_{sim} := P$; $R_{sim} := \{(B, C) \in P \times P \,\big|\, C \in B.\mathsf{Rel}\}$;
25 **return** (P_{sim}, R_{sim})

Given a LTS $T = (Q, \Sigma, \rightarrow)$ and an initial antisymmetric partition-relation pair (P_{init}, R_{init}), inducing a preorder $\mathscr{R}_{init}$, the algorithm manipulates relevant refiners to iteratively refine (P, R) initially set to (P_{init}, R_{init}). At the end, (P, R) represents (P_{sim}, R_{sim}) the partition-relation pair whose induced relation $\mathscr{R}_{sim}$ is the coarsest simulation included in $\mathscr{R}_{init}$.

The partition P is a set of blocks. To represent R, we simply associate to each block $B \in P$ a set $B.\mathsf{Rel} \subseteq P$ such that $R = \cup_{B \in P}\{B\} \times B.\mathsf{Rel}$. A block is assimilated with its set of states. For a given state $q \in Q$, the block of P which contains q is noted $q.\mathsf{block}$. We also associate to a block B a set $B.\mathsf{NotRel}$ included in the complement of $\cup B.\mathsf{Rel}$. The refiners will be of the form: $(B, B \times (\cup B.\mathsf{Rel}), B \times (B.\mathsf{NotRel}))$.

The algorithm is decomposed in three functions: `Split`, `Init` and `Sim`, the main one. The function `Split`$(Remove, P)$, used by the two others, splits, possibly, the blocks touched by Remove and returns the updated partition, the list of blocks included in Remove and the list of block couples issued from a split. This last list permits the SplitRefine$_{a,ref}$ of Definition 7 during the **forall** loop at line 15 of `Sim`. Note that, even if `Split` possibly modify the current partition-relation pair, thanks to lines 12 and 16–18 it does not modify the induced relation. This is done out of `Split`, in `Sim`.

The main role of function `Init` is to transform the initial partition-relation pair to a partition-relation pair whose induced relation satisfies condition (2). It also initializes the set S to those B's whose $B.\mathsf{NotRel}$ is not empty.

After the initialization, the `Sim` function mainly executes the following loop. As long as there is a block B whose $B.\mathsf{NotRel}$ is not empty, all non empty $a.\mathsf{Remove}$ sets are computed by only one (this is important for the time efficiency) scan of the transitions leading into $B.\mathsf{NotRel}$. Each of them corresponds to a Remove$_{a,ref}$ of Definition 7. The relevant $\mathrm{pre}_a(B)$, encoded by $a.\mathsf{PreB}$, are also computed by only one (idem) scan of the transitions leading into $\mathrm{pre}_a(B)$. Then, for each letter, with a non empty $a.\mathsf{Remove}$, a refinement step is executed with the refiner $(B, B \times (\cup B.\mathsf{Rel}), B \times (B.\mathsf{NotRel}))$. Note that, during a refinement step, each time a relation (C, D) is removed from R, the content of D is added to $C.\mathsf{NotRel}$. This is done in order to preserve the second invariant of Lemma 12. The remainder of the section validates the algorithm.

Lemma 11. *Let $T = (Q, \Sigma, \rightarrow)$ be a LTS and (P_{init}, R_{init}) be a partition-relation pair over Q inducing a preorder $\mathscr{R}_{init}$. Let $(P, \_) =$ `Init`(T, P_{init}, R_{init}) and $\mathscr{R} = \cup_{G \in P} G \times (\cup G.\mathsf{Rel})$. Then, $\mathscr{R} = \mathrm{InitRefine}(\mathscr{R}_{init})$. Furthermore, for all $G \in P$, we have: $G.\mathsf{NotRel} = Q \setminus \cup G.\mathsf{Rel}$.*

Proof. Unless otherwise specified, all line numbers refer to function `Init`. The purpose of the **forall** loop at line 3 is to associate to each block $B \in P$ the initial set of blocks which simulate it: $B.\mathsf{Rel} = R_{Init}(B)$. In the **forall** loop at line 5 we identify $\mathrm{pre}_a(Q)$, encoded by $a.\mathsf{Remove}$, for each letter $a \in \Sigma$. Then, in the **forall** loop at line 7, for each relevant letter $a \in \Sigma$:

- We split each block $B \in P$ in two parts, $B \cap \mathrm{pre}_a(Q)$ and $B \setminus \mathrm{pre}_a(Q)$, and we update R such that the induced relation of (P, R) stays the same (lines 11–12 and 16–18 of function `Split`).
- Now, each block of P is either included in $\mathrm{pre}_a(Q)$ or disjoint from it. We then delete from R all couple (C, D) such that C is included in $\mathrm{pre}_a(Q)$ and D is disjoint from $\mathrm{pre}_a(Q)$.

At the end $\mathscr{R}$, the induced relation of (P, R), is $\mathscr{R}_{init}$ where all couples (q, q') such that $q \in \mathrm{pre}_a(Q)$ and $q' \notin \mathrm{pre}_a(Q)$ have been deleted. Otherwise said $\mathscr{R} = \mathrm{InitRefine}(\mathscr{R}_{init})$. Then, the **forall** loop at line 12 implies $C.\mathsf{NotRel} = Q \setminus \cup C.\mathsf{Rel}$ for all node $C \in P$. □

Lemma 12. *Let $T = (Q, \Sigma, \to)$ be a LTS, (P_{init}, R_{init}) be an initial partition-relation pair over Q inducing a preorder $\mathscr{R}_{init}$ and $\mathscr{S}$ be a simulation over T such that $\mathscr{S} \subseteq \mathscr{R}_{init}$. Let $\mathscr{R} = \cup_{G \in P} G \times (\cup G.\mathsf{Rel})$. Then, the following properties are invariants of the* **while** *loop of function* `Sim`*:*

1. *$\mathscr{R}$ is a reflexive, block-definable and $\mathscr{S}$-stable relation,*
2. *$\forall G \in P \ \forall c \in \Sigma \ . \ [\mathrm{pre}_c(\cup G.\mathsf{Rel} \cup G.\mathsf{NotRel})]_{\mathscr{R}} \cup \mathscr{R}(\mathrm{pre}_c(G)) \subseteq \mathrm{pre}_c(\cup G.\mathsf{Rel} \cup G.\mathsf{NotRel})$*

Proof. The proof is done by an induction on the iterations of the loop. The two properties are true just after the initialization. For the first one, from Lemmas 4 and 11, we deduce that $\mathscr{S} \subseteq \mathscr{R}$, $\mathscr{R}$ is a preorder and thus reflexive and block-definable. With the fact that a preorder is, by definition, transitive we deduce that $\mathscr{S} \subseteq \mathscr{R}$ implies the $\mathscr{S}$-stability of $\mathscr{R}$. For the second one, this is a direct consequence of item 3 of Lemma 4 and the fact that just after the initialization: $\cup G.\mathsf{Rel} \cup G.\mathsf{NotRel} = Q$, see Lemma 11, for all block $G \in P$.

Let us consider an iteration of the loop. For the ease of the demonstration, we prime a variable for its value before the iteration. A value during the iteration is not primed. The two properties are supposed true before the iteration, we show they are still true after. Therefore, we assume:

$$\forall G \in P' \ \forall c \in \Sigma \ . \quad \begin{array}{c} [\mathrm{pre}_c(\cup G.\mathsf{Rel}' \cup G.\mathsf{NotRel}')]_{\mathscr{R}'} \cup \\ \mathscr{R}'(\mathrm{pre}_c(G)) \\ \subseteq \\ \mathrm{pre}_c(\cup G.\mathsf{Rel}' \cup G.\mathsf{NotRel}') \end{array} \tag{5}$$

In this proof, all line numbers, if not stated otherwise, refer to function `Sim`, and B is the block considered at line 4. Let $ref = (B, \mathscr{R}_1, \mathscr{R}_2)$ with $\mathscr{R}_1 = B \times (\cup B.\mathsf{Rel}')$, $\mathscr{R}_2 = B \times (B.\mathsf{NotRel}')$. Then, ref is a refiner of $\mathscr{R}'$. This is due to the following facts:

- the reflexivity of $\mathscr{R}'$ implies that B is a block of $\mathscr{R}_1$,
- $\mathscr{R}'(B) = \mathscr{R}_1(B) \subseteq \mathscr{R}_1(B)$,
- the partitionability of $\mathscr{R}'$ implies the partitionability of $\mathscr{R}_1$,
- from (5) we have:

$$\forall c \in \Sigma \ . \ [\mathrm{pre}_c(\mathscr{R}_1(B) \cup \mathscr{R}_2(B))]_{\mathscr{R}'} \cup \mathscr{R}'(\mathrm{pre}_c(B)) \subseteq \mathrm{pre}_c(\mathscr{R}_1(B) \cup \mathscr{R}_2(B))$$

Clearly, after the first iteration of the **forall** loop at line 13, from Theorem 9, we have $\mathscr{R} = \mathrm{Refine}_{a,ref}(\mathscr{R}')$ and $\mathscr{R}$ is reflexive, block-definable, $\mathscr{S}$-stable and $\mathscr{R} \subseteq \mathscr{R}'$. Therefore, ref is still a refiner of $\mathscr{R}$. The same happens for the successive iterations of the **forall** loop at line 13. The first property of the current lemma is thus true.

To prove the second property of the lemma, we need two intermediate results.

$$\forall F \in P' \setminus \{B\}\ \forall G \in P\ .\ G \subseteq F \Rightarrow \cup F.\mathsf{Rel}' \cup F.\mathsf{NotRel}' = \cup G.\mathsf{Rel} \cup G.\mathsf{NotRel} \quad (6)$$

By an induction on the number of splits from F to G. If there has been no split then $G = F$ and only line 20 can modify $G.\mathsf{Rel}$ or $G.\mathsf{NotRel}$, but such that the expression $\cup G.\mathsf{Rel} \cup G.\mathsf{NotRel}$ stays constant. Suppose the property is true before a split. If that split does not involve G then, thanks to lines 16–18 in `Split`, $\cup G.\mathsf{Rel}$ and $G.\mathsf{NotRel}$ are not modified. If it is a split of G in G_1 and G_2. Then, the split is done such that, function `Split` lines 11–13, $\cup G_i.\mathsf{Rel} = \cup G.\mathsf{Rel}$, $G_i.\mathsf{NotRel} = G.\mathsf{NotRel}$ and, function `Split` lines 16–18, $\mathscr{R}$ is not changed. With the induction hypothesis we get: $\cup F.\mathsf{Rel}' \cup F.\mathsf{NotRel}' = \cup G_i.\mathsf{Rel} \cup G_i.\mathsf{NotRel}$. After the split, only lines 16 and 20 can modify $G_i.\mathsf{Rel}$ or $G_i.\mathsf{NotRel}$, but such that the expression $\cup G_i.\mathsf{Rel} \cup G_i.\mathsf{NotRel}$ stays constant.

$$\forall G \in P\ .\ G \subseteq B \Rightarrow \cup B.\mathsf{Rel}' = \cup G.\mathsf{Rel} \cup G.\mathsf{NotRel} \quad (7)$$

The proof is similar to the previous one except that $B.\mathsf{NotRel}'$ has been emptied at line 10.

Let $G \in P$ after a given iteration of the **forall** loop at line 13. There are two cases:

- There is $F \in P' \setminus \{B\}$ such that $G \subseteq F$. From (5) and (6) we get: $\forall c \in \Sigma\ .\ [\mathrm{pre}_c(\cup G.\mathsf{Rel} \cup G.\mathsf{NotRel})]_{\mathscr{R}'} \cup \mathscr{R}'(\mathrm{pre}_c(G)) \subseteq \mathrm{pre}_c(\cup G.\mathsf{Rel} \cup G.\mathsf{NotRel})$. From the fact that $\mathscr{R} \subseteq \mathscr{R}'$ we obtain: $\forall c \in \Sigma\ .\ [\mathrm{pre}_c(\cup G.\mathsf{Rel} \cup G.\mathsf{NotRel})]_{\mathscr{R}} \cup \mathscr{R}(\mathrm{pre}_c(G)) \subseteq \mathrm{pre}_c(\cup G.\mathsf{Rel} \cup G.\mathsf{NotRel})$.
- $G \subseteq B$. We have two sub cases:
  - $c \in \mathit{alph}$. Let $\mathscr{R}_c$ be the value of $\mathscr{R}$ after the iteration of the **forall** loop at line 13 with $a = c$. From what precede, remember that *ref* is still a refiner of $\mathscr{R}$, and $\mathscr{R}$ and $\mathscr{R}_1$ are still $\mathscr{S}$-stable. Then, from Theorem 9 we get $[\mathrm{pre}_c(\mathscr{R}_1(B))]_{\mathscr{R}_c} \cup \mathscr{R}_c(\mathrm{pre}_c(B)) \subseteq \mathrm{pre}_c(\mathscr{R}_1(B))$. At the end of the iteration of the while loop, we obviously have $\mathscr{R} \subseteq \mathscr{R}_c$. With (7) and $G \subseteq B$ we obtain: $[\mathrm{pre}_c(\cup G.\mathsf{Rel} \cup G.\mathsf{NotRel})]_{\mathscr{R}} \cup \mathscr{R}(\mathrm{pre}_c(G)) \subseteq \mathrm{pre}_c(\cup G.\mathsf{Rel} \cup G.\mathsf{NotRel}))$.
  - $c \notin \mathit{alph}$. In that case, $c.\mathsf{Remove} = \emptyset$, thus $\mathrm{pre}_c(B.\mathsf{NotRel}') \subseteq \mathrm{pre}_c(\cup B.\mathsf{Rel}')$. With (5) we get: $[\mathrm{pre}_c(\cup B.\mathsf{Rel}')]_{\mathscr{R}'} \cup \mathscr{R}'(\mathrm{pre}_c(B)) \subseteq \mathrm{pre}_c(\cup B.\mathsf{Rel}')$. With (7), $\mathscr{R} \subseteq \mathscr{R}'$ and $G \subseteq B$ we obtain: $[\mathrm{pre}_c(\cup G.\mathsf{Rel} \cup G.\mathsf{NotRel})]_{\mathscr{R}} \cup \mathscr{R}(\mathrm{pre}_c(G)) \subseteq \mathrm{pre}_c(\cup G.\mathsf{Rel} \cup G.\mathsf{NotRel}))$.

□

Theorem 13. *Let $T = (Q, \Sigma, \rightarrow)$ be a LTS and (P_{init}, R_{init}) be an initial partition-relation pair over Q inducing a preorder $\mathscr{R}_{init}$. Function* `Sim` *computes the partition-relation pair (P_{sim}, R_{sim}) inducing $\mathscr{R}_{sim}$ the maximal simulation over T contained in $\mathscr{R}_{init}$. Furthermore, $\mathscr{R}_{sim}$ is a preorder.*

Proof. From line 14 of function `Init`, lines 17 and 21 of function `Sim`, a block $G \in P$ is added in S whenever $G.\mathsf{NotRel}$ is not empty. Furthermore, each time a block G is withdrawn from S, line 5, $G.\mathsf{NotRel}$ is emptied, line 10. Therefore, a block G is in S iff $G.\mathsf{NotRel}$ is not empty.

1. **Function `Sim` terminates**. Let $\mathscr{R}' = \cup_{G \in P} G \times (\cup G.\mathsf{Rel} \cup G.\mathsf{NotRel})$. At each iteration of the **while** loop $\mathscr{R}'$ strictly decrease (a not empty $B.\mathsf{NotRel}$ is emptied). Since $\mathscr{R}'$ is a finite set the algorithm terminates necessarily.

2. $\mathscr{R}_{sim}$ **is a simulation**. The algorithm terminates when S is empty. From what precede, at this moment, for all $G \in P$, $G.\mathsf{NotRel} = \emptyset$. With Lemma 12 we get: $\forall G \in P\ \forall a \in \Sigma\ .\ \mathscr{R}_{sim}(\mathrm{pre}_a(G)) \subseteq \mathrm{pre}_a(\mathscr{R}_{sim}(G))$. With Proposition 5 this means that $\mathscr{R}_{sim}$ is a simulation since $\mathscr{R}_{sim}$ is reflexive and block-definable (Lemma 12).

3. $\mathscr{R}_{sim}$ **contains all simulation included in** $\mathscr{R}_{init}$. From Lemma 12 we deduce that for all simulation $\mathscr{S}$ over T included in $\mathscr{R}_{init}$, $\mathscr{R}_{sim}$ is reflexive and $\mathscr{S}$-stable. These two properties imply $\mathscr{S} \subseteq \mathscr{R}_{sim}$.

4. $\mathscr{R}_{sim}$ **is a preorder**. We have seen that $\mathscr{R}_{sim}$ is $\mathscr{S}$-stable for any simulation included in $\mathscr{R}_{init}$. Since $\mathscr{R}_{sim}$ is such a simulation, $\mathscr{R}_{sim}$ is also $\mathscr{R}_{sim}$-stable and thus transitive. This relation being reflexive and transitive is a preorder.

□

5 Complexity

From now on, all space complexities are given in bits. Let X be a set of elements, we qualify an encoding of X as *indexed* if the elements of X are encoded in an array of $|X|$ slots, one for each element. Therefore, an elements of X can be identify with its index in this array. Let $T = (Q, \Sigma, \to)$ be a LTS, an encoding of T is said *normalized* if the encodings of Q, Σ and $\to$ are indexed, a transition is encoded by the index of its source state, the index of its label and the index of its destination state, and if $|Q|$ and $|\Sigma|$ are in $O(|\to|)$. If $|\Sigma|$ is not in $O(|\to|)$, we can restrict it to its useful part $\Sigma' = \{a \in \Sigma \mid \exists q, q' \in Q\ .\ q \xrightarrow{a} q' \in \to\}$ whose size is less than $|\to|$. To do this, we can use hash table techniques, sort the set $\to$ with the keys being the letters labelling the transitions, or more efficiently use a similar technique of that we used in the algorithm to distribute a set of transitions relatively to its labels (see, as an example, the **forall** loop at line 7 of `Sim`). This is done in $O(|\Sigma| + |\to|)$ time and uses $O(|\Sigma|.\log|\Sigma|)$ space. We have recently seen that this may be done in $O(|\to|)$ time, still with $O(|\Sigma|.\log|\Sigma|)$ space, by using a technique presented in [10]. If $|Q|$ is not in $O(|\to|)$ this means there are states that are not involved in any transition. In general, these states are ignored. Indeed, any state can simulate them and they can not simulate any state with an outgoing transition. Therefore, we can restrict Q to its useful part $\{q \in Q \mid \exists q' \in Q\ \exists a \in \Sigma\ .\ q \xrightarrow{a} q' \in \to \vee\ q' \xrightarrow{a} q \in \to\}$ whose size is in $O(|\to|)$. This is done like for Σ.

All encodings of LTS in this paper are assumed to be normalized.

We also assume the encoding of the initial partition-relation pair (P_{init}, R_{init}) to be such that: the encoding of P_{init} is indexed, for each block $B \in P$ scanning of the states in B can be done in $O(|B|)$ time and scanning of $R_{init}(B)$ can be done in $O(|P_{init}|)$ time. Furthermore, for each state $q \in Q$, we assume set $q.\mathsf{block}$ the block of P_{init} to which q belongs.

One difficulty concerning the data structures was to design an efficient encoding of the NotRel's avoiding the need to update them each time a block is split. This led us to design an original encoding of P (encoding which finally happens to be similar to the one designed in [10]). The first two versions of the algorithm essentially differ by the implementation of the test $r \notin \mathrm{pre}_a(\cup B.\mathsf{Rel})$ at line 7; the overall time complexity of the other parts of the algorithm being in $O(|P_{sim}|.|{\to}|)$.

5.1 Hypothesis

In this sub-section, we state the relevant complexity properties of the data structures we use. We postpone on Section 6 the explanations on how we meet these properties.

During an initialization phase, we set the following:

- For each transition $t = q \xrightarrow{a} q'$, we set $t.\mathsf{sl} = q_a \in \mathrm{sl}(\to)$ the state-letter associated with t.
- For each state-letter $q_a \in \mathrm{sl}(\to)$, we set $q_a.\mathsf{state} = q$ and $q_a.\mathsf{post} = \{q \xrightarrow{a} q' \in {\to} \mid q' \in Q\}$ such that scanning the transitions in $q_a.\mathsf{post}$ is done in linear time.
- For each state $q' \in Q$, we set $q'.\mathsf{pre} = \{q \xrightarrow{a} q' \in {\to} \mid q \in Q,\ a \in \Sigma\}$ such that scanning the transitions of this set is done in linear time.

This initialization phase is done in $O(|{\to}|)$ time and uses $O(|{\to}|.\log|{\to}|)$ space.

The partition P is encoded such that adding a new block is done in constant amortized time and scanning the states of a block is done in linear time. The encoding of P uses $O(|P_{sim}|.\log|{\to}|)$ space. Note that the content of all the blocks uses $O(|Q|.\log|Q|)$ space.

The union $B.\mathsf{NotRel}$ of blocks that do not simulate a given $B \in P$ is encoded such that resetting to $\emptyset$ is done in constant time and adding the content of a block is done in constant amortized time (relatively to the number of added blocks) while scanning the states present in the union is done in linear time of the number of states. The encoding of all NotRel's uses $O(|P_{sim}|^2.\log|P_{sim}|)$ space.

The set of blocks, $B.\mathsf{Rel}$, that simulate a given $B \in P$ is encoded such that membership test and removing of a block are done in constant time while adding of a block is done in constant amortized time (relatively to the size of P_{sim}). The encoding of all Rel's is done in $O(|P_{sim}|^2)$ space.

The set S of blocks to be treated by the main loop of the algorithm is encoded such that the emptiness test and the extraction of one element (arbitrarily chosen by the data structure) are done in constant time, and adding an element in S is done in constant amortized time. The encoding of S uses $O(|P_{sim}|.\log|{\to}|)$ space.

The sets *alph*, *SplitCouples*, *Touched* and *BlocksInRemove* are encoded such that adding of an element is done in constant time and, scanning of their elements and resetting to $\emptyset$ are done in linear time (relatively to the number of elements). The encoding of these sets uses $O(|{\to}|.\log|{\to}|)$ space.

For all $a \in \Sigma$, $a.\mathsf{PreB}$ and $a.\mathsf{Remove}$ are encoded such that adding of an element is done in constant time, scanning of their elements and resetting to $\emptyset$ is done in linear time (relatively to the number of elements). The encoding of all $a.\mathsf{PreB}$ and $a.\mathsf{Remove}$ takes $O(|{\to}|.\log|{\to}|)$ space.

Finally, `Split`$(Remove, P)$ is done in $O(|Remove|)$ time.

5.2 Common analysis

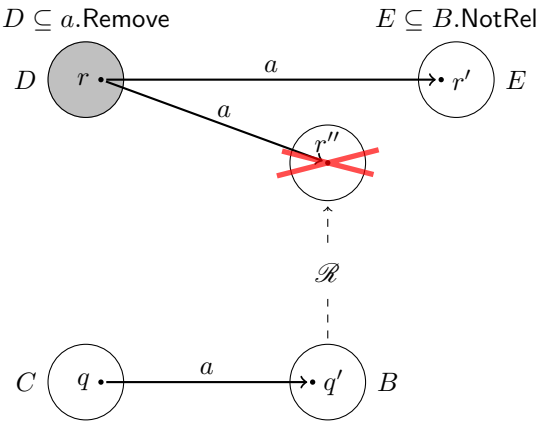


Figure 2: A configuration during a iteration of the while loop

Figure 2 illustrates the main lemma of this section.

Lemma 14. *Let B be a block defined at line 4 of* `Sim`*. During the execution of* `Sim`*, the following configurations can happen at most once at line 4 (line 18 for the last one):*

1. *B and a block E such that $E \subseteq B.\mathsf{NotRel}$,*
2. *B and a transition $r \xrightarrow{a} r'$ such that $r' \in B.\mathsf{NotRel}$,*
3. *B, a block $E \subseteq B.\mathsf{NotRel}$ and a transition $q \xrightarrow{a} q'$ such that $q' \in B$,*
4. *B, a block D and a transition $q \xrightarrow{a} q'$ such that $D \subseteq a.\mathsf{Remove}$ (or $D \in BlocksInRemove$) and $q' \in B$.*

Proof.

1. After the initialization, the content of a block can be added into $B.\mathsf{NotRel}$ only if this block is removed from $\cup B.\mathsf{Rel}$, lines 16 and 20. Furthermore, $\cup B.\mathsf{Rel}$ can only decrease and if E is included in $B.\mathsf{NotRel}$ at line 4, $B.\mathsf{NotRel}$ is emptied at line 10. From what precedes, it will not be possible again for E to be included in $B.\mathsf{NotRel}$ during another iteration of the while loop.

2,3. Direct consequences of the preceding point.

4. Let us suppose this can happen twice. Let $B.\mathsf{Rel}'$ be the value of $B.\mathsf{Rel}$ the first time it happens and $B.Rel''$, $B.\mathsf{NotRel}''$ be the values of $B.\mathsf{Rel}$ and $B.\mathsf{NotRel}$ the second time it happens. With a same reasoning than that of the first point, we get: $B.\mathsf{NotRel}'' \subseteq \cup B.\mathsf{Rel}'$. Let r be any element of D. The first time the configuration happens, we necessarily have, lines 6, 7 and 9: $r \notin \mathrm{pre}_a(\cup B.\mathsf{Rel}')$. The second time the configuration happens we necessarily have: $r \in \mathrm{pre}_a(B.\mathsf{NotRel}'') \subseteq \mathrm{pre}_a(\cup B.\mathsf{Rel}')$ which contradicts $r \notin \mathrm{pre}_a(\cup B.\mathsf{Rel}')$.

□

5.2.1 Time

In this sub-section, the O notation refers to time complexity and the *overall complexity* of a line is the time complexity of all the executions of that line during the lifetime of the algorithm.

Initialisation In this paragraph, we consider the complexity of the initialization. All line numbers refer to function `Init`.

Line 1 essentially corresponds to a copy of P_{init}, this is done in $O(|P_{init}|)$ and thus $O(|P_{sim}|)$. Line 2 is done in $O(|\Sigma|)$ and thus in $O(|{\to}|)$. The **forall** loop at line 3 is done in $O(|P_{init}|^2)$ and thus in $O(|P_{sim}|^2)$. The **forall** loop at line 5 is done in $O(|{\to}|)$. Since we have $\Sigma_{a\in\Sigma}|a.\mathsf{Remove}| \leq |\operatorname{sl}(\to)| \leq |{\to}|$, the overall complexity of line 8 is $O(|{\to}|)$. At first glance the overall complexity of lines 9–11 is in $O(|\Sigma|.|P_{sim}|^2)$ but it is also in $O(|\operatorname{sl}(\to)|.|P_{sim}|)$, since there is at most $|\operatorname{sl}(\to)|$ blocks C concerned by $BlocksInRemove$ and each time it is for a given $a \in \Sigma$, and thus in $O(|{\to}|.|P_{sim}|)$. The **forall** loop at line 12 is done in $O(|P_{sim}|^2)$. From all of that, the complexity of `Init` is $O(|{\to}|.|P_{sim}|)$.

Simulation algorithm Thanks to Lemma 14, item 1, the **while** loop of function `Sim` is executed at most $|P_{sim}|^2$ times since a block B is concerned by the loop only if $B.\mathsf{NotRel} \neq \emptyset$ and $B.\mathsf{NotRel}$ is made of a union of blocks.

The first two versions of the algorithm differ by the test $r \notin \operatorname{pre}_a(\cup B.\mathsf{Rel})$ at line 7. Therefore, in this paragraph, we do not consider the overall complexity of this test. We only consider right now the overall complexity of the scanning of all transitions $r \xrightarrow{a} r'$ such that $r' \in \operatorname{pre}_a(B.\mathsf{NotRel})$ and the overall complexity of lines 8–9. From Lemma 14 item 2 it is $O(|P_{sim}|.|{\to}|)$.

From Lemma 14 item 3, the overall complexity of the `forall` loop at line 11 is $O(|P_{sim}|.|{\to}|)$.

The two preceding paragraphs imply that the overall complexity of lines 22 and 23 is $O(|P_{sim}|.|{\to}|)$ since resetting $a.\mathsf{PreB}$ or $a.\mathsf{Remove}$ is linear in their sizes and thus less than what has been added in them, which is less than $O(|P_{sim}|.|{\to}|)$ (overall complexity of lines 12 and 9).

The overall complexity of line 13 is less than that of line 8 and thus $O(|P_{sim}|.|{\to}|)$.

The complexity of `Split(`$a.\mathsf{Remove}, P$`)` is $O(|a.\mathsf{Remove}|)$. From the time complexity of line 9 we get the overall complexity of line 14: $O(|P_{sim}|.|{\to}|)$.

There is at most $|P_{sim}|$ couples (C, D) issued from the splits of blocks. So, the overall complexity of the **forall** loop at line 15 is $O(|P_{sim}|)$.

From the overall complexity of lines 9 and 12, the overall complexity of the calculation of all D and of all C concerned by line 18 is $O(|P_{sim}|.|{\to}|)$. From Lemma 14 item 4, there is at most $O(|P_{sim}|.|{\to}|)$ couples (C, D) which have been involved at line 18. This implies the overall time complexity of the **forall** loop of line 18: $O(|P_{sim}|.|{\to}|)$.

With what precedes, the test $r \notin \operatorname{pre}_a(\cup B.\mathsf{Rel})$ at line 7 being aside, the overall time complexities of the other lines of the algorithm are all in $O(|P_{sim}|.|{\to}|)$.

5.2.2 Space

Apart from the data structures needed to do the test $r \notin \operatorname{pre}_a(\cup B.\mathsf{Rel})$ at line 7, from Section 5.1, the space complexity of the algorithm is $O(|P_{sim}|^2.\log|P_{sim}| + |{\to}|.\log|{\to}|)$.

5.3 The nice compromise

We use a set of state-letters, $SL \subseteq \text{sl}(\rightarrow)$, with the same time and space complexity properties as those of *alph*. Before line 7 this set is emptied. Then, let us consider a given $r' \in B.\mathsf{NotRel}$. From r' we get, in linear time, all $t = r \xrightarrow{a} r' \in r'.\mathsf{pre}$. If $r_a = t.\mathsf{sl}$ is already in SL it has already been treated, so we stop there and consider the next element of $r'.\mathsf{pre}$. Otherwise, we have not yet tested whether $r \notin \text{pre}_a(\cup B.\mathsf{Rel})$. To do that, first, we add r_a into SL and then, consider all $r \xrightarrow{a} r'' \in r_a.\mathsf{post}$. If for none of them $r''.\mathsf{block}$ is in $B.\mathsf{Rel}$, which is tested each time in constant time, then $r \notin \text{pre}_a(\cup B.\mathsf{Rel})$. Thanks to the use of SL, from Lemma 14, item 1, a transition $r \xrightarrow{a} r'' \in r_a.\mathsf{post}$ is considered only once for a given couple (B, E) of blocks in Figure 2. Therefore, the overall time complexity of the test $r \notin \text{pre}_a(\cup B.\mathsf{Rel})$ in line 7 is $O(|P_{sim}|^2.|\rightarrow|)$.

We can also express the time complexity in another way. For that, we need to introduce the state-letter branching factor of a LTS. The *branching factor of a state* is the number of its outgoing transitions. The *branching factor of a state-letter* q_a is the number of the outgoing transitions of q labelled by a. The *state branching factor* of a LTS is the greatest branching factor of its states. The *state-letter branching factor* of a LTS is the greatest branching factor of its state-letters. Let us come back to the analysis of the complexity of the test $r \xrightarrow{a} (B.\mathsf{NotRel}) \wedge r \notin \text{pre}_a(\cup B.\mathsf{Rel})$. From Lemma 14 item 2, a configuration such that $r \xrightarrow{a} r'$ is a transition with $r' \in B.\mathsf{NotRel}$ happens only once. From this configuration, and with the use of the set SL described above, we have to consider at most b transitions $r \xrightarrow{a} r'' \in r_a.\mathsf{post}$ to test whether $r \notin \text{pre}_a(\cup B.\mathsf{Rel})$.

From what precedes we obtain the following theorem.

Theorem 15. *Let $T = (Q, \Sigma, \rightarrow)$ be a LTS and (P_{init}, R_{init}) be an initial partition-relation pair over Q inducing a preorder $\mathscr{R}_{init}$. The nice compromise version of* `Sim` *computes the partition-relation pair (P_{sim}, R_{sim}) inducing $\mathscr{R}_{sim}$ the maximal simulation over T contained in $\mathscr{R}_{init}$ in:*

- $O(\min(|P_{sim}|, b).|P_{sim}|.|\rightarrow|)$ *time, and*
- $O(|P_{sim}|^2.\log|P_{sim}| + |\rightarrow|.\log|\rightarrow|)$ *space.*

with $b = \max_{q_a \in \text{sl}(\rightarrow)} |\{q \xrightarrow{a} q' \in \rightarrow \mid q' \in Q\}|$ *the* state-letter branching factor *of T.*

Clearly, the state-letter branching factor of a LTS is smaller than its state branching factor. For the state-letter branching factor b in the preceding theorem, we have: $b \leq |Q|$. We also have $|P_{sim}| \leq |Q|$. But there is no definitive comparison between b and $|P_{sim}|$. However if one considers the VLTS Benchmark Suite (`http://cadp.inria.fr/resources/benchmark_`
the state branching factor is rarely more than one hundred even for systems with millions of states. Furthermore, in the case of deterministic systems, the state-letter branching factor is 1. Therefore, we consider this version of `Sim` as a nice compromise between space and time efficiency.

5.4 The Time Efficient Version

To get an efficient time version of the algorithm, we need counters. To each block $B \in P$ we associate $B.\mathsf{RelCount}$, an array of counters indexed on the set of state-letters $\text{sl}(\rightarrow)$ such

that: $B.\mathsf{RelCount}(r_a) = |\{r \xrightarrow{a} r' \in\!\rightarrow \mid r' \in \cup B.\mathsf{Rel} \cup B.\mathsf{NotRel}\}|$. Let $r_a.\mathsf{post} = \{r \xrightarrow{a} r' \in\!\rightarrow \mid r' \in Q\}$. The initialization consists just of setting $B.\mathsf{RelCount}(r_a) = |r_a.\mathsf{post}|$ since at the end of the initialization, for any $B \in P$, $Q = \cup B.\mathsf{Rel} \cup B.\mathsf{NotRel}$. Therefore, the time complexity of the initialization of all the counters is $O(|P_{sim}|.|\operatorname{sl}(\rightarrow)|)$ and thus $O(|P_{sim}|.|\rightarrow|)$.

Let us come back to the overall time complexity of line 7. For each transition $r \xrightarrow{a} r'$ with $r' \in B.\mathsf{NotRel}$, $B.\mathsf{RelCount}(r_a)$ is decremented, and if after that $B.\mathsf{RelCount}(r_a) = 0$ this implies that $r \notin \operatorname{pre}_a(\cup B.\mathsf{Rel})$. This means that the test $r \notin \operatorname{pre}_a(\cup B.\mathsf{Rel})$ is done in constant time for each transition $r \xrightarrow{a} r'$ with $r' \in B.\mathsf{NotRel}$. Note that when a block is split during the function `Split` its array of counters must be copied to the new block. This is done in $O(|\operatorname{sl}(\rightarrow)|)$. Since during the computation there is at most $|P_{sim}|$ splits, the overall time complexity of all the copy operations is $O(|P_{sim}|.|\operatorname{sl}(\rightarrow)|)$ and thus $O(|P_{sim}|.|\rightarrow|)$. We then get the following theorem.

Theorem 16. *Let $T = (Q, \Sigma, \rightarrow)$ be a LTS and (P_{init}, R_{init}) be an initial partition-relation pair over Q inducing a preorder $\mathscr{R}_{init}$. The time efficient version of* `Sim` *computes the partition-relation pair (P_{sim}, R_{sim}) inducing $\mathscr{R}_{sim}$ the maximal simulation over T contained in $\mathscr{R}_{init}$ in:*

$$O(|P_{sim}|.|\rightarrow|) \text{ time and } O(|P_{sim}|.|\operatorname{sl}(\rightarrow)|.\log|Q| + |\rightarrow|.\log|\rightarrow|) \text{ space.}$$

5.5 The Space Efficient Version

The algorithm GPP has a time complexity of $O(|P_{sim}|^2.|\rightarrow|)$, for Kripke structures, but an announced space complexity of $O(|P_{sim}|^2 + |Q|.\log|P_{sim}|)$. Unfortunately, this space complexity does not correspond to that of GPP. As announced in the introduction of the present paper, GPP uses (a modified version of) HHK. For each state $q' \in Q$, HHK uses an array of counters, to speed up the algorithm, and a set of states, Remove(q'), that do not lead via a transition to a state simulating q'. The counters and the Remove sets use $O(|Q|^2.\log|Q|)$ bits. As GPP uses HHK on an abstract structure whose states correspond to blocks of the current partition, the initial version of GPP uses $O(|P_{sim}|^2.\log|P_{sim}|)$ bits for the counters and the Remove sets. Then, the authors explain how to avoid the use of the counters, but do not do the same for the Remove sets. Therefore their algorithm still uses at least $O(|P_{sim}|^2.\log|P_{sim}|)$ bits. The $O(|Q|.\log|P_{sim}|)$ part of their announced space complexity comes from the necessity to memorize for each state q the block to which it belongs ($q.\mathsf{block}$ in the present paper). But GPP, like the algorithm in [4], scan in linear time the states belonging to a block. To do that the set of the states of a block is encoded by a doubly linked list which also enable to remove and to add a state in a block in constant time. This implies that the size of each pointer of these lists need to be sufficient to distinguish the $|Q|$ states: $\log|Q|$. Since there is $|Q|$ states, GPP, like the algorithm in [4], needs at least $|Q|.\log|Q|$ bits. The real space complexity of GPP is thus $O(|P_{sim}|^2.\log|P_{sim}| + |Q|.\log|Q|)$.

By removing the use of the NotRel's in our base algorithm we are able to propose the space efficient version. The time complexities of `Init` and `Split` do not change, but now the overall time complexity of almost all the lines in the **while** loop of `Sim` becomes $O(|P_{sim}|^2.|\rightarrow|)$.

We now present how to avoid, in our nice compromise version of `Sim`, the use of the NotRel's in order to lower the space complexity from $O(|P_{sim}|^2.\log|P_{sim}| + |\rightarrow|.\log|\rightarrow|)$ to $O(|P_{sim}|^2 + |\rightarrow|.\log|\rightarrow|)$ while keeping a time complexity of $O(|P_{sim}|^2.|\rightarrow|)$.

The transformation of the algorithm is quite simple: we mainly replace line 14 of `Init` by "$S := \text{copy}(P)$;", line 7 of `Sim` by "**forall** $r \xrightarrow{a} (\overline{\cup B.\mathsf{Rel}}) \wedge r \notin \text{pre}_a(\cup B.\mathsf{Rel})$ **do**", and we remove all other instructions where NotRel appears. Just to simplify the proof, we also replace line 17 of `Sim` by "$S := S \cup \{C, D\}$;".

In the remainder of this sub-section, lines refer only to the function `Sim`.

Correctness Clearly, after the initialization, which puts us under condition (2), for any block $B \in P$, $(B, B \times (\cup B.\mathsf{Rel}), B \times \overline{\cup B.\mathsf{Rel}})$ is a refiner of the current relation. By noting $B.\mathsf{NotRel} \triangleq \overline{\cup B.\mathsf{Rel}}$, second item of Lemma 12 becomes trivial. For the first item, we follow the same proof. The algorithm terminates since, after the first time, a block B can be chosen again by line 4 only if it is issued from a split (new line 17) or if a block has been removed from $B.\mathsf{Rel}$ (line 21). Each case can happen at most $|P_{sim}|$ times. Let $\mathscr{R}_{sim}$ be the relation induced by the result (P_{sim}, R_{sim}) of `Sim`. Like in the proof of Theorem 13, we use Lemma 12 to deduce that $\mathscr{R}_{sim}$ contains all simulation included in $\mathscr{R}_{init}$ the relation induced by the initial partition-relation pair. Now, for a given block B of the last partition, consider the last time B has been chosen by line 4. After, the execution of the corresponding iteration of the loop, from Theorem 9 we deduce that $\mathscr{R}(\text{pre}_a(B)) \subseteq \text{pre}_a(\cup B.\mathsf{Rel}')$ with $B.\mathsf{Rel}'$ the value of $B.\mathsf{Rel}$ before the iteration. But since this is the last use of B, $B.\mathsf{Rel}$ has not been modified during this iteration of the while loop. Thus, $B.\mathsf{Rel} = B.\mathsf{Rel}'$. From this moment on, $B.\mathsf{Rel}$ will not be modified. So we have $\mathscr{R}_{sim}(\text{pre}_a(B)) \subseteq \text{pre}_a(\mathscr{R}_{sim}(B))$ for all block $B \in P_{sim}$ and all letter $a \in \Sigma$. This defines a simulation (Proposition 5).

Complexity The **forall** loop at line 7 is encoded by the following lines:

forall $q \xrightarrow{a} q' \in\rightarrow$ **do**
  if $q'.\mathsf{block} \in B.\mathsf{Rel}$ **then** $a.\mathsf{PreRel} := a.\mathsf{PreRel} \cup \{q\}$;
forall $q \xrightarrow{a} q' \in\rightarrow$ **do**
  if $q'.\mathsf{block} \notin B.\mathsf{Rel} \wedge q \notin a.\mathsf{PreRel}$ **then**
    $alph := alph \cup \{a\}$;
    $a.\mathsf{Remove} := a.\mathsf{Remove} \cup \{q\}$;

In addition, we add "$a.\mathsf{PreRel} := \emptyset$;" to the bodies of lines 2 and 22. The $a.\mathsf{PreRel}$'s are data structures with the same complexity properties as those of $a.\mathsf{PreB}$ and $a.\mathsf{Remove}$. For a given iteration of the **while** loop the time complexity of these lines is $O(|\rightarrow|)$. Since the number of iterations of the **while** loop is in $O(|P_{sim}|^2)$, the overall time complexity of these lines is $O(|P_{sim}|^2.|\rightarrow|)$. This is thus also the overall time complexity of lines 13, 14, 22 and 23, and the overall time complexity of calculation of all C and D concerned by line 18. The overall time complexity of the **forall** loop at line 15 does not change: $O(|P_{sim}|)$. Consider now Figure 2. A transition $q \xrightarrow{a} q'$ with q' in a block B chosen at line 4 is considered only $O(|P_{sim}|)$ times. Knowing that for each time there is at most $|P_{sim}|$ blocks D in $a.\mathsf{Remove}$ we deduce the overall time complexity of the **forall** loop at line 18: $O(|P_{sim}|^2.|\rightarrow|)$.

Note for the attentive reader: continuing the discussion just before Definition 7, in practice, it should be more interesting to do the split with $\text{pre}_a(\cup B.\mathsf{Rel})$ instead of $a.\mathsf{Remove}$ and to do the refinement step with $BlocksInRemove = \{D \in P \,\big|\, D \cap \text{pre}_a(\cup B.\mathsf{Rel}) = \emptyset\}$. Because, $\text{pre}_a(\cup B.\mathsf{Rel})$ is supposed to decrease at each iteration which is not the case for $a.\mathsf{Remove}$ when we no longer have $B.\mathsf{NotRel}$.

Theorem 17. *Let $T = (Q, \Sigma, \rightarrow)$ be a LTS and (P_{init}, R_{init}) be an initial partition-relation pair over Q inducing a preorder $\mathscr{R}_{init}$. The space efficient version of* `Sim` *computes the partition-relation pair (P_{sim}, R_{sim}) inducing $\mathscr{R}_{sim}$ the maximal forward simulation on T contained in $\mathscr{R}_{init}$ in:*

$$O(|P_{sim}|^2.|\rightarrow|) \text{ time and } O(|P_{sim}|^2 + |\rightarrow|. \log |\rightarrow|) \text{ space.}$$

In the case of Kripke structures, it seems possible to derive a version of the algorithm which still works in $O(|P_{sim}|^2.|\rightarrow|)$ time but uses only $O(|P_{sim}|^2 + |Q|. \log |P_{sim}|)$ space. The idea is to not use the option to scan the states of a block in linear time. The split operation is thus now done in $O(|Q|)$ time and so one. This is just a bit tedious to do. Moreover, in practical cases, the problem is not the $O(|\rightarrow|. \log |\rightarrow|)$ part of our space complexity but the $O(|P_{sim}|^2)$ part.

6 Data structures

In what follows, we use different kinds of data: simple objects, arrays and lists of objects. The size of the pointers has an importance for bit space complexity. It should be enough to differentiate all the considered objects and thus $O(|\rightarrow|)$ for normalized LTS. We call a resizable array an array which double its size when needed. Therefore, adding a new item in this kind of array is done in constant amortized time.

First, we have to set $t.\mathsf{sl}$ for each transition $t \in \rightarrow$, $q_a.\mathsf{state}$ and $q_a.\mathsf{post}$ for each state-letter q_a. To do that we create a new array of transitions, $Post$, as the result of sorting the set of transitions with the labels as keys, then with the source states as keys. We use counting sorts. This means that the two sorts are done in $O(|\rightarrow|)$ time since Q and Σ are in $O(|\rightarrow|)$. Counting sorts are stable. As a result, in $Post$, transitions are packed by source states and within a pack of transition sharing the same source state, there are the sub-packs of transitions sharing the same label. Then, we scan the elements of $Post$ from the first one to the last one. For each transition $t = q \xrightarrow{a} q'$, we consider the couple (q, a) and whenever it changes we create a new state-letter q_a and we set $t.\mathsf{sl} = q_a$. Then, we set $q_a.\mathsf{state} = q$ and $q_a.\mathsf{range} = (idx_{start}, idx_{end})$ with idx_{start} the index in $Post$ of the first transition from q and labelled by a, and idx_{end} the index in $Post$ of the last transition from q labelled by a. Thanks to the two sorts, $(q_a.\mathsf{range}, Post)$ provides an encoding of $q_a.\mathsf{post}$. To represent $q'.\mathsf{pre}$, we just create a new array of transitions, Pre, as the result of sorting, by a counting sort, the set of transitions with destination states as keys. Then, by scanning this array, we associate to each state q' the couple $q'.\mathsf{range} = (idx_{start}, idx_{end})$ with idx_{start} the index of the first transition in Pre having q' as destination state and idx_{end} the index of the last transition in Pre having q' as destination state. Therefore, $(q'.\mathsf{range}, Pre)$ provides an encoding of $q'.\mathsf{pre}$. All of this is done in $O(|\rightarrow|)$ time and uses $O(|\rightarrow|. \log |\rightarrow|)$ space.

We do not encode the content of a block of P by a doubly-linked list of states like the other papers because we need a certain stability property. The problem is the following. let C be a block, we want to be able to add the content of another block D in $C.\mathsf{NotRel}$ in constant time, without scanning the content of D. We also want the encoding of $C.\mathsf{NotRel}$ to be in $O(|P_{sim}|.\log|P_{sim}|)$ space. A first idea is to store in $C.\mathsf{NotRel}$ only the reference of D. But a problem arises when D is split in D_1 and D_2: we have to update all the $C.\mathsf{NotRel}$ and replace D by D_1 and D_2; this implies an overall time complexity of $O(|P_{sim}|^3)$ since there may have $|P_{sim}|$ splits. But $O(|P_{sim}|^3)$ is too much for the time efficient version of our algorithm. A solution is to use a kind of family tree. A block of P is a leave of the tree and is linked to the set of its states. When a block is split, it becomes an internal node of the tree and is no more directly linked to the set of its states but to its two son blocks (which are both linked to their respective set of states). This solution satisfy all the requirements since in a binary tree the number of nodes is at most two times the number of leaves.

However here is a more efficient solution. The set of states Q is copied in a new array Q_p such that the set of states of a block $B \in P$ is arranged in consecutive slots of this array Q_p. Therefore, to memorize the content of B, we just have to memorize $B.\mathsf{start}$ the starting position and $B.\mathsf{end}$ the ending position of the corresponding subarray in Q_p. When a block B is split in two sub-blocks B_1 and B_2, we just arrange the content of the subarray of B in Q_p such that the first slots are for B_1 and the last slots are for B_2. This way, even after several splits, the set of states which once corresponded to a block of P will always be in the same subarray, even if the order of the states is modified. Note that to do the rearrangement, during a split, of the states of B in Q_p we need to memorize for a given state $r \in Q$ its position, $r.\mathsf{posQp}$, in Q_p. See function `SplitImplementation`.

For a given $C \in P$, $C.\mathsf{NotRel}$ may thus be encoded as a set of couples (x, y), which once corresponded to blocks of P, such that x is the start of a subarray in Q_p and y is the end of that subarray. Due to the fact that $B.\mathsf{NotRel} \cap (\cup B.\mathsf{Rel}) = \emptyset$ and when the content of a block is added in $B.\mathsf{NotRel}$ the block is removed from $B.\mathsf{Rel}$, all the blocks encoded in $C.\mathsf{NotRel}$ are different. Therefore, $|C.\mathsf{NotRel}|$ is in $O(|P_{sim}|)$ and thus the encoding of all the NotRel's is done in $O(|P_{sim}|^2.\log(|Q|)$ space. The factor $\log(|Q|)$ is due to the fact that in a couple (x, y), the maximum value for x and y is $|Q|$. However, we want the encoding of all the NotRel's to be in $O(|P_{sim}|^2.\log(|P_{sim}|)$. To do that, remember the family tree mentioned above. The number of past and actual blocks of P is in $O(|P_{sim}|)$. Therefore, we introduce N, a set of nodes. During the initialization, we associate $B.\mathsf{node} \in N$, a node, to each block $B \in P$. The starting and ending position in Q_p corresponding to a block B is not directly store in B but in $B.\mathsf{node}$ via $B.\mathsf{node.start}$ and $B.\mathsf{node.end}$. Therefore, when we want to add the content of a block D in $C.\mathsf{NotRel}$, in fact we add $D.\mathsf{node}$ in $C.\mathsf{NotRel}$. Since $|N|$ is in $O(|P_{sim}|)$, the encoding of all the NotRel's is done in $O(|P_{sim}|^2.\log(|P_{sim}|)$ space. Note that the encoding of all the nodes in N is done in $O(|P_{sim}|.\log(|Q|)$ space. The $O(\log(|Q|)$ factor being for the couple $(n.\mathsf{start}, n.\mathsf{end})$ for each $n \in N$. The set N and the NotRel's are encoded by resizable arrays.

For a given $B \in P$, the set $B.\mathsf{Rel}$ is encoded by a resizable boolean array. To know whether a block C belongs to $B.\mathsf{Rel}$ we check $B.\mathsf{Rel}[C.\mathsf{index}]$ with $C.\mathsf{index}$ the index of C in the array encoding P. The encoding of all Rel's is therefore done in $O(|P_{sim}|^2)$ space.

A given block $B \in P$ is encoded in $O(\log(|\rightarrow|)$ space since we just need a constant number of integers, less than $|\rightarrow|$, or pointers for $B.\mathsf{index}$, $B.\mathsf{node}$, $B.\mathsf{NotRel}$, $B.\mathsf{Rel}$,

B.splitCount (see function `SplitImplementation`) and B.RelCount (for the time efficient version). Thanks to, Q_p, B.node.start and B.node.end scanning of the states contained in a block $B \in P$ is done in linear time. The set P is encoded as a resizable array of blocks. Therefore, the encoding of P is done in $O(|P_{sim}|.\log(|\rightarrow|)$ space and the encoding of the contents of the blocks of P is done in $O(|Q|.\log(|Q|)$.

The set S is encoded as a list of blocks (we could have used a resizable array) but we also need to add a boolean mark to the blocks of P to know whether a given block is already in S. That way, we keep the encoding of S in $O(|P_{sim}|.\log(|\rightarrow|))$ space.

The sets *alph*, *SplitCouples* and $Touched$ are implemented like S: a list and a binary mark on the respective elements. To reset one of these sets, we simply scan the list of elements; for each of them we unset the corresponding mark, then we empty the list. All of this is done in linear time. The maximum sizes for *alph* is $|\Sigma|$, for *SplitCouples* and $Touched$ it is $|P_{sim}|$. Therefore, they are all encoded in $O(|\rightarrow|.\log(|\rightarrow|))$ space.

To represent a set a.PreB or a.Remove with $a \in \Sigma$ we should not use a list of states and a binary array indexed on $|Q|$. This would have implied a total size of $|\Sigma|.|Q|$ for all the letters, which may exceed $|\rightarrow|$. Instead, we use a list of elements of $\mathrm{sl}(\rightarrow)$ per letter and only one common (for all the letters) binary array indexed on $\mathrm{sl}(\rightarrow)$. We also use the fact that for a given $a \in \Sigma$ a state can not belongs to both a.PreB and a.Remove in an iteration of the **while** loop of `Sim`. When we need to add a state r in a.Remove, for example, it is from a transition $r \xrightarrow{a} r'$ issued from a call of r'.pre. This call provides r_a too. Then, we add r_a in the list of a.Remove and we set the mark associate with r_a, only if this mark is not already set. Cleaning of a.Remove is done like cleaning of *alph* (scanning the elements and unsetting the associated marks). Note that we store r_a instead of r in a.Remove, but this is not a problem since r_a.state gives us r. The encoding of all a.PreB and a.Remove is done in $O(|\,\mathrm{sl}(\rightarrow)|.\log|\rightarrow|)$ space and thus in $O(|\rightarrow|.\log|\rightarrow|)$ space.

As denoted by the name, function `SplitImplementation` is an implementation of function `Split` taking into account the new way of encoding the partition. Clearly, a call of `SplitImplementation(`$Remove, P$`)` is done in $O(|Remove|)$ time.

7 Future Works

In order to simplify the presentation, no practical optimization has been proposed. This will be done in a future work with the implementation of the algorithms. For the moment we just recall an easy theoretical optimization: the coarsest bisimulation relation should be computed before, and used by the algorithms computing the coarsest simulation relation. This reduces $\mathrm{sl}(\rightarrow)$, which is really important for the space complexity of the time efficient version of the algorithm, and also reduces the transition relation, which has a positive impact on the time complexity of all the versions of the algorithm.

Concerning the search of the coarsest bisimulation relation in a LTS, the framework presented in the present paper can be adapted. We have recently learned that an algorithm avoiding the effect of the size of the alphabet in the time and space complexities of the bisimulation problem has already been presented by Valmari [10] in 2009. The approach of Valmari is different. His splitters (roughly speaking, they play the same role of our refiners but are adapted for the bisimulation problem) depend conceptually on letters but he uses two partitions of the set of transitions, beside the classical one for the states,

Function `SplitImplementation`$(Remove, P)$

```
1  SplitCouples := ∅; Touched := ∅; BlocksInRemove := ∅;
2  // Assert : ∀C ∈ P . C.splitCount = 0;
3  // When a block is created, all its counters are set to 0.;
4  forall r ∈ Remove do
5      C := r.block;
6      Touched := Touched ∪ {C};
7      oldpos := r.posQp; newpos := C.node.start + C.splitCount;
8      r' := Q_p[newpos];
9      Q_p[newpos] := r; Q_p[oldpos] := r';
10     r.posQp := newpos; r'.posQp := oldpos;
11     C.splitCount := C.splitCount + 1 ;
12 forall C ∈ Touched do
13     if C.splitCount = |C| then
14         BlocksInRemove := BlocksInRemove ∪ {C};
15     else  //C must be splitted
16         D := newBlock(); P := P ∪ {D};
17         D.node := newNode(); N := N ∪ {D.node};
18         BlocksInRemove := BlocksInRemove ∪ {D};
19         D.node.start := C.node.start;
20         D.node.end := C.node.start + C.splitCount − 1;
21         C.node.start := D.node.end + 1;
22         D.Rel := copy(C.Rel);
23         D.NotRel := copy(C.NotRel);
24         SplitCouples := SplitCouples ∪ {(C, D)};
25         forall pos ∈ {D.node.start, ..., D.node.end} do Q_p[pos].block := D;
26     C.splitCount := 0;
27 forall (C, D) ∈ SplitCouples, E ∈ P do
28     if C ∈ E.Rel then
29         E.Rel := E.Rel ∪ {D};
30 return (P, BlocksInRemove, SplitCouples)
```

to avoid the negative effect of the size of the alphabet. At first glance, an adaptation of our present work in the case of bisimulation yields a simpler algorithm than the one of Valmari and, furthermore, closer to the one of Paige and Tajan for Kripke structures [8]. This will be made precise in a future paper.

References

[1] Parosh Aziz Abdulla, Ahmed Bouajjani, Lukás Holík, Lisa Kaati, and Tomás Vojnar. Computing Simulations over Tree Automata. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 93–108. Springer, 2008.

[2] Parosh Aziz Abdulla, Yu-Fang Chen, Lukás Holík, Richard Mayr, and Tomás Vojnar. When Simulation Meets Antichains. In Javier Esparza and Rupak Majumdar, editors, *TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 158–174. Springer, 2010.

[3] Gérard Cécé and Alain Giorgetti. Simulations over two-dimensional on-line tessellation automata. In Giancarlo Mauri and Alberto Leporati, editors, *Developments in Language Theory*, volume 6795 of *Lecture Notes in Computer Science*, pages 141–152. Springer, 2011.

[4] Silvia Crafa, Francesco Ranzato, and Francesco Tapparo. Saving space in a time efficient simulation algorithm. *Fundam. Inform.*, 108(1-2):23–42, 2011.

[5] Raffaella Gentilini, Carla Piazza, and Alberto Policriti. From bisimulation to simulation: Coarsest partition problems. *J. Autom. Reasoning*, 31(1):73–103, 2003.

[6] Monika Rauch Henzinger, Thomas A. Henzinger, and Peter W. Kopke. Computing simulations on finite and infinite graphs. In *FOCS*, pages 453–462. IEEE Computer Society, 1995.

[7] Robin Milner. An Algebraic Definition of Simulation Between Programs. In *IJCAI*, pages 481–489, 1971.

[8] Robert Paige and Robert Endre Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987.

[9] Francesco Ranzato and Francesco Tapparo. An efficient simulation algorithm based on abstract interpretation. *Inf. Comput.*, 208(1):1–22, 2010.

[10] Antti Valmari. Bisimilarity minimization in o(m log n) time. In Giuliana Franceschinis and Karsten Wolf, editors, *Petri Nets*, volume 5606 of *Lecture Notes in Computer Science*, pages 123–142. Springer, 2009.

[11] Rob J. van Glabbeek and Bas Ploeger. Correcting a space-efficient simulation algorithm. In Aarti Gupta and Sharad Malik, editors, *CAV*, volume 5123 of *Lecture Notes in Computer Science*, pages 517–529. Springer, 2008.