Foundation for a Series of Efficient Simulation Algorithms

Gérard Cécé
FEMTO-ST Institute/CNRS, Univ. Bourgogne Franche-Comté, Montbéliard, France
Email: Gerard.Cece@femto-st.fr

***Abstract*—Compute the coarsest simulation preorder included in an initial preorder is used to reduce the resources needed to analyze a given transition system. This technique is applied on many models like Kripke structures, labeled graphs, labeled transition systems or even word and tree automata. Let $(Q,\rightarrow)$ be a given transition system and $\mathscr{R}_{\text{init}}$ be an initial preorder over Q. Until now, algorithms to compute $\mathscr{R}_{\text{sim}}$, the coarsest simulation included in $\mathscr{R}_{\text{init}}$, are either memory efficient or time efficient but not both. In this paper we propose the foundation for a series of efficient simulation algorithms with the introduction of the notion of maximal transitions and the notion of stability of a preorder with respect to a coarser one. As an illustration we solve an open problem by providing the first algorithm with the best published time complexity, $O(|P_{\text{sim}}|.|\rightarrow|)$, and a bit space complexity in $O(|P_{\text{sim}}|^2.\log(|P_{\text{sim}}|)+|Q|.\log(|Q|))$, with P_{sim} the partition induced by $\mathscr{R}_{\text{sim}}$.**

I. Introduction

The simulation relation has been introduced by Milner [1] as a behavioural relation between process. This relation can also be used to speed up the test of inclusion of languages [2] or as a sufficient condition when this test of inclusion is undecidable in general [3]. Another very helpful use of a simulation relation is to exhibit an equivalence relation over the states of a system. This allows to reduce the state space of the given system to be analyzed while preserving an important part of its properties, expressed in temporal logics for examples [4]. Note that the simulation equivalence yields a better reduction of the state space than the better known bisimulation equivalence.

A. State of the Art

The paper that has most influenced the literature is that of Henzinger, Henzinger and Kopke [5]. Their algorithm, designed over Kripke structures, and here named HHK, to compute $\mathscr{R}_{\text{sim}}$, the coarsest simulation, runs in $O(|Q|.|\rightarrow|)$-time, with $\rightarrow$ the transition relation over the state space Q, and uses $O(|Q|^2.\log(|Q|))$ bits.

But it happens that $\mathscr{R}_{\text{sim}}$ is a preorder. And as such, it can be more efficiently represented by a partition-relation pair (P,R) with P a partition, of the state space Q, whose blocks are classes of the simulation equivalence relation and with $R\subseteq P\times P$ a preorder over the blocks of P. Bustan and Grumberg [6] used this to propose an algorithm, here named BG, with an optimal bit-space complexity in $O(|P_{\text{sim}}|^2+|Q|.\log(|P_{\text{sim}}|))$ with $|P_{\text{sim}}|$ (in general significantly smaller than $|Q|$) the number of blocks of the partition P_{sim} associated with $\mathscr{R}_{\text{sim}}$. Unfortunately, BG suffers from a very bad time complexity. Then, Gentilini, Piazza and Policriti [7] proposed an algorithm, here named GPP, with a better time complexity, in $O(|P_{\text{sim}}|^2.|\rightarrow|)$, and a claimed bit space complexity like the one of BG. This algorithm had a mistake and was corrected in [8]. It is very surprising that none of the authors citing [7], including these of [8], [9], [10], [11] and [12], realized that the announced bit space complexity was also not correct. Indeed, as shown in [13] and [14] the real bit space complexity of GPP is $O(|P_{\text{sim}}|^2.\log(|P_{\text{sim}}|)+|Q|.\log(|Q|))$. In a similar way, [10] and [11] did a minor mistake by considering that a bit space in $O(|Q|.\log(|P_{\text{sim}}|))$ was sufficient to represent the partition in their algorithms while a space in $O(|Q|.\log(|Q|))$ is needed.

Ranzato and Tapparo [9], [10] made a major breakthrough with their algorithm, here named RT, which runs in $O(|P_{\text{sim}}|.|\rightarrow|)$-time but uses $O(|P_{\text{sim}}|.|Q|.\log(|Q|))$ bits, which is more than GPP. The difficulty of the proofs and the abstract interpretation framework put aside, RT is a reformulation of HHK but with a partition-relation pair instead of a mere relation between states. Over unlabelled transition systems, this is the best algorithm regarding the time complexity.

Since [9] a question has emerged: is there an algorithm with the time complexity of RT while preserving the space complexity of GPP ?

Crafa, Ranzato and Tapparo [11], modified RT to enhance its space complexity. They proposed an algorithm with a time complexity in $O(|P_{\text{sim}}|.|\rightarrow|+|P_{\text{sim}}|^2.|\rightarrow_{P_{\text{sp}},P_{\text{sim}}}|)$ and a bit space complexity in $O(|P_{\text{sp}}|.|P_{\text{sim}}|.\log(|P_{\text{sp}}|)+|Q|.\log(|Q|))$ with $|P_{\text{sp}}|$ between $|P_{\text{sim}}|$ and $|P_{\text{bis}}|$, the number of bisimulation classes, and $\rightarrow_{P_{\text{sp}},P_{\text{sim}}}$ a smaller abstraction of $\rightarrow$. Unfortunately (although this algorithm provided new insights), for 22 examples, out of the 24 they provided, there is no difference between $|P_{\text{bis}}|$, $|P_{\text{sp}}|$ and $|P_{\text{sim}}|$. For the two remaining examples the difference is marginal. With a little provocation, we can then consider that $|P_{\text{sp}}|\approx|P_{\text{bis}}|$ and compute the bisimulation equivalence (what should be done every time as it produces a considerable speedup) then compute the simulation equivalence with GPP on the obtained system is a better solution than the algorithm in [11] even if an efficient computation of the bisimulation equivalence requires, see [15], a bit space in $O(|\rightarrow|.\log(|Q|))$.

Ranzato [14] almost achieved the challenge by announcing an algorithm with the space complexity of GPP but with the

978-1-5090-3018-7/17/$31.00

time complexity of RT multiplied by a $\log(|Q|)$ factor. He concluded that the suppression of this $\log(|Q|)$ factor seemed to him quite hard to achieve. Gentilini, Piazza and Policriti [12] outperformed the challenge by providing an algorithm with the space complexity of BG and the time complexity of RT, but only in the special case of acyclic transition systems.

B. Our Contributions

In this paper, we respond positively to the question and propose the first simulation algorithm with the time complexity of RT and the space complexity of GPP.

Our main sources of inspiration are [15] for its implicit notion of stability against a coarser partition, that we generalize in the present paper for preorders, and for the counters it uses, [5] for the extension of these counters for simulation algorithms, [6] for its use of little brothers to which we prefer the use of what we define as maximal transitions, [14] for its implicit use of maximal transitions to split blocks and for keeping as preorders the intermediate relations of its algorithm and [13] for its equivalent definition of a simulation in terms of compositions of relations.

Note that almost all simulation algorithms are defined for Kripke structures. However, in each of them, after an initial step which consists in the construction of an initial preorder $\mathscr{R}_{\text{init}}$, the algorithm is equivalent to calculating the coarsest simulation inside $\mathscr{R}_{\text{init}}$ over a classical transition system. We therefore directly start from a transition system $(Q,\rightarrow)$ and an initial preorder $\mathscr{R}_{\text{init}}$ inside which we compute the coarsest simulation.

Remark. The proofs which are not in the main text may be found in [16].

II. Preliminaries

Let Q be a set of elements, or *states*. The number of elements of Q is denoted $|Q|$. A *relation* over Q is a subset of $Q\times Q$. Let $\mathscr{R}$ be a relation over Q. For all $q,q'\in Q$ we may write $q\mathscr{R}q'$, or $q\text{-}\!\!\cdot\!\!\cdot\!\!\mathscr{R}\!\!\cdot\!\!\cdot\!\!\rightarrow q'$ 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 $X\subseteq Q$. We write $X\mathscr{R}Y$, or $X\text{-}\!\!\cdot\!\!\cdot\!\!\mathscr{R}\!\!\cdot\!\!\cdot\!\!\rightarrow Y$ in the figures, when $X\times Y\cap\mathscr{R}\neq\emptyset$. For $q\in Q$ and $X\subseteq Q$, we also write $X\mathscr{R}q$ (resp. $q\mathscr{R}X$) for $X\mathscr{R}\{q\}$ (resp. $\{q\}\mathscr{R}X$). A relation $\mathscr{R}$ is said *coarser* than another relation $\mathscr{R}'$ when $\mathscr{R}'\subseteq\mathscr{R}$. The *inverse* of $\mathscr{R}$ is $\mathscr{R}^{-1}\triangleq\{(y,x)\in Q\times Q\,|\,(x,y)\in\mathscr{R}\}$. The relation $\mathscr{R}$ is said *symmetric* if $\mathscr{R}^{-1}\subseteq\mathscr{R}$ and *antisymmetric* if $q\mathscr{R}q'$ and $q'\mathscr{R}q$ implies $q=q'$. Let $\mathscr{S}$ be a second relation over 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 $q\mathscr{R}q$, and *transitive* if $\mathscr{R}\circ\mathscr{R}\subseteq\mathscr{R}$. A *preorder* is a reflexive and transitive relation. A *partition* P 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* is a pair (P,R) with P a partition and R a relation over P. To a partition-relation pair (P,R) we associate a relation $\mathscr{R}_{(P,R)}\triangleq\bigcup_{(C,D)\in R}C\times D$. Let $\mathscr{R}$ be a preorder on Q and $q\in Q$, we define $[q]_{\mathscr{R}}\triangleq\{q'\in Q\,|\,q\mathscr{R}q'\wedge q'\mathscr{R}q\}$ and $P_{\mathscr{R}}\triangleq\{[q]_{\mathscr{R}}\subseteq Q\,|\,q\in Q\}$. It is easy to show that $P_{\mathscr{R}}$ is a partition of Q. Therefore, given any preorder $\mathscr{R}$ and a state $q\in Q$, we also call *block*, the *block of* q, the set $[q]_{\mathscr{R}}$. A symmetric preorder $\mathscr{P}$ is totally represented by the partition $P_{\mathscr{P}}$ since $\mathscr{P}=\cup_{E\in P_{\mathscr{P}}}E\times E$. Let us recall that a symmetric preorder is traditionally named an *equivalence relation*. Conversely, given a partition P, there is an associated equivalence relation $\mathscr{P}_P\triangleq\cup_{E\in P}E\times E$. In the general case, a preorder $\mathscr{R}$ is efficiently represented by the partition-relation pair $(P_{\mathscr{R}},R_{\mathscr{R}})$ with $R_{\mathscr{R}}\triangleq\{([q]_{\mathscr{R}},[q']_{\mathscr{R}})\in P_{\mathscr{R}}\times P_{\mathscr{R}}\,|\,q\mathscr{R}q'\}$ a reflexive, transitive and antisymmetric relation over $P_{\mathscr{R}}$. Furthermore, for a preorder $\mathscr{R}$, we note $[\cdot]_{\mathscr{R}}$ the relation over Q which associates to a state the elements of its block. Said otherwise: $[\cdot]_{\mathscr{R}}\triangleq\cup_{q\in Q}\{q\}\times[q]_{\mathscr{R}}$. Finally, for a set X of sets we note $\cup X$ for $\cup_{E\in X}E$.

Proposition 1. *Let X and Y be two blocks of a preorder $\mathscr{R}$. Then*

$$(X'\subseteq X\wedge Y'\subseteq Y\wedge X'\mathscr{R}Y')\Rightarrow X\times Y\subseteq\mathscr{R}.$$

Said otherwise, when two subsets of two blocks of $\mathscr{R}$ are related by $\mathscr{R}$ then all the elements of the first block are related by $\mathscr{R}$ with all the elements of the second block.

Proof. Thanks to the transitivity of $\mathscr{R}$. □

A *finite transition systems (TS)* is a pair $(Q,\rightarrow)$ with Q a finite set of states, and $\rightarrow$ a relation over Q called the *transition relation*. A relation $\mathscr{S}$ is a *simulation* over $(Q,\rightarrow)$ if:

$$\mathscr{S}\circ{\rightarrow}^{-1}\subseteq{\rightarrow}^{-1}\circ\mathscr{S}\tag{1}$$

For a simulation $\mathscr{S}$, when we have $q\,\mathscr{S}\,q'$, we say that q is *simulated* by q' (or q' *simulates* q).

A relation $\mathscr{B}$ is a *bisimulation* if $\mathscr{B}$ and $\mathscr{B}^{-1}$ are both simulations. The interesting bisimulations, such as the coarsest one included in a preorder, are equivalence relations. It is easy to show that an equivalence relation $\mathscr{B}$ is a bisimulation iff :

$$\mathscr{B}\circ{\rightarrow}^{-1}\subseteq{\rightarrow}^{-1}\circ\mathscr{B}\tag{2}$$

Remark. The classical definition is to say that a relation $\mathscr{S}$ is a simulation if: $q_1\,\mathscr{S}\,q_2\wedge q_1\rightarrow q_1'\Rightarrow\exists q_2'\,.\,q_2\rightarrow q_2'\wedge q_1'\,\mathscr{S}\,q_2'$. However, we prefer the formula (1), which is equivalent, because it is more global and to design efficient simulation algorithms we must abstract from individual states.

In the remainder of the paper, all relations are over the same finite set Q and the underlying transition system is $(Q,\rightarrow)$.

III. Key Ideas

Let us start from **equation** (1)**.** If a relation $\mathscr{R}$ is not a simulation, we have $\mathscr{R}\circ{\rightarrow}^{-1}\not\subseteq{\rightarrow}^{-1}\circ\mathscr{R}$. This implies the existence of a relation *Remove* such that: $\mathscr{R}\circ{\rightarrow}^{-1}\subseteq({\rightarrow}^{-1}\circ\mathscr{R})\cup\mathit{Remove}$. It can be shown that most of the simulation algorithms cited in the introduction, like HHK, GPP and RT, are based on this last equation. In this paper, like in [13], we make the following different choice. When $\mathscr{R}$ is not a simulation, we reduce the problem of finding the coarsest simulation inside $\mathscr{R}$ to the case where there is a relation *NotRel* such that: $\mathscr{R}\circ{\rightarrow}^{-1}\subseteq{\rightarrow}^{-1}\circ(\mathscr{R}\cup\mathit{NotRel})$. Let us note $\mathscr{U}\triangleq\mathscr{R}\cup\mathit{NotRel}$. We will say that $\mathscr{R}$ is **$\mathscr{U}$-stable** since we have:

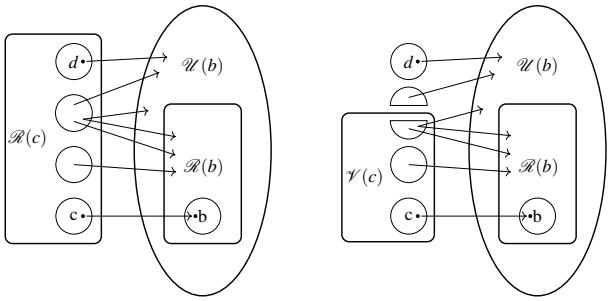

Fig. 1. $\mathscr{R}$ is $\mathscr{U}$-stable and $\mathscr{V}$, obtained after a split of blocks of $\mathscr{R}$ and a refinement of $\mathscr{R}$, is $\mathscr{R}$-stable.

$$\mathscr{R} \circ \rightarrow^{-1} \subseteq \rightarrow^{-1} \circ \mathscr{U} \tag{3}$$

Our definition of stability is new. However, it is implicit in the bisimulation algorithm of [15, p. 979] where, with the notations from [15], a partition Q is said stable with every block of a coarser partition X. Within our formalism we can say the same thing with the formula $\mathscr{P}_Q \circ \rightarrow^{-1} \subseteq \rightarrow^{-1} \circ \mathscr{P}_X$.

Consider the transition $c \rightarrow b$ in Fig. 1. The preorder $\mathscr{R}$ is assumed to be $\mathscr{U}$-stable and we want to find the coarsest simulation included in $\mathscr{R}$. Since $\mathscr{R}$ is a preorder, the set $\mathscr{R}(c)$ is a union of blocks of $\mathscr{R}$. A state d in $\mathscr{R}(c)$ which doesn't have an outgoing transition to $\mathscr{R}(b)$ belongs to $\rightarrow^{-1} \circ \mathscr{U}(b)$, thanks to (3), but cannot simulate c. Thus, we can safely remove it from $\mathscr{R}(c)$. But to do this effectively, we want to manage blocks of states and not the individual states. Hence, we first do a **split step** by splitting the blocks of $\mathscr{R}$ such that a resulting block, included in both $\mathscr{R}(c)$ and $\rightarrow^{-1} \circ \mathscr{U}(b)$, is either completely included in $\rightarrow^{-1} \circ \mathscr{R}(b)$, which means that its elements still have a chance to simulate c, or totally outside of it, which means that its elements cannot simulate c. Let us call $\mathscr{P}$ the equivalence relation associated to the resulting partition. We will say that $\mathscr{P}$ is **$\mathscr{R}$-block-stable**. Then, to test whether a block, E of $\mathscr{P}$, which has an outgoing transition in $\rightarrow^{-1} \circ (\mathscr{U} \setminus \mathscr{R})(b)$, is included in $\rightarrow^{-1} \circ \mathscr{R}(b)$, it is sufficient to do the test **for only one** of its elements, arbitrarily choosen, we call the **representative** of E: $E.\mathsf{rep}$. To do this test in constant time we manage a counter which, at first, **count the number of transitions from** $E.\mathsf{rep}$ **to** $\mathscr{U}(b) = \mathscr{U}([b]_{\mathscr{P}})$. By scanning the transitions whose destination belongs to $(\mathscr{U} \setminus \mathscr{R})(b)$ this counter is updated to count the transitions from $E.\mathsf{rep}$ to $\mathscr{R}(b) = \mathscr{R}([b]_{\mathscr{P}})$. Therefore we get the equivalences: there is no transition from E to $\mathscr{R}(b)$ iff there is no transition from $E.\mathsf{rep}$ to $\mathscr{R}(b)$ iff this counter is null. Remark that the total bit size of all the counters is in $O(|P_{\text{sim}}|^2.\log(|Q|))$ since there is at most $|P_{\text{sim}}|$ blocks like E, $|P_{\text{sim}}|$ blocks like $[b]_{\mathscr{P}}$ and $|Q|$ transitions from a state like $E.\mathsf{rep}$. The difference is not so significative in practice but we will reduce this size to $O(|P_{\text{sim}}|^2.\log(|P_{\text{sim}}|))$, at a cost of $O(|P_{\text{sim}}|.|\rightarrow|)$ elementary steps, which is hopefully within our time budget. Removing from $\mathscr{R}(c)$ the blocks of $\rightarrow^{-1} \circ (\mathscr{U} \setminus \mathscr{R})(b)$, like $[d]_{\mathscr{P}}$, which do not have an outgoing transition to $\mathscr{R}(b)$ is called the **refine step**. After this refine step, $\mathscr{R}(c)$ has been reduced to $\mathscr{V}(c)$. Doing these split and refine steps for all transitions $c \rightarrow b$ results in the relation $\mathscr{V}$ that we will prove to be a $\mathscr{R}$-stable preorder.

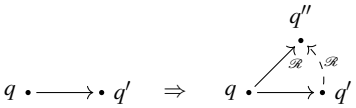

Fig. 2. Illustration of the left property of Lemma 3.

In summary, from an initial preorder we will build a strictly decreasing series of preorders $(\mathscr{R}_i)_{i\geq 0}$ such that $\mathscr{R}_{i+1}$ is $\mathscr{R}_i$-stable and contains, by construction, all simulations included in $\mathscr{R}_i$. Since all the relations are finite, this series has a limit, reached in a finite number of steps. Let us call $\mathscr{R}_{\text{sim}}$ this limit. We have: $\mathscr{R}_{\text{sim}}$ is $\mathscr{R}_{\text{sim}}$-stable. Therefore, with (1) and (3), $\mathscr{R}_{\text{sim}}$ is a simulation and by construction contains all simulations included in the initial preorder: this is the coarsest one.

Remark. The counters which are used in the previous paragraphs play a similar role as the counters used in [15]. Without them, the time complexity of the algorithm of the present paper would have been multiplied by a $|P_{\text{sim}}|$ factor and would have been this of GPP: $O(|P_{\text{sim}}|^2.|\rightarrow|)$.

IV. Underlying Theory

In this section we give the necessary theory to define what should be the ideal split step and we justify the correctness of our refine step which allows to treat blocks as if they were single states. We begin by introducing the notion of maximal transition. This is the equivalent concept for transitions from that of little brothers, introduced in [6], for states. The main difference is that little brothers have been defined relatively to the final coarsest simulation in a Kripke structure. Here we define maximal transitions relatively to a current preorder $\mathscr{R}$.

Definition 2. *Let $\mathscr{R}$ be a preorder. The transition $q \rightarrow q'$ is said* maximal for $\mathscr{R}$*, or* $\mathscr{R}$-maximal*, which is noted $q \rightarrow_{\mathscr{R}} q'$, when:*

$$\forall q'' \in Q \,.\, (q \rightarrow q'' \wedge q' \,\mathscr{R}\, q'') \Rightarrow q'' \in [q']_{\mathscr{R}}$$

The set of $\mathscr{R}$-maximal transitions and the induced relation are both noted $\rightarrow_{\mathscr{R}}$.

Lemma 3 (Fig. 2)**.** *For a preorder $\mathscr{R}$, the two following properties are verified:*

$$\rightarrow^{-1} \subseteq \rightarrow_{\mathscr{R}}^{-1} \circ \mathscr{R} \text{ and } \rightarrow^{-1} \circ \mathscr{R} = \rightarrow_{\mathscr{R}}^{-1} \circ \mathscr{R}$$

In the last section, we introduced the notions of stability and of block-stability. Let us define them formaly.

Definition 4. *Let $\mathscr{R}$ a preorder.*

- *$\mathscr{R}$ is said* $\mathscr{U}$-stable*, with $\mathscr{U}$ a coarser preorder than $\mathscr{R}$, if:*

$$\mathscr{R} \circ \rightarrow^{-1} \subseteq \rightarrow^{-1} \circ \mathscr{U} \tag{4}$$

- *An equivalence relation $\mathscr{P}$ included in $\mathscr{R}$, is said* $\mathscr{R}$-block-stable *if:*

$$\forall b,d,d' \in Q.\, d\,\mathscr{P}\,d' \Rightarrow (d \in {\rightarrow}^{-1} \circ \mathscr{R}(b) \Leftrightarrow d' \in {\rightarrow}^{-1} \circ \mathscr{R}(b)) \quad (5)$$

Remark. Say that $\mathscr{P}$ is included in $\mathscr{R}$ means that each block of $\mathscr{P}$ is included in a block of $\mathscr{R}$.

As seen in the following lemma we have a nice equivalence: an equivalence relation $\mathscr{P}$ is $\mathscr{R}$-block-stable iff it is $\mathscr{R}$-stable.

Lemma 5. *Let $\mathscr{P}$ be an equivalence relation included in a preorder $\mathscr{R}$. Then* (5) *is equivalent with:*

$$\mathscr{P} \circ {\rightarrow}^{-1} \subseteq {\rightarrow}^{-1} \circ \mathscr{R} \quad (6)$$

With (4) and (6) the reader should now be convinced by the interest of (1) to define a simulation.

Following the keys ideas given in Section III there is an interest, for the time complexity, of having a coarse $\mathscr{R}$-block-stable equivalence relation $\mathscr{P}$. Hopefully there is a coarsest one.

Proposition 6. *Given a preorder $\mathscr{R}$, there is a coarsest $\mathscr{R}$-stable equivalence relation.*

Proof. With Lemma 5 and by an easy induction based on the two following properties:

- the identity relation, $\mathscr{I} = \{(q,q) \,|\, q \in Q\}$, is a $\mathscr{R}$-stable equivalence relation.
- the reflexive and transitive closure $(\mathscr{P}_1 \cup \mathscr{P}_2)^*$ of the union of two $\mathscr{R}$-stable equivalence relations, $\mathscr{P}_1$ and $\mathscr{P}_2$, is also a $\mathscr{R}$-stable equivalence relation, coarser than them.

□

We are now ready to introduce the main result of this section. It is a formalization, and a justification, of the refine step given in Section III. In the following theorem, the link with the decreasing sequence of relations $(\mathscr{R}_i)_{i\geq 0}$ mentioned at the end of Section III is: if $\mathscr{R}_i$ is the current value of $\mathscr{R}$ then $\mathscr{R}_{i-1}$ is $\mathscr{U}$ and $\mathscr{R}_{i+1}$ will be $\mathscr{V}$. The reader can also ease its comprehension of the theorem by considering Fig. 1.

Theorem 7. *Let $\mathscr{U}$ be a preorder, $\mathscr{R}$ be a $\mathscr{U}$-stable preorder and $\mathscr{P}$ be the coarsest $\mathscr{R}$-stable equivalence relation. Let $NotRel = \mathscr{U} \setminus \mathscr{R}$ and $\mathscr{V} = \mathscr{R} \setminus NotRel'$ with*

$$NotRel' = \bigcup_{\substack{b,c,d \in Q,\, c \rightarrow b,\, c\,\mathscr{R}\,d,\\ d \in {\rightarrow}^{-1} \circ NotRel(b),\\ d \notin {\rightarrow}^{-1} \circ \mathscr{R}(b)}} [c]_{\mathscr{P}} \times [d]_{\mathscr{P}}$$

Then:

1) *$NotRel' = X$ with*

$$X = \bigcup_{\substack{b,c,d \in Q,\, c \rightarrow_{\mathscr{R}} b,\, c\,\mathscr{R}\,d,\\ d \in {\rightarrow_{\mathscr{R}}}^{-1} \circ NotRel(b),\\ d \notin {\rightarrow_{\mathscr{R}}}^{-1} \circ \mathscr{R}(b)}} \{(c,d)\}$$

2) *Any simulation $\mathscr{S}$ included in $\mathscr{R}$ is also included in $\mathscr{V}$.*
3) $\mathscr{V} \circ {\rightarrow}^{-1} \subseteq {\rightarrow}^{-1} \circ \mathscr{R}$
4) *$\mathscr{V}$ is a preorder.*
5) *$\mathscr{V}$ is $\mathscr{R}$-stable.*
6) *Blocks of $\mathscr{V}$ are blocks of $\mathscr{P}$ (i.e. $P_{\mathscr{V}} = P_{\mathscr{P}}$).*

Remark. 1) means that blocks of $\mathscr{P}$ are sufficiently small to do the refinement step efficiently, as if they were states. 6) means that these blocks cannot be bigger. 5) means that we are ready for next split and refinement steps.

Proof of Theorem 7.

1) Since (c,d) belongs to $[c]_{\mathscr{P}} \times [d]_{\mathscr{P}}$, $\rightarrow_{\mathscr{R}} \subseteq \rightarrow$ and ${\rightarrow}^{-1} \circ \mathscr{R} = {\rightarrow_{\mathscr{R}}}^{-1} \circ \mathscr{R}$, from Lemma 3, we get $X \subseteq NotRel'$. For the converse, let $(c',d') \in NotRel'$. By definition, there are $b,c,d \in Q$ such that $c \rightarrow b$, $c\,\mathscr{R}\,d$, $d \in {\rightarrow}^{-1} \circ NotRel(b)$, $d \notin {\rightarrow}^{-1} \circ \mathscr{R}(b)$, $c' \in [c]_{\mathscr{P}}$ and $d' \in [d]_{\mathscr{P}}$. From $d \notin {\rightarrow}^{-1} \circ \mathscr{R}(b)$ and Lemma 3 we have $d \notin {\rightarrow_{\mathscr{R}}}^{-1} \circ \mathscr{R}(b)$. From $c \rightarrow b$, Lemma 3, Lemma 5, and the hypothesis that $\mathscr{P}$ is $\mathscr{R}$-stable, we have $c' \rightarrow_{\mathscr{R}}^{-1} \circ \mathscr{R}(b)$. Therefore, there is a state b' such that $c' \rightarrow_{\mathscr{R}} b'$ and $b\,\mathscr{R}\,b'$. Les us suppose $d' \in {\rightarrow_{\mathscr{R}}}^{-1} \circ \mathscr{R}(b')$. Since $b\,\mathscr{R}\,b'$, we would have had $d \in {\rightarrow_{\mathscr{R}}}^{-1} \circ \mathscr{R}(b)$. Thus $d' \notin {\rightarrow_{\mathscr{R}}}^{-1} \circ \mathscr{R}(b')$. We have $c'\,\mathscr{P}\,c$, $c\,\mathscr{R}\,d$, $d\,\mathscr{P}\,d'$, and thus $c'\,\mathscr{R}\,d'$ since $\mathscr{R}$ is a preorder and $\mathscr{P} \subseteq \mathscr{R}$. With $c' \rightarrow_{\mathscr{R}} b'$ and the hypothesis that $\mathscr{R}$ is $\mathscr{U}$-stable, we get $d' \in {\rightarrow}^{-1} \circ \mathscr{U}(b')$ and thus, with Lemma 3, $d' \in {\rightarrow_{\mathscr{R}}}^{-1} \circ \mathscr{R} \circ \mathscr{U}(b')$ and thus $d' \in {\rightarrow_{\mathscr{R}}}^{-1} \circ \mathscr{U}(b')$ since $\mathscr{U}$ is a preorder and $\mathscr{R} \subseteq \mathscr{U}$. As seen above, $d' \notin {\rightarrow_{\mathscr{R}}}^{-1} \circ \mathscr{R}(b')$. So we have $d' \in {\rightarrow_{\mathscr{R}}}^{-1} \circ (\mathscr{U} \setminus \mathscr{R})(b')$. In summary: $c' \rightarrow_{\mathscr{R}} b'$, $c'\,\mathscr{R}\,d'$, $d' \in {\rightarrow_{\mathscr{R}}}^{-1} \circ NotRel(b')$ and $d' \notin {\rightarrow_{\mathscr{R}}}^{-1} \circ \mathscr{R}(b')$. All of this implies that $(c',d') \in X$. So we have $NotRel' \subseteq X$ and thus $NotRel' = X$.
2) By contradiction. Let $(c,d) \in \mathscr{S}$ such that $(c,d) \notin \mathscr{V}$. This means that $(c,d) \in NotRel'$. From 1) and the hypothesis $\mathscr{S} \subseteq \mathscr{R}$ there is $b \in Q$ such that $c \rightarrow_{\mathscr{R}} b$, $d \notin {\rightarrow_{\mathscr{R}}}^{-1} \circ \mathscr{R}(b)$ and thus $d \notin {\rightarrow}^{-1} \circ \mathscr{R}(b)$, from Lemma 3. From $c \rightarrow_{\mathscr{R}} b$, thus $c \rightarrow b$, and the assumption that $\mathscr{S}$ is a simulation there is $d' \in Q$ with $d \rightarrow d'$ and $b\,\mathscr{S}\,d'$ thus $b\,\mathscr{R}\,d'$. This contradicts $d \notin {\rightarrow}^{-1} \circ \mathscr{R}(b)$. Therefore $\mathscr{S} \subseteq \mathscr{V}$.
3) If this is not the case, there are $b,c,d \in Q$ such that $c\,\mathscr{V}\,d$, $c \rightarrow b$ and $d \notin {\rightarrow}^{-1} \circ \mathscr{R}(b)$. Since $\mathscr{V} \subseteq \mathscr{R}$ and $\mathscr{R}$ is a $\mathscr{U}$-stable relation there is $d' \in Q$ such that $d \rightarrow d'$ and $b\,\mathscr{U}\,d'$. The case $b\,\mathscr{R}\,d'$ would contradict $d \notin {\rightarrow}^{-1} \circ \mathscr{R}(b)$. Therefore $d \in {\rightarrow}^{-1} \circ NotRel(b)$ and all the conditions are met for (c,d) belonging in $NotRel'$ which contradicts $c\,\mathscr{V}\,d$.
4) Let us show that $\mathscr{V}$ is both reflexive and transitive. If it is not reflexive, since $\mathscr{R}$ is reflexive, from 1) there is (c,d) in X and a state b such that $c \rightarrow_{\mathscr{R}} b$ and $d \notin {\rightarrow_{\mathscr{R}}}^{-1} \circ \mathscr{R}(b)$ and $c = d$. But this is impossible since $\mathscr{R}$ is reflexive. Hence, $\mathscr{V}$ is reflexive. We also prove by contradiction that $\mathscr{V}$ is transitive. If it is not the case, there are $c,e,d \in Q$ such that $c\,\mathscr{V}\,e$, $e\,\mathscr{V}\,d$ but $\neg\, c\,\mathscr{V}\,d$. Since $\mathscr{V} \subseteq \mathscr{R}$ and $\mathscr{R}$ is transitive then $c\,\mathscr{R}\,d$. With $\neg\, c\,\mathscr{V}\,d$ and 1), there is b such that $c \rightarrow_{\mathscr{R}} b$ and $d \notin {\rightarrow_{\mathscr{R}}}^{-1} \circ \mathscr{R}(b)$. But from 3), there is $b' \in Q$ such that $b\,\mathscr{R}\,b'$ and $e \rightarrow b'$. With $e\,\mathscr{V}\,d$ and the same reason, there is b'' such that $b'\,\mathscr{R}\,b''$ and $d \rightarrow b''$. By transitivity of $\mathscr{R}$ we get $b\,\mathscr{R}\,b''$ and

thus $d \in {\rightarrow}^{-1} \circ \mathscr{R}(b)$. With Lemma 3 this contradicts $d \notin {\rightarrow}^{-1}_{\mathscr{R}} \circ \mathscr{R}(b)$. Hence, $\mathscr{V}$ is transitive.

5) This is a direct consequence of the two preceding items and the fact that by construction $\mathscr{V} \subseteq \mathscr{R}$.
6) By hypothesis, $\mathscr{P} \subseteq \mathscr{R}$. This means that blocks of $\mathscr{R}$ are made of blocks of $\mathscr{P}$. By definition, $\mathscr{V}$ is obtained by deleting from $\mathscr{R}$ relations between blocks of $\mathscr{P}$. This implies that blocks of $\mathscr{V}$ are made of blocks of $\mathscr{P}$. To proove that a block of $\mathscr{V}$ is made of a single block of $\mathscr{P}$, let us assume, by contradiction, that there are two different blocks, B_1 and B_2, of $\mathscr{P}$ in a block of $\mathscr{V}$. We show that $\mathscr{P}$ is not the coarsest $\mathscr{R}$-block-stable equivalence relation. Let $\mathscr{P}' = \mathscr{P} \cup B_1 \times B_2 \cup B_2 \times B_1$. Then $\mathscr{P}'$ is an equivalence relation strictly coarser than $\mathscr{P}'$. Furthermore, since B_1 and B_2 are blocks of $\mathscr{V}$, we get $\mathscr{P}' \subseteq \mathscr{V}$. With 3) we get that $\mathscr{P}'$ is $\mathscr{R}$-stable and thus $\mathscr{R}$-block-stable with Lemma 5. This contradicts the hypothesis that $\mathscr{P}$ was the coarsest one. Therefore, blocks of $\mathscr{V}$ are blocks of $\mathscr{P}$.

□

In what precedes, we have assumed that for the preorder $\mathscr{R}$ inside which we want to compute the coarsest simulation, there is another preorder $\mathscr{U}$ such that condition (4) holds. The fifth item of Theorem 7 says that if this true at a given iteration (made of a split step and a refinement step) of the algorithm then this is true at the next iteration. For the end of this section we show that we can safely modify the initial preorder such that this is also initially true. This is indeed a simple consequence of the fact that a state with an outgoing transition cannot be simulated by a state with no outgoing transition.

Definition 8. *Let $\mathscr{R}_{\text{init}}$ be a preorder. We define* $\text{InitRefine}(\mathscr{R}_{\text{init}})$ *such that:*

$$\text{InitRefine}(\mathscr{R}_{\text{init}}) \triangleq \mathscr{R}_{\text{init}} \cap \{(c,d) \in Q \times Q \,\big|\, \exists c' \in Q \,.\, c \rightarrow c' \;\Rightarrow\; \exists d' \in Q \,.\, d \rightarrow d'\}$$

Proposition 9. *Let $\mathscr{R} = \text{InitRefine}(\mathscr{R}_{\text{init}})$ with $\mathscr{R}_{\text{init}}$ a preorder. Then:*

1) *$\mathscr{R}$ is $(Q \times Q)$-stable,*
2) *a simulation $\mathscr{S}$ included in $\mathscr{R}_{\text{init}}$ is also included in $\mathscr{R}$.*

The total relation $Q \times Q$ will thus play the role of the initial $\mathscr{U}$ in the algorithm.

Remark. In [15, p. 979] there is also a similar preprocessing of the initial partition where states with no output transition are put aside.

V. The Algorithm

The approach of the previous section can be applied to several algorithms, with different balances between time complexity and space complexity. It can also be extended to labelled transition systems. In this section the emphasis is on the, theoretically, most efficient in memory of the fastest simulation algorithms of the moment.

A. Counters and Splitter Transitions

Let us remember that a partition P and its associated equivalence relation $\mathscr{P}_P$ (or a equivalence relation $\mathscr{P}$ and its associated partition $P_{\mathscr{P}}$) denote essentially the same thing. The difference is that for a partition we focus on the set of blocks whereas for an equivalence relation we focus on the relation which relates the elements of a same block. For a first reading, the reader may consider that a partition is an equivalence relation, and vice versa.

From a preorder $\mathscr{R}$ that satisfies (4) we will need to split its blocks in order to find its coarsest $\mathscr{R}$-stable equivalence relation. Then, Theorem 7 will be used for a refine step. For all this, we first need the traditional `Split` function.

Definition 10. *Given a partition P and a set of states $Marked$, the function* $\texttt{Split}(P, Marked)$ *returns a partition similar to P, but with the difference that each block E of P such that $E \cap Marked \neq \emptyset$ and $E \not\subseteq Marked$ is replaced by two blocks: $E_1 = E \cap Marked$ and $E_2 = E \setminus E_1$.*

To efficiently perform the split and refine steps we need a set of counters which associates to each representative state of a block, of an equivalence relation, the number of blocks, of that same equivalence relation, it reaches in $\mathscr{R}(B)$, for B a block of $\mathscr{R}$.

Definition 11. *Let $\mathscr{P}$ be an equivalence relation included in a preorder $\mathscr{R}$. We assume that for each block E of $\mathscr{P}$, a representative state $E.\mathsf{rep}$ has been chosen. Let E be a block of $\mathscr{P}$, B be a block of $\mathscr{R}$ and $B' \subseteq B$. We define:*

$$\texttt{RelCount}_{(\mathscr{P},\mathscr{R})}(E,B') \triangleq \left|\{E' \in P_{\mathscr{P}} \,\big|\, E.\mathsf{rep} \rightarrow E' \wedge B' \,\mathscr{R}\, E'\}\right| \quad (7)$$

Proposition 12. *Let $\mathscr{P}$ be an equivalence relation included in a preorder $\mathscr{R}$, E be a block of $\mathscr{P}$, B be a block of $\mathscr{R}$ and B' be a non empty subset of B. Then:*

$$\texttt{RelCount}_{(\mathscr{P},\mathscr{R})}(E,B) = \texttt{RelCount}_{(\mathscr{P},\mathscr{R})}(E,B')$$

Proof. Thanks to the transitivity of $\mathscr{R}$. □

Following Section III, the purpose of these counters is to check in constant time whether a block E of an equivalence relation $\mathscr{P}$ is included in ${\rightarrow}^{-1} \circ \mathscr{R}(b)$ for a given state b. But this is correct only if $\mathscr{P}$ is already $\mathscr{R}$-block-stable. If this is not the case, its underlying partition should be split accordingly. We thus introduce the first condition which necessitates a split of the current equivalence relation $\mathscr{P}$ to approach the coarsest $\mathscr{R}$-block-stable equivalence relation. For this, we take advantage of the existence of $\texttt{RelCount}_{(\mathscr{P},\mathscr{R})}$.

Definition 13. *Let $\mathscr{P}$ be an equivalence relation included in a preorder $\mathscr{R}$, E be a block of $\mathscr{P}$ and B be a block of $\mathscr{R}$ such that $E \rightarrow B$ and $\texttt{RelCount}_{(\mathscr{P},\mathscr{R})}(E,B) = 0$. The transition $E \rightarrow B$ is called a* $(\mathscr{P},\mathscr{R})$-splitter transition of type 1.

The intuition is as follows. With a block E of $\mathscr{P}$ and a block B of $\mathscr{R}$, if $\mathscr{P}$ was $\mathscr{R}$-block-stable, with $E \rightarrow B$ and Lemma 5 we would have had $E \subseteq {\rightarrow}^{-1} \circ \mathscr{R}(B)$. But

$\texttt{RelCount}_{(\mathscr{P},\mathscr{R})}(E,B) = 0$ denies this. So we have to split $P_{\mathscr{P}}$.

Lemma 14. *Let $E \to B$ be a $(\mathscr{P},\mathscr{R})$-splitter transition of type 1. Let $P' = \texttt{Split}(P_{\mathscr{P}}, \to^{-1} \circ \mathscr{R}(B))$. Then $\mathscr{P}_{P'}$ is strictly included in $\mathscr{P}$ and contains all $\mathscr{R}$-block-stable equivalence relations included in $\mathscr{P}$.*

Saying that $\mathscr{P}_{P'}$ is strictly included in $\mathscr{P}$ means that at least one block of $\mathscr{P}$, here E, has been split to obtain P'.

Lemma 15. *Let $\mathscr{P}$ be an equivalence relation included in a preorder $\mathscr{R}$ such that there is no $(\mathscr{P},\mathscr{R})$-splitter transition of type 1. Let E be a block of $\mathscr{P}$ and B be a block of $\mathscr{R}$ such that $E \to B$. Then, $E.\mathsf{rep} \in \to^{-1}_{\mathscr{R}} \circ \mathscr{R}(B)$.*

Now, for E to be really a representative, we need the following implication: $E.\mathsf{rep} \to B \Rightarrow E \subseteq \to^{-1} \circ \mathscr{R}(B)$. But to check this property effectively, taking advantage of the counters, we need a stronger property equivalent with the one, (5), defining block-stability.

Lemma 16. *Let $\mathscr{P}$ be an equivalence relation included in a preorder $\mathscr{R}$. Then* (5) *is equivalent with:*

$$\mathscr{P} \circ \to^{-1}_{\mathscr{R}} \subseteq \to^{-1} \circ [\cdot]_{\mathscr{R}} \tag{8}$$

Definition 17. *Let $\mathscr{P}$ be an equivalence relation included in a preorder $\mathscr{R}$. Let E be a block of $\mathscr{P}$ and B be a block of $\mathscr{R}$ such that $E.\mathsf{rep} \to B$, $\texttt{RelCount}_{(\mathscr{P},\mathscr{R})}(E,B) = |\{[b]_{\mathscr{P}} \subseteq B \,|\, E.\mathsf{rep} \to b\}|$ and $E \not\subseteq \to^{-1}(B)$. The transition $E \to B$ is called a* $(\mathscr{P},\mathscr{R})$-splitter transition of type 2.

Remark. The conditions in Definition 17 are inspired from those used in [14] for its split step.

The intuition is as follows. If $E.\mathsf{rep} \to B$ and the condition on the counter is true (all transitions from $E.\mathsf{rep}$ that reach states greater, relatively to $\mathscr{R}$, than B actually have their destination states in B), this means that the transition $E.\mathsf{rep} \to B$ is maximal. With Lemma 16, if $\mathscr{P}$ is $\mathscr{R}$-block-stable this should imply $E \subseteq \to^{-1}(B)$. Since this is not the case, $\mathscr{P}$ must be split.

Lemma 18. *Let $\mathscr{P}$ be an equivalence relation included in a preorder $\mathscr{R}$ such that there is no $(\mathscr{P},\mathscr{R})$-splitter transition of type 1 and let $E \to B$ be a splitter transition of type 2. Let $P' = \texttt{Split}(P_{\mathscr{P}}, E \cap \to^{-1}(B))$. Then $\mathscr{P}_{P'}$ is strictly included in $\mathscr{P}$ and contains all $\mathscr{R}$-block-stable equivalence relations included in $\mathscr{P}$.*

Theorem 19. *Let $\mathscr{P}$ be an equivalence relation included in a preorder $\mathscr{R}$ such that there is no $(\mathscr{P},\mathscr{R})$-splitter transition of type 1 or $(\mathscr{P},\mathscr{R})$-splitter transition of type 2. Then $\mathscr{P}$ is $\mathscr{R}$-block-stable.*

Therefore, in the algorithm, before a refine step on $\mathscr{R}$ using Theorem 7, we will start from the partition $P_{\mathscr{R}}$ and split it in conformity with Lemma 14 and Lemma 18. By doing so, we will obtain the coarsest $\mathscr{R}$-block-stable equivalence relation.

The next proposition shows where to search splitter transitions: those who ends in blocks B of $\mathscr{R}$ such that $\mathit{NotRel}(B)$ is not empty.

Proposition 20. *Let $\mathscr{U}$ be a preorder, $\mathscr{R}$ be a $\mathscr{U}$-stable preorder, $\mathscr{P}$ be an equivalence relation included in $\mathscr{R}$ and let $\mathit{NotRel} = \mathscr{U} \setminus \mathscr{R}$. Then*

1) *If $E \to B$ is a $(\mathscr{P},\mathscr{R})$-splitter transition of type 1 then $\mathit{NotRel}(B) \neq \emptyset$.*
2) *Under the absence of $(\mathscr{P},\mathscr{R})$-splitter transition of type 1, if $E \to B$ is a $(\mathscr{P},\mathscr{R})$-splitter transition of type 2 then $\mathit{NotRel}(B) \neq \emptyset$.*

We have now everything to propose an efficient algorithm.

B. Data Structures and Space Complexity

Remark. Since the final partition P_{sim} is obtained after several splits of the initial partition P_{init} we have $|P| \leq |P_{\text{sim}}|$ with P the current partition at a given step of the algorithm.

The current relation $\mathscr{R}$ is represented in the algorithm by a partition-relation pair (P, Rel). The data structure used to represent a partition is traditionally a (possibly) doubly linked list of blocks, themselves represented by a doubly linked list of states. But in practice, each node of a list contains a reference, to the next node of the list. The size (typically 64 bits nowadays) of these references is static and does not depend on the size of the list. We therefore prefer the use of arrays because we can control the size of the slots and manipulate arrays is faster than manipulate lists. The idea, see [17] and [13] for more details, is to identify a state with its index in Q and to distribute these indexes in an array, let us name it T, such that states belonging to the same block are in consecutive slots in T. A block could thus be represented by the indexes of its first and last elements in T, the two indexes defining a contiguous subarray. If a block is split, the two subblocks form two contiguous subarrays of that subarray. By playing with these arrays (other arrays are needed, like the one giving the position in T of a state) we obtain a representation of a partition which allows splitting (some elements of a block are removed from their original block and form a new block) and scanning of a block in linear time.

However, as seen in the previous sections, we need two generations of blocks at the same time: the first one corresponds to blocks of $\mathscr{R}$ and the second one corresponds to blocks of the next generation, $\mathscr{V}$, of this preorder. Hence, we need an intermediate between blocks and their corresponding states: *nodes*. A node corresponds to a block or to an ancestor of a block in the family tree of the different generations of blocks issued from the split steps of the algorithm. To simplify the writing, we associate in the present paper, a node to the set of its corresponding states. As an example, consider a block B which consists of the following three states $\{q_1, q_2, q_3\}$. In reality, we will associate B to a node $N = \{q_1, q_2, q_3\}$. In this way, if B is split in B_1 and B_2, corresponding respectively to $\{q_1, q_2\}$ and $\{q_3\}$, we create two new nodes $N_1 = \{q_1, q_2\}$ and $N_2 = \{q_3\}$ and we associate B_1 to N_1 and B_2 to N_2. By doing so, N remains bounds to the set $\{q_1, q_2, q_3\}$. To represent a

node we just need to keep in memory the index of its first element and the index of its last element in the array T (see the previous paragraph). When a block which corresponds to a node is split, the corresponding states change their places in T but keep in the same subarray. Let us note that when a block is split, it is necessarily in two parts. Therefore, the number of nodes is at most twice the number of blocks and the bit space needed to represent the partition and the nodes is in $O(|Q|.\log(|Q|))$ since there is less blocks than states.

To be more precise about the relations between states, blocks and nodes: at any time of the algorithm, the index of a state q is associated to the index of its block q.block, the index of a block E is associated to the index of its node E.node and the index of a node is associated to the states it contains (via two indexes of the array T). A node which is not linked by a block is an ancestor of, at least two, blocks. By the data structure chosen to represent the partition it is easy to see that given a node N we can scan in linear time the states it contains (this correspond to the scan of a contiguous subarray) and the blocks it contains (by a similar process). The function `ChooseBlock`(N) which arbitrarily choose one block whose set of elements is included in those of N is executed in constant time (we choose e.block, with e the first element of N). Similarly, the function `ChooseState`(E) which returns a state of a block E, used to defined E.rep a representative of E, is also executed in constant time (we choose the first element in E.node).

The relation *Rel* is distributed on the blocks. To each block C we associate an array of booleans, C.Rel, such that $(C,D) \in Rel$, what we note $D \in C$.Rel, iff the boolean at the index of the block D in C.Rel is true. These arrays are resizable arrays whose capacities are doubled as needed. Therefore the classical operations on these arrays, like get and set, take constant amortized time. We use this type of array wherever necessary. The bit size needed to represent *Rel* is therefore in $O(|P_{\text{sim}}|^2)$.

The relation $\mathscr{U}$ that appears in the previous sections is not directly represented. We use instead the equality $\mathscr{U} = \mathscr{R} \cup NotRel$ and represent *NotRel*. Since $\mathscr{U}$ is a coarser preorder than $\mathscr{R}$, for a given node B which represents a block of $\mathscr{R}$, the set $NotRel(B)$ is represented in the algorithm by B.NotRel a set of nodes (encoded by a resizable array of the indexes of the corresponding nodes) which represent blocks of $\mathscr{R}$. As explained earlier, we have to use nodes instead of blocks because nodes never change whereas blocks can be split afterwards. The bit space representation of *NotRel* is thus in $O(|P_{\text{sim}}|^2.\log(|P_{\text{sim}}|))$. Remember, the number of nodes is linear in the maximal number of blocks: $|P_{\text{sim}}|$.

In Section V-A we introduced a counter, $\texttt{RelCount}_{(\mathscr{P},\mathscr{R})}(E,B)$, for each pair made of a block E of the current equivalence relation $\mathscr{P}$ represented in the algorithm by the current partition P, and a block B of the current relation $\mathscr{R}$ represented in the algorithm by (P, Rel). As seen in Proposition 12, for any subblock $B' \in P_{\mathscr{P}}$ of B we have $\texttt{RelCount}_{(\mathscr{P},\mathscr{R})}(E,B) = \texttt{RelCount}_{(\mathscr{P},\mathscr{R})}(E,B')$. Therefore, we can limit these counters to any pair of blocks of the current partition P in the algorithm. Such a counter counts a number of blocks. This means that the total bit size of these counters is in $O(|P_{\text{sim}}|^2.\log(|P_{\text{sim}}|))$. In practice, we associate to each block B' a resizable array, B'.`RelCount`, of $|P|$ elements such that $B'.\text{RelCount}(E) = \texttt{RelCount}_{(\mathscr{P},\mathscr{R})}(E,B')$.

At several places in the algorithm we use a data structure to manage a set of indexed elements. This is the case for *Touched*, *Touched*$'$, *Marked*, *RefinerNodes*, *RefinerNodes*$'$, *PreB*$'$, *Remove*, *PreE* and *PreE*$'$. Such a set is implemented by a resizable boolean array, to know in constant time whether a given element belongs to the set, and another resizable array to store the indexes of the elements which belongs to the set. This last array serves to scan in linear time the elements of the set or to emptied the set in linear time of the number of the elements in the set. So the operations, add an element in the set and test whether an element belongs to the set are done in constant time or amortized constant time. Also, scanning the elements of the set and emptying the set are executed in linear time of the size of the set. We use a finite number of these sets for states, blocks or nodes. The overall bit space used for them is therefore in $O(|Q|.\log(|Q|))$. The other variables used in the algorithm, some booleans and a counter, E.count, associated to each block E in Function Split2 are manipulated by constant time operations and need a bit space in $O(|Q|.\log(|Q|))$ since $|P|$ and the number of nodes are both in $O(|Q|)$. From all of this, we derive the following theorem.

Theorem 21. *The overall bit space used by the presented simulation algorithm is in* $O(|P_{\text{sim}}|^2.\log(|P_{\text{sim}}|) + |Q|.\log(|Q|))$.

Remark. We assume one can iterate through the transition relation $\to$ in linear time of its size. From each state q we also assume one can iterate through the set $\to^{-1}(q)$ in linear time of its size. It is a tradition in most articles dealing with simulation to not count the space used to represent the transition relation since it is considered as an input data. If it was to be counted it would cost $O(|\to|.\log(|Q|))$ bits.

C. Procedures and Time Complexity

In this section we analyze the different functions of the algorithm and give their overall time complexities. The reader should remember that, in the algorithm, a block is just an index and the set of states corresponding to a block $E \in P$ is E.node.

1) Function Sim: This is the main function of the algorithm. It takes as input a transition system $(Q,\to)$ and an initial preorder, $\mathscr{R}_{\text{init}}$, represented by the partition-relation pair $(P_{\text{init}}, R_{\text{init}})$. Let us define the two following relations:

$$\mathscr{R} \triangleq \bigcup_{\{(E,E') \in P^2 \,|\, E' \in E.\text{Rel}\}} E.\text{node} \times E'.\text{node} \tag{9}$$

$$NotRel \triangleq \bigcup_{B \in RefinerNodes} B \times (\cup B.\text{NotRel}) \tag{10}$$

Let $\mathscr{R}_0 = Q \times Q$ and $\mathscr{R}_i$ (resp. $NotRel_i$) be the value of $\mathscr{R}$ (resp. *NotRel*) at the i^{th} iteration of the while loop at line 3 of Function Sim. We will show (in the analysis of Function Init for the base case and procedures SimUpdateData and Refine

Function Sim$(Q,\rightarrow,P_{\text{init}},R_{\text{init}})$

Data: $RefinerNodes$: the set of nodes B corresponding to blocks of $\mathscr{R}$ such that $B.\mathsf{NotRel} \neq \emptyset$

1 $RefinerNodes := \emptyset$;
2 $P :=$ `Init` $(Q,\rightarrow,P_{\text{init}},R_{\text{init}},RefinerNodes)$;
3 **while** $RefinerNodes \neq \emptyset$ **do**
4 `SimUpdateData` $(P,RefinerNodes)$;
5 `Split1` $(P,RefinerNodes)$;
6 `Split2` $(P,RefinerNodes)$;
7 `Refine` $(RefinerNodes)$;
8 $P_{\text{sim}} := P$; $R_{\text{sim}} := \{(C,D) \in P \times P \,|\, D \in C.\mathsf{Rel}\}$;
9 **return** $(P_{\text{sim}},\ R_{\text{sim}})$

for the inductive step) that, at this line 3, we maintain the five following properties at the i^{th} iteration of the while loop:

$$\mathscr{R}_i \text{ is } \mathscr{R}_{i-1}\text{-stable} \tag{11}$$

$$\text{A simulation included in } \mathscr{R}_{i-1} \text{ is included in } \mathscr{R}_i \tag{12}$$

$$NotRel_i = \mathscr{R}_{i-1} \setminus \mathscr{R}_i \text{ and thus } \mathscr{R}_{i-1} = \mathscr{R}_i \cup NotRel_i \tag{13}$$

$$RefinerNodes \text{ is the set of nodes } B \text{ corresponding to blocks of } \mathscr{R}_i \text{ such that } B.\mathsf{NotRel} \neq \emptyset \tag{14}$$

$$\forall E,B' \in P \,.\, B'.\mathsf{RelCount}(E) = |\{E' \in P \,|\, E.\mathsf{rep} \rightarrow E'.\mathsf{node} \wedge B'.\mathsf{node} \times E'.\mathsf{node} \subseteq \mathscr{R}_{i-1}\}| \tag{15}$$

From (11), and thus $\mathscr{R}_i \subseteq \mathscr{R}_{i-1}$, (13), (14) and the condition of the while loop we get that $(\mathscr{R}_i)_{i\geq 0}$ is a strictly decreasing sequence of relations. Since the underlying set of states is finite, this sequence reaches a limit in a finite number of iterations. Furthermore, if this limit is reached at the k^{th} iteration, then, from (14) and the condition of the while loop, we have $NotRel_{k+1} = \emptyset$ and from (13) and (11) we obtain that $\mathscr{R}_k = \mathscr{R}_{k+1}$ and $\mathscr{R}_{k+1}$ is $\mathscr{R}_k$-stable. Which means that $\mathscr{R}_k$ is a simulation. From (12) and the fact, to be shown in the analysis of function Init, that all simulation included in $\mathscr{R}_{\text{init}}$ is also included in $\mathscr{R}_1$, we deduce that $\mathscr{R}_k$ contains all simulation included in $\mathscr{R}_{\text{init}}$. Therefore, Function Sim returns a partition-relation pair that corresponds to $\mathscr{R}_{\text{sim}}$, the coarsest simulation included in $\mathscr{R}_{\text{init}}$.

The fact that $(\mathscr{R}_i)_{i\geq 0}$ is a strictly decreasing sequence of relations and (13) imply the following lemma that will be used as a key argument to analyze the time complexity of the algorithm.

Lemma 22. *Two states, and thus two blocks or two nodes, are related by NotRel in at most one iteration of the while loop in function Sim.*

Procedure SimUpdateData$(P,RefinerNodes)$

1 $PreE' := \emptyset$;
2 **foreach** $B \in RefinerNodes$ **do**
3 $B' =$ `ChooseBlock` (B) ;
`// At this stage, B' is the only block of B`
4 **foreach** $E' \in P \,|\, E'.\mathsf{node} \subseteq \cup B.\mathsf{NotRel}$ **do**
5 **foreach** $e \in \rightarrow^{-1}(E'.\mathsf{node})$ **do**
6 $E := e.\mathsf{block}$;
7 **if** $e = E.\mathsf{rep}$ **then**
8 $PreE' := PreE' \cup \{E\}$;
9 **foreach** $E \in PreE'$ **do** $B'.\mathsf{RelCount}(E)--$;
10 $PreE' := \emptyset$;

Procedure Split$(P,Marked)$

1 $Touched := \emptyset$;
2 **foreach** $r \in Marked$ **do** $Touched := Touched \cup \{r.\mathsf{block}\}$;
3 **foreach** $C \in Touched \,|\, C.\mathsf{node} \nsubseteq Marked$ **do**
4 $D :=$ `newBlock()` ; $P := P \cup \{D\}$;
5 $D.\mathsf{node} := C.\mathsf{node} \cap Marked$;
6 **foreach** $q \in D.\mathsf{node}$ **do** $q.\mathsf{block} := D$;
`// The subblock which is disjoint from Marked keeps the identity of C.`
7 $C.\mathsf{node} := C.\mathsf{node} \setminus D.\mathsf{node}$;
8 `SplitUpdateData` (P,C,D) ;
9 $Touched := \emptyset$;

2) Procedure SimUpdateData: Assuming (15) is true, the role of this procedure is to render the following formula true after line 4 of Function Sim during the i^{th} iteration of the while loop. In this way, the counters are made consistent with (7).

$$\forall E,B' \in P \,.\, B'.\mathsf{RelCount}(E) = |\{E' \in P \,|\, E.\mathsf{rep} \rightarrow E'.\mathsf{node} \wedge B'.\mathsf{node} \times E'.\mathsf{node} \subseteq \mathscr{R}_i\}| \tag{16}$$

From (10), (13), (14) and (15) we just have, for each node B in $RefinerNodes$, to scan the blocks in $\cup B.\mathsf{NotRel}$ in order to identify their predecessor blocks. The corresponding counters are then decreased. The lines which are the most executed are those in the loop starting at line 5. They are executed once for each pair of $(e \rightarrow e', B)$ with $e' \in \cup B.\mathsf{NotRel}$. But from Lemma 22 such a pair can be considered only once during the life time of the algorithm and thus the overall time complexity of this procedure is in $O(|P_{\text{sim}}|.|\rightarrow|)$.

3) Procedure Split: This procedure corresponds to Definition 10. The differences are that Procedure Split transforms the current partition (it is not a function) and each time a block is split, Procedure SplitUpdateData is called to update the data structures (mainly Rel and RelCount). Apart from the call

Procedure SplitUpdateData(P,C,D)

Data: C keeps the identity of the parent block ; D is the new block

1 D.count $:= 0$;

// Update of the Rel's

2 D.Rel $:=$ copy(C.Rel) ;

3 **foreach** $E \in P \,|\, C \in E$.Rel **do** E.Rel $:= E$.Rel $\cup \{D\}$;

// Update of the RelCount's

4 D.rep $:= C$.rep ;

5 D.RelCount $:=$ copy(C.RelCount) ;

6 **if** C.rep.block $= C$ **then**

7 $\quad X := D$;

8 **else**

9 $\quad$ **foreach** $B' \in P$ **do**

10 $\quad\quad B'$.RelCount(D) $:= B'$.RelCount(C) ;

11 $\quad X := C$;

// Update of the RelCount's from predecessors of both C and D

12 $Touched := \emptyset$; $Touched' := \emptyset$;

13 **foreach** $e \in \rightarrow^{-1}(C.\text{node}) \,|\, e = e$.block.rep **do**

14 $\quad Touched := Touched \cup \{e.\text{block}\}$;

15 **foreach** $e \in \rightarrow^{-1}(D.\text{node}) \,|\, e = e$.block.rep $\wedge$ e.block $\in Touched \wedge e$.block $\notin Touched'$ **do**

16 $\quad Touched' := Touched' \cup \{e.\text{block}\}$;

17 **foreach** $E \in Touched'$ **do**

18 $\quad$ **foreach** $B' \in P \,|\, C \in B'$.Rel **do**

19 $\quad\quad B'$.RelCount(E)++ ;

20 $Touched := \emptyset$; $Touched' := \emptyset$;

// Compute the RelCount's from X (C or D) which did not inherited the old C.rep.

21 X.rep $:=$ ChooseState(X) ;

22 **foreach** $B' \in P$ **do** B'.RelCount(X) $:= 0$;

23 **foreach** $q \rightarrow q' \,|\, q = X$.rep **do**

24 $\quad Touched := Touched \cup \{q'.\text{block}\}$;

25 **foreach** $E' \in Touched$ **do**

26 $\quad$ **foreach** $B' \in P \,|\, E' \in B'$.Rel **do**

27 $\quad\quad B'$.RelCount(X)++ ;

28 $Touched := \emptyset$;

29 C.node.NotRel $:= \emptyset$; C.node.NotRel$' := \emptyset$;

30 D.node.NotRel $:= \emptyset$; D.node.NotRel$' := \emptyset$;

of SplitUpdateData, which we discuss right after, it is known that, with a correct implementation found in most articles from the bibliography of the present paper, a call of Split is done in $O(|Marked|)$-time. Therefore, we only give a high level presentation of the procedure.

4) Procedure SplitUpdateData: The purpose of this procedure is to maintain the data structures coherent after a split of a block C in two subblocks, the new C and a new D. Since this procedure only modifies the Rel's and NotRel's, we will only look at (9) and (16).

The Rel's are updated at lines 2 and 3. Let $\mathscr{R}'$ be the value of $\mathscr{R}$ before the split, let $\mathscr{R}''$ be its value after the split and let N be the node associated with the block C before it was split. Before the split, we have $N = C$.node and after the split we have $N = C$.node $\cup D$.node. With line 2, we have, for each block $E \in P$, the equivalence $N \times E$.node $\subseteq \mathscr{R}' \Leftrightarrow (C$.node $\cup D$.node$) \times E$.node $\subseteq \mathscr{R}''$ and with line 3 we have the equivalence E.node $\times N \subseteq \mathscr{R}' \Leftrightarrow$ E.node $\times (C$.node $\cup D$.node$) \subseteq \mathscr{R}''$. And thus $\mathscr{R}$ has not changed since the other products E.node $\times E'$.node in its definition have not changed. For one call of SplitUpdateData these two lines are executed in $O(|P|)$-time. Since SplitUpdateData is called only when a block is split and since there is, at the end, at most $|P_{\text{sim}}|$ blocks. The overall time complexity of these two lines is in $O(|P_{\text{sim}}|^2)$.

The RelCount's are updated in the other lines of the procedure. In (16), the split can involve three blocks: B', E or E'. Let us remember that $\mathscr{R}$ is not changed during this procedure.

Line 5 treats the case where the split block is a B'. Since $\mathscr{R}$ is a preorder, after lines 2 and 3, we have $C \in D$.Rel and $D \in C$.Rel. It is therefore normal that for any $E \in P$ we have D.RelCount(E) $= C$.RelCount(E) since $\mathscr{R}$ is a preorder. The overall time complexity of line 5 is thus in $O(|P_{\text{sim}}|^2)$.

If in (16) the split block is E, the test at line 6 determines the block X among the new C and D that did not inherit E.rep (which is C.rep at this line since C keeps the identity of the parent block, the old C and thus E). This means that we will have to initialise B'.RelCount(X) for all $B' \in P$. This is done after at lines 21 to 27. For now, if D has inherited E.rep we do B'.RelCount(D) $:= B'$.RelCount(E) for all $B' \in P$. Remember, at this stage, we have B'.RelCount(E) $= B'$.RelCount(C).

Lines 12 to 20 treat the case where the split block is E'. We thus have E'.node $= C$.node $\cup D$.node. There is three alternatives for a given block E: either E.node $\rightarrow C$.node, or E.node $\rightarrow D$.node, or both. For the two first alternatives B'.RelCount(E) does not change. But for the third one, we have to increment this count by one. Apart for lines 17 to 19 the overall time complexity of these lines is in $O(|P_{\text{sim}}|.|\rightarrow|)$. Remember, SplitUpdateData is called only when a block is split and this occurs at most $|P_{\text{sim}}|$ times. To correctly analyze the time complexity of lines 17 to 19, for a state e let us first define $\rightarrow(e)_P \triangleq \{E' \in P \,|\, e \rightarrow E'\}$. This set $\rightarrow(e)_P$ is a partition of $\rightarrow(e)$, the set of the successors of e. Then, each time a state e is involved at line 16 this means that a block E' in $\rightarrow(e)_P$ has been split in C and in D. For a given state e there can be at most $|\rightarrow(e)|$ splits of $\rightarrow(e)_P$. Hence, the sum of the sizes of $Touched'$ for all executions of SplitUpdateData is in $O(|\rightarrow|)$. Furthermore, for one execution of this procedure, the time complexity of lines 18–19 is in $O(|P_{\text{sim}}|)$. Therefore, the overall time complexity of lines 17 to 19 is in $O(|P_{\text{sim}}|.|\rightarrow|)$.

Lines 21 to 28 treat the case where the split block is E. The block E has been split in C and D. One of them contains

Function Init($Q, \to, P_{\text{init}}, R_{\text{init}}, RefinerNodes$)

```
1  P := copy(P_init) ;
2  foreach E ∈ P do
3  |  E.count := 0 ; E.rep := ChooseState(E) ;
4  |  E.node := {q ∈ Q | q.block = E} ;
5  |  E.Rel := {E' ∈ P | (E, E') ∈ R_init} ;
   // Initialization to take into account
      Proposition 9.
6  Marked := ∅ ; Touched := ∅ ;
7  foreach q → q' do Marked := Marked ∪ {q} ;
8  Split(P, Marked) ;
9  foreach E ∈ P | E.rep ∈ Marked do
10 |  Touched := Touched ∪ {E} ;
11 foreach C ∈ Touched do
12 |  foreach D ∉ Touched do C.Rel := C.Rel \ {D} ;
13 Marked := ∅ ; Touched := ∅ ;
   // Initialization of RefinerNodes and the
      NotRel 's.
14 foreach C ∈ P do
15 |  C.node.NotRel' := ∅ ;
16 |  C.node.NotRel := {D.node | D ∉ C.Rel} ;
17 |  if C.node.NotRel ≠ ∅ then
18 |  |  RefinerNodes := RefinerNodes ∪ {C.node} ;
   // Initialization of the RelCount's.
19 foreach E, B' ∈ P do B'.RelCount(E) := 0 ;
20 PreE' := ∅ ;
21 foreach E' ∈ P do
22 |  foreach e ∈ →^-1 (E'.node) | e = e.block.rep do
23 |  |  PreE' := PreE' ∪ {e.block} ;
24 |  foreach E ∈ PreE', B' ∈ P do
25 |  |  B'.RelCount(E)++ ;
26 |  PreE' := ∅ ;
27 return (P) ;
```

Procedure Split1($P, RefinerNodes$)

```
1  Marked := ∅ ; atLeastOneSplit := false ;
2  foreach B ∈ RefinerNodes do
3  |  B' = ChooseBlock(B) ;
4  |  foreach e ∈ →^-1 (B) do
5  |  |  if B'.RelCount(e.block) = 0 then
6  |  |  |  atLeastOneSplit := true ;
7  |  if atLeastOneSplit = true then
8  |  |  foreach q → q' | q'.block ∈ B'.Rel do
9  |  |  |  Marked := Marked ∪ {q} ;
10 |  |  Split(P, Marked) ;
11 |  |  Marked := ∅ ; atLeastOneSplit := false ;
```

$E.rep$. It is therefore not necessary to recompute the counters associated with it (although these counters have been possibly updated at lines 12 to 20). The variable X represents the block, C or D, which has not inherited E.rep. Therefore, we have to initialize the counters associated with it. Note that lines 21 to 28 are executed at most once for each block. Therefore, apart from the nested loops at lines 25–27, the overall time complexity of them is in $O(|P_{\text{sim}}|.|\to|)$. For the nested loops, we have to observe that the size of *Touched* is less than the number of outgoing transitions from X.rep and those transitions are considered only once during the execution of the algorithm. Therefore the overall time complexity of the nested loops is also in $O(|P_{\text{sim}}|.|\to|)$.

All of this implies that the overall time complexity of Procedure SplitUpdateData is in $O(|P_{\text{sim}}|.|\to|)$.

5) Function Init: This function initializes the data structures and transforms the initial preorder such that we start from a preorder stable with the total relation $\mathscr{R}_0 = Q \times Q$. The first lines require no special comment except that the Rel array for each block E is initialized according to (9). The time complexity of these lines 1–5 is in $O(|P_{\text{sim}}|^2)$.

Lines 6–13 transform the initial preorder according to Proposition 9. Note that the call of Function Split at line 8 has the side effect to transform the counters before their initialization. This is not a problem since this does not change the overall time complexity and the real initialization of the counters is done after, at lines 19–26. Just after line 13, with Proposition 9, the invariants (11) is true for $i = 1$ and each simulation included in the preorder represented by the partition-relation pair $(P_{\text{init}}, R_{\text{init}})$ is included in $\mathscr{R}_1$. Apart from the inner call of Procedure SplitUpdateData, whose we know the overall time complexity, $O(|P_{\text{sim}}|.|\to|)$, the time complexity of these lines is in $O(|\to| + |P_{\text{sim}}|^2)$.

The loop starting at line 14 initializes the NotRel's according to (10) such that (13) is also true for $i = 1$. The set *RefinerNodes* is also initialized according to (14) for $i = 1$. The time complexity of this loop is in $O(|P_{\text{sim}}|^2)$.

Lines 19–26 initialize the counters such that for all $E, B' \in P$ we have:

$$B'.\mathsf{RelCount}(E) = |\{E' \in P \,|\, E.\mathsf{rep} \to E'.\mathsf{node}\}|$$

The invariant (15) is thus true for $i = 1$ since $B'.\mathsf{node} \times E'.\mathsf{node} \subseteq \mathscr{R}_0$ is always true, with $\mathscr{R}_0 = Q \times Q$. The time complexity of the loop at line 19 is in $O(|P_{\text{sim}}|^2)$. The time complexity of the loop at line 22 is in $O(|P_{\text{sim}}|.|\to|)$. An iteration of the loop at line 24 corresponds to a meta-transition $E.\mathsf{node} \to E'.\mathsf{node}$ and a block $B' \in P$. Its time complexity is therefore in $O(|P_{\text{sim}}|.|\to|)$.

6) Procedure Split1: The purpose of this procedure is to apply Lemma 14 to split the current partition P until there is no $(\mathscr{P}, \mathscr{R})$-splitter transition of type 1, with $\mathscr{P} = \mathscr{P}_P$ and $\mathscr{R}$ defined by (9). This is done such that the coarsest $\mathscr{R}$-block-stable equivalence relation included in $\mathscr{P}_P$ before the execution of Split1 is still included in $\mathscr{P}_P$ after its execution.

Procedure Split2($P, RefinerNodes$)

```
1  PreB' := ∅ ; Touched := ∅ ; Touched' := ∅ ; Marked := ∅ ;
2  foreach B ∈ RefinerNodes do
3      foreach B' ∈ P | B'.node ⊆ B do
4          foreach e ∈→^-1 (B'.node) do
5              E := e.block ;
6              if e = E.rep then
7                  PreB' := PreB' ∪ {E} ;
8          foreach E ∈ PreB' do
9              E.count++ ;
10             Touched := Touched ∪ {E} ;
11         PreB' := ∅ ;
12     B' = ChooseBlock(B) ;
13     foreach E ∈ Touched do
14         if B'.RelCount(E) = E.count then
               // The transition E.rep → B is
                  maximal
15             Touched' := Touched' ∪ {E} ;
16         E.count := 0 ;
17     foreach e ∈→^-1 (B) | e.block ∈ Touched' do
18         Marked := Marked ∪ {e} ;
19     Split (P, Marked) ;
20     Touched := ∅ ; Touched' := ∅ ; Marked := ∅ ;
```

Procedure Refine($RefinerNodes$)

```
1  Remove := ∅ ; RefinerNodes' := ∅ ;
2  foreach B ∈ RefinerNodes do
3      B' = ChooseBlock(B) ;
4      foreach d ∈→^-1 (∪B.NotRel) do
5          D := d.block ;
6          if B'.RelCount(D) = 0 then
7              Remove := Remove ∪ {D} ;
8      foreach c ∈→^-1 (B) do
9          C := c.block ;
10         foreach D ∈ Remove | D ∈ C.Rel do
11             C.Rel := C.Rel \ {D} ;
12             C.node.NotRel' :=
               C.node.NotRel' ∪ {D.node} ;
13             RefinerNodes' := RefinerNodes' ∪ {C.node}
               ;
14     Remove := ∅ ; B.NotRel := ∅ ;
15 RefinerNodes := ∅ ;
16 foreach B ∈ RefinerNodes' do
17     swap(B.NotRel, B.NotRel') ;
18 swap(RefinerNodes, RefinerNodes') ;
```

Proposition 20 guarantees that all splitter transitions of type 1 have been treated since we assume (14). Note that at line 3, B is a node, a set of states, which corresponds to a block of the relation $\mathscr{R}$. As explained in section V-B, the counters are defined between two blocks of the current partition P. We thus need to choose one of this block included in B to represent B since we have Proposition 12.

A transition $e \to e'$ with $e' \in B$ at line 4 is considered only if there is a block in $\cup B$.NotRel. From Lemma 22 this can happen only $|P_{\text{sim}}|$ times. Therefore, the overall time complexity of the loop at line 4 is in $O(|P_{\text{sim}}|.| \to |)$. From Lemma 14, lines 8–11 are executed only when at least one block is split. Therefore, their overall time complexity is in $O(|P_{\text{sim}}|.| \to |)$.

7) Procedure Split2: This procedure is applied after Split1. We can therefore assume that there is no more $(\mathscr{P}, \mathscr{R})$-splitter transition of type 1, with $\mathscr{P}$ and $\mathscr{R}$ defined in the analysis of Split1. The aim of this procedure is to implement Lemma 18. It works as follows. For a given node B which corresponds to a block of $\mathscr{R}$ such that B.NotRel $\neq \emptyset$ we scan the blocks B' of P which are included in B. For each of those B' we scan, loop at line 4, the incoming transitions from representatives states of blocks E. For each of these blocks we increment E.count, loop at line 8. Therefore, at the end of the loop at line 3, for each block $E \in P$ such that $E \to B$, we have $E.\text{count} = |\{[b]_{\mathscr{P}} \subseteq B \,|\, E.\text{rep} \to b\}|$. This allows us to identify, loop at line 13, the blocks that may be split according to Lemma 18. The split is done thanks to lines17–19. For reasons similar to those for Procedure Split1, we have: the overall time complexity of Procedure Split2 is in $O(|P_{\text{sim}}|.| \to |)$.

8) Procedure Refine: Procedures Split1 and Split2 possibly change the current partition P, but not $\mathscr{R}$, and preserve (10), (11), (12), (13), (14) and (16). Thanks to Lemma 14 and Lemma 18 all $\mathscr{R}$-block-stable equivalence relation presents in $\mathscr{P}_P$ before the execution of Split1 and Split2 are still presents after. Furthermore, with Theorem 19, we know that $\mathscr{P}_P$, after the execution of Split2, is $\mathscr{R}$-block-stable. From all of this, before the execution of Refine, $\mathscr{P}_P$ is the coarsest $\mathscr{R}$-block-stable equivalence relation. The conditions are thus met to apply Theorem 7 to do a refine step of the algorithm. Note that, thanks to (16) and Proposition 12, we have the equivalence: $d \notin \to^{-1} \circ \mathscr{R}(B) \Leftrightarrow B'$.RelCount($D$) $= 0$ at line 6. At the end of the while loop at line 2, the relation $\mathscr{R}$ has been refined by $NotRel'$. In lines 15–18 $NotRel$ is set to $NotRel'$ and $RefinerNodes$ is set to $RefinerNodes'$ to prepare the next iteration of the while loop in Procedure Sim. By construction, see the loop starting at line 10, (13) and (14) are set for the next iteration of the while loop in Sim. In a similar way, the RelCount's are not modified by Refine, (16) is thus still true and (15) will be true for the next iteration of the while loop in Sim. Thanks to Theorem 7, (11) and (12) are also preserved.

From Lemma 22, a node B and a transition $d \to d'$ with $d' \in \cup B$.NotRel are considered at most once during the execution of the algorithm. Therefore, the overall time complexity of the loop at line 4 is in $O(|P_{\text{sim}}|.| \to |)$. Let us now consider a block D in $Remove$ and a transition $c \to b$ with $b \in B$. By contradiction, let us suppose this pair $(D, c \to b)$ can happen

twice, at iteration i and at iteration j of the while loop of Function Sim, with $i < j$. From (13) and line 4 we have for $k = i$ and $k = j$: $D \to \mathscr{R}_{k-1}(b)$ and $D \not\to \mathscr{R}_k(b)$. But this is not possible since $(\mathscr{R}_i)_{i\geq 0}$ is a strictly decreasing sequence of relations. This means that the overall time complexity of the loop at line 8 is also in $O(|P_{\text{sim}}|.|\to|)$. The other lines have a lower overall time complexity. From all of this, the overall time complexity of this procedure is in $O(|P_{\text{sim}}|.|\to|)$.

9) Time Complexity of the Algorithm: From the analysis of the functions and procedures of the algorithm, we derive the following theorem.

Theorem 23. *The time complexity of the presented simulation algorithm is in* $O(|P_{\text{sim}}|.|\to|)$.

VI. Improvements and Future Work

With the new notions of maximal transition, stable preorder, block-stable equivalence relation and representative state, we have introduced new foundations that we will use to design some efficient simulation algorithms. This formalism has been illustrated with the presentation of the most efficient in memory of the fastest simulation algorithms of the moment.

It is possible to increase in practice the time efficiency of procedures SplitUpdateData, Split1 and Split2 if we allow the use of both $\to$ and $\to^{-1}$ (or if we calculate one from the other, which requires an additional bit space in $O(|\to|.\log(|Q|))$ to store the result). Procedures SplitUpdateData and SimUpdateData can be further improved in practice, but this changes $O(|P_{\text{sim}}|^2.\log(|P_{\text{sim}}|))$ to $O(|P_{\text{sim}}|^2.\log(|Q|))$, which is not really noticeable in practice, in the bit space complexity, if we count states instead of blocks in (7), (15) and (16).

Simulation algorithms are generally extended for labeled transition systems (LTS) by embedding them in normal transition systems. This is what is proposed in [10], [7] and [12] for example. By doing this, the size of the alphabet is introduced in both time and space complexities. Even in [18], where a more specific algorithm is proposed for LTS, the size of the alphabet still matter. In [13] we proposed three extensions of [10] for LTS with significant reduction of the incidence of the size of the alphabet. We will therefore propose the same extensions but from the foundations given in the present paper. Note that the bit space complexity of the algorithm presented here is in fact in $\Theta(|P_{\text{sim}}|^2.\log(|P_{\text{sim}}|)+|Q|.\log(|Q|))$. It is therefore interesting to propose other algorithms with better compromises between time and space complexities. We will therefore compare in practice different propositions.

Then, we will have all the prerequisites to address a more challenging problem that we open here: the existence of a simulation algorithm with a time complexity in $O(|\to|.\log(|Q|)+|P_{\text{sim}}|.|\to_{\text{sim}}|)$, with $\to_{\text{sim}}$ the relation over P_{sim} induced by $\to$, and a bit space complexity in $O(|P_{\text{sim}}|^2.\log(|P_{\text{sim}}|)+|\to|.\log(|Q|))$. What is surprising is that the biggest challenge is not the part in $O(|P_{\text{sim}}|.|\to_{\text{sim}}|)$ but the part in $O(|\to|.\log(|Q|))$ in the time complexity. Such an algorithm will lead to an even greater improvement from the algorithm of the present paper than that of the passage from HHK to RT since there are, in general, many more transitions than states in a transition system.

Acknowledgements

I thank the anonymous reviewers. Their questions and suggestions have helped to improve the presentation of the paper. I am also grateful to Philippe Canalda for his helpful advice.

References

[1] R. Milner, "An algebraic definition of simulation between programs," in *IJCAI*, pp. 481–489, 1971.

[2] P. A. Abdulla, Y.-F. Chen, L. Holík, R. Mayr, and T. Vojnar, "When simulation meets antichains," in *TACAS* (J. Esparza and R. Majumdar, eds.), vol. 6015 of *Lecture Notes in Computer Science*, pp. 158–174, Springer, 2010.

[3] G. Cécé and A. Giorgetti, "Simulations over two-dimensional on-line tessellation automata," in *Developments in Language Theory* (G. Mauri and A. Leporati, eds.), vol. 6795 of *Lecture Notes in Computer Science*, pp. 141–152, Springer, 2011.

[4] O. Grumberg and D. E. Long, "Model checking and modular verification," *ACM Trans. Program. Lang. Syst.*, vol. 16, no. 3, pp. 843–871, 1994.

[5] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke, "Computing simulations on finite and infinite graphs," in *FOCS*, pp. 453–462, IEEE Computer Society, 1995.

[6] D. Bustan and O. Grumberg, "Simulation-based minimization," *ACM Trans. Comput. Logic*, vol. 4, no. 2, pp. 181–206, 2003.

[7] R. Gentilini, C. Piazza, and A. Policriti, "From bisimulation to simulation: Coarsest partition problems," *J. Autom. Reasoning*, vol. 31, no. 1, pp. 73–103, 2003.

[8] R. J. van Glabbeek and B. Ploeger, "Correcting a space-efficient simulation algorithm," in *CAV* (A. Gupta and S. Malik, eds.), vol. 5123 of *Lecture Notes in Computer Science*, pp. 517–529, Springer, 2008.

[9] F. Ranzato and F. Tapparo, "A new efficient simulation equivalence algorithm," in *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings*, pp. 171–180, IEEE Computer Society, 2007.

[10] F. Ranzato and F. Tapparo, "An efficient simulation algorithm based on abstract interpretation," *Inf. Comput.*, vol. 208, no. 1, pp. 1–22, 2010.

[11] S. Crafa, F. Ranzato, and F. Tapparo, "Saving space in a time efficient simulation algorithm," *Fundam. Inform.*, vol. 108, no. 1-2, pp. 23–42, 2011.

[12] R. Gentilini, C. Piazza, and A. Policriti, "Rank and simulation: the well-founded case," *J. Log. Comput.*, vol. 25, no. 6, pp. 1331–1349, 2015.

[13] G. Cécé, "Three simulation algorithms for labelled transition systems," *CoRR*, vol. http://arxiv.org/abs/1301.1638, 2013.

[14] F. Ranzato, "An efficient simulation algorithm on kripke structures," *Acta Inf.*, vol. 51, no. 2, pp. 107–125, 2014.

[15] R. Paige and R. E. Tarjan, "Three partition refinement algorithms," *SIAM J. Comput.*, vol. 16, no. 6, pp. 973–989, 1987.

[16] G. Cécé, "Foundation for a series of efficient simulation algorithms," *arXiv.org*, 2017.

[17] A. Valmari and P. Lehtinen, "Efficient minimization of dfas with partial transition," in *STACS* (S. Albers and P. Weil, eds.), vol. 1 of *LIPIcs*, pp. 645–656, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2008.

[18] P. A. Abdulla, A. Bouajjani, L. Holík, L. Kaati, and T. Vojnar, "Computing simulations over tree automata," in *TACAS* (C. R. Ramakrishnan and J. Rehof, eds.), vol. 4963 of *Lecture Notes in Computer Science*, pp. 93–108, Springer, 2008.