Bisimulations over DLTS in O(m.log n)-time

Gérard Cece

Bisimulations over DLTS in O(m.log n)-time

Gérard Cécé

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

Abstract

The well known Hopcroft's algorithm to minimize deterministic complete automata runs in $O(kn \log n)$-time, where k is the size of the alphabet and n the number of states. The main part of this algorithm corresponds to the computation of a coarsest bisimulation over a finite Deterministic Labelled Transition System (DLTS). By applying techniques we have developed in the case of simulations, we design a new algorithm which computes the coarsest bisimulation over a finite DLTS in $O(m \log n)$-time and $O(k+m+n)$-space, with m the number of transitions. The underlying DLTS does not need to be complete and thus: $m \leq kn$. This new algorithm is much simpler than the two others found in the literature.

1 Introduction

Bisimulation and simulation equivalences are behavioral relations between processes. They are mainly used to reduce the state space of models since they preserve branching time temporal logic like CTL and CTL* for bisimulation [4] and their existential fragment for simulation [6]. Simulation can also be used as a sufficient condition for the inclusion of languages when this test is undecidable in general [3].

Let us call *coarsest bisimulation problem*, the problem of finding the coarsest bisimulation equivalence in a Labelled Transition System (LTS) and *coarsest simulation problem* the corresponding problem for simulation. One can establish a hierarchy, with increasing difficulty, from the coarsest bisimulation problem in a finite deterministic LTS, to the coarsest simulation problem in a finite non deterministic LTS. The first case was efficiently solved by Hopcroft, to minimize finite deterministic automata, in [7] with an algorithm whose complexities are $O(kn \log n)$-time and $O(kn)$-space [8], with k the size of the alphabet and n the number of states. The next important step, the coarsest bisimulation problem for finite non deterministic LTS, was partially solved by Paige and Tarjan in [9]. We say partially because it was for $k = 1$. The complexities of their algorithm are $O(m \log n)$-time and $O(m + n)$-space with m the number of transitions. An extension to the general case $k > 1$ was proposed later by Fernandez [5]. According to [11] the time complexity of the algorithm of Fernandez is $O(kn \log n)$.

In the present paper, by using techniques we have developed for the coarsest simulation problem in [2], we design now a new algorithm to avoid the k factor in the time complexity of the coarsest bisimulation problem over finite DLTS. We do this in order to obtain a time complexity of $O(m \log n)$ and a space complexity of $O(k + n + m)$.

In the literature, there are two papers which achieve the same result over finite DLTS, both are designed to minimize finite deterministic automata. The first one [11], by Valmari and Lehtinen, may be considered as the best solution using the ideas of Hopcroft, in the sense that conceptually their splitters are couples made of a block and of a letter. However, they need, beside the traditional partition of the states during the computation, a partition of the set of transitions. The second one, by Béal and Crochemore [1] is closer to our solution but is more complex with its use of "all but the largest strategy" and "signature of states".

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 $X, Y \subseteq Q$, we note $X \mathscr{R} Y$ to express the existence of two states $q, q' \in Q$ such that $(q, q') \in X \times Y \cap \mathscr{R}$. By abuse of notation, we also note $q \mathscr{R} Y$ for $\{q\} \mathscr{R} Y$, $X \mathscr{R} q'$ for $X \mathscr{R} \{q'\}$ and $q \mathscr{R} q'$ for $\{q\} \mathscr{R} \{q'\}$. In the figures we draw $X \dashrightarrow Y$ for $X \mathscr{R} Y$. We note $\mathscr{R}^{-1}$ the *inverse* of $\mathscr{R}$ such that $q \mathscr{R}^{-1} q'$ iff $q' \mathscr{R} q$. We define $\mathscr{R}(q) \triangleq \{q' \in Q \mid q \mathscr{R} q'\}$ for $q \in Q$ and $\mathscr{R}(X) \triangleq \cup_{q \in X} \mathscr{R}(q)$ for $X \subseteq Q$. Let $\mathscr{S}$ be another relation on Q, the *composition* of $\mathscr{R}$ by $\mathscr{S}$ is $\mathscr{S} \circ \mathscr{R} \triangleq \{(x, y) \in Q \times Q \mid y \in \mathscr{S}(\mathscr{R}(x))\}$. 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 *symmetric* if $\mathscr{R}^{-1} = \mathscr{R}$. The relation $\mathscr{R}$ is said *transitive* if $\mathscr{R} \circ \mathscr{R} \subseteq \mathscr{R}$. An *equivalence relation* is a reflexive, symmetric and transitive relation. For an equivalence relation $\mathscr{R}$ on Q and $q \in Q$ we call $\mathscr{R}(q)$ a *block*, the block of q. It is well known, and immediate consequences of the definitions, that for any equivalence relation $\mathscr{R}$ and any block B of $\mathscr{R}$, we have $\forall q, q' \in B \,.\, q \mathscr{R} q'$, and for any $X \subseteq Q$, $\mathscr{R}(X)$ is a, disjoint, union of blocks.

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, also called *blocks*, that are pairwise disjoint and whose union gives Q. There is a duality between a partition of Q and an equivalence relation on Q since from a partition P we can derive the equivalence relation $\mathscr{R}_P = \cup_{B \in P} B \times B$ and from an equivalence relation $\mathscr{R}$ we can derive a partition $P_{\mathscr{R}} = \cup\{\mathscr{R}(q) \subseteq Q \mid q \in Q\}$.

Let $T = (Q, \Sigma, \rightarrow)$ be a triple such that Q is a set of elements called *states*, Σ is an *alphabet*, a set of elements called *letters* or *labels*, and $\rightarrow \subseteq Q \times \Sigma \times Q$ 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 on Q: $\stackrel{a}{\rightarrow} \triangleq \{(q, q') \in Q \times Q \mid (q, a, q') \in \rightarrow\}$ and its reverse $\mathrm{pre}_{\stackrel{a}{\rightarrow}} \triangleq \left(\stackrel{a}{\rightarrow}\right)^{-1}$. When $\rightarrow$ is clear from the context, we simply note pre_a instead of $\mathrm{pre}_{\stackrel{a}{\rightarrow}}$. Finally, if for all $a \in \Sigma$, the relation $\stackrel{a}{\rightarrow}$ is a function (i.e. $\forall a \in \Sigma \forall q \in Q \,.\, |\stackrel{a}{\rightarrow}(q)| \leq 1$) then T is said *deterministic* and thus a DLTS.

3 Underlying Theory

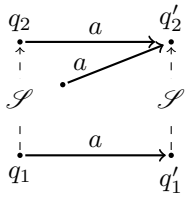
The presentation of this section is quite different from what is found in the literature. Thanks to the results it provides, I hope the reader will be convinced of its pertinence.

Let $T = (Q, \Sigma, \to)$ be a LTS. The classical definition of a simulation says that 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 $q_1 \mathscr{S} q_2$, there is a transition $q_2 \xrightarrow{a} q_2'$ such that $q_1' \mathscr{S} q_2'$. However, the following definition happens to be more effective than the classical one. As put in evidence by the picture on the right, the two definitions are clearly equivalent.

Definition 1. *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 is simulated by q'" if there is a simulation $\mathscr{S}$ over T such that $q \mathscr{S} q'$.



A *bisimulation* $\mathscr{S}$ is just a simulation such that $\mathscr{S}^{-1}$ is also a simulation. For a bisimulation $\mathscr{S}$, two states q and q' such that $q \mathscr{S} q'$ are said *bisimilar*.

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 bisimulation expressed over blocks.

Proposition 2. *Let $T = (Q, \Sigma, \to)$ be a LTS and $\mathscr{S}$ be an equivalence relation on Q. The relation $\mathscr{S}$ is a bisimulation over T if and only if for all block B of $\mathscr{S}$ we have:* $\forall a \in \Sigma \,.\, \mathscr{S} \circ \mathrm{pre}_a(B) \subseteq \mathrm{pre}_a(B)$.

Proof. If $\mathscr{S}$ is a bisimulation, by definition it is a simulation. Then, we have for any $B \subseteq Q$: $\forall a \in \Sigma \,.\, \mathscr{S} \circ \mathrm{pre}_a(B) \subseteq \mathrm{pre}_a(\mathscr{S}(B))$. This inclusion is thus also true for a block $B = \mathscr{S}(q)$ for a given $q \in Q$. With the transitivity of an equivalence relation, we get $\mathscr{S}(B) = \mathscr{S} \circ \mathscr{S}(q) = \mathscr{S}(q) = B$. All of this implies: $\forall a \in \Sigma \,.\, \mathscr{S} \circ \mathrm{pre}_a(B) \subseteq \mathrm{pre}_a(B)$.

In the other direction, let us consider a given $q \in Q$. Let $B = \mathscr{S}(q)$. An equivalence relation is reflexive. Therefore, we get: $q \in B$. Then,

$$\mathscr{S} \circ \mathrm{pre}_a(q) \subseteq \mathscr{S} \circ \mathrm{pre}_a(B) \subseteq \mathrm{pre}_a(B) = \mathrm{pre}_a \circ \mathscr{S}(q) \ .$$

Which implies that $\mathscr{S}$ is a simulation. Since $\mathscr{S}$ is an equivalence relation we have $\mathscr{S} = \mathscr{S}^{-1}$ and thus $\mathscr{S}^{-1}$ is also a simulation. This ends the proof. □

Let $\mathscr{R}$ be an equivalence relation on Q. Thanks to the preceding proposition, if $\mathscr{R}$ is not a bisimulation over T, there is $a \in \Sigma$ and B a block of $\mathscr{R}$ such that $\mathscr{R} \circ \mathrm{pre}_a(B) \not\subseteq \mathrm{pre}_a(B)$. This implies the existence of a subset of Q, let us call it $\mathrm{Remove}_a(B)$, such that:

$$\mathscr{R} \circ \mathrm{pre}_a(B) \subseteq \mathrm{pre}_a(B) \cup \mathrm{Remove}_a(B) \ . \tag{1}$$

The problem with (1) is that $\mathrm{Remove}_a(B)$ depends on a letter which we want to avoid in order to obtain an algorithm whose complexity does not depend on the size of the alphabet. It would therefore be more interesting to have something like:

$$\mathscr{R} \circ \mathrm{pre}_a(B) \subseteq \mathrm{pre}_a(B \cup \mathrm{NotRel}(B)) \ . \tag{2}$$

But, this is possible only if $\mathscr{R} \circ \mathrm{pre}_a(B) \subseteq \mathrm{pre}_a(Q)$. A sufficient condition is:

$$\forall a \in \Sigma \,.\, \mathscr{R} \circ \mathrm{pre}_a(Q) \subseteq \mathrm{pre}_a(Q) \ . \tag{3}$$

The fact is that (3) is not a real restriction because any bisimulation included in $\mathscr{R}$ satisfies this condition: a state which has an outgoing transition labelled by a letter a can be bisimilar only with states which have at least one outgoing transition labelled by a. This is exactly what (3) says about $\mathscr{R}$. Surprisingly, all of this is well known, but we have found no algorithm that uses this as an optimization during an initialization phase. For the present paper this not an optimization, this is a necessity.

The next definition and lemma establish that we can restrict our problem of finding the coarsest bisimulation inside an equivalence relation $\mathscr{R}$ to the problem of finding the coarsest bisimulation inside an equivalence relation $\mathscr{R}$ that satisfies (3).

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

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

Lemma 4. *Let $T = (Q, \Sigma, \to)$ be a LTS and $\mathscr{U} = \mathrm{InitRefine}(\mathscr{R})$ with $\mathscr{R}$ an equivalence relation on Q. Then, $\mathscr{U}$ is an equivalence relation on Q and all bisimulation over T included in $\mathscr{R}$ is included in $\mathscr{U}$.*

Proof.

- Since $\mathscr{R}$ is an equivalence relation and thus reflexive, $\mathscr{U}$ is also trivially reflexive. Its definition being symmetric, $\mathscr{U}$ is also symmetric. 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 an equivalence relation, 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 only one of $\{q_1, q_3\}$ belongs to $\mathrm{pre}_a(Q)$. Let us suppose we have $q_1 \in \mathrm{pre}_a(Q)$ and $q_3 \notin \mathrm{pre}_a(Q)$. The problem is that $q_1 \in \mathrm{pre}_a(Q)$ 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 $q_3 \notin \mathrm{pre}_a(Q)$.

- Let us suppose the existence of a bisimulation $\mathscr{S}$ included in $\mathscr{R}$ but not in $\mathscr{U}$. 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 $\mathscr{S} \subseteq \mathscr{R}$ 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 only one of $\{q_1, q_2\}$ belongs to $\mathrm{pre}_a(Q)$. Let us suppose we have $q_1 \in \mathrm{pre}_a(Q)$ and $q_2 \notin \mathrm{pre}_a(Q)$. 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 $q_2 \notin \mathrm{pre}_a(Q)$.

□

The conjunction of (3) and of the trivial condition $\mathscr{R}(Q) = Q$ is the cornerstone of what follows.

Definition 5. *Let $T = (Q, \Sigma, \to)$ be a LTS and $\mathscr{R}$ be an equivalence relation on Q.*

- *A subset L of Q is said a* potential splitter *of $\mathscr{R}$ if $\mathscr{R}(L) = L$, L is composed of at least two blocks of $\mathscr{R}$ and: $\forall a \in \Sigma \,.\, \mathscr{R} \circ \mathrm{pre}_a(L) \subseteq \mathrm{pre}_a(L)$.*

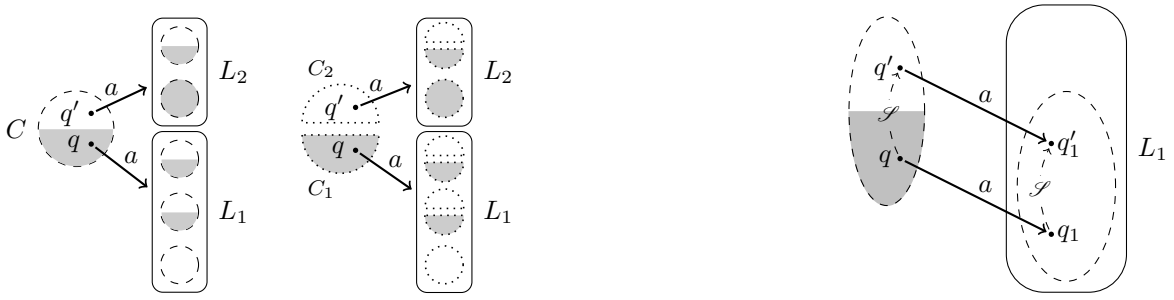
- *a couple* (L_1, L_2) *of two non empty subsets of* Q *is said a* splitter *of* $\mathscr{R}$ *if:* $L_1 \cap L_2 = \emptyset$, $\mathscr{R}(L_1) = L_1$, $\mathscr{R}(L_2) = L_2$ *and:* $\forall a \in \Sigma \,.\, \mathscr{R} \circ \mathrm{pre}_a(L_1 \cup L_2) \subseteq \mathrm{pre}_a(L_1 \cup L_2)$.

The key element of the preceding definition is that a splitter does not depend on a specific letter. This is in sharp contrast to what is generaly found in the literature since [7] (to our knowledge, there is only one exception: [1]). The second difference with the literature is that our splitters are couples of sets of blocks. Indeed, in the literature, the second element of our splitters is hidden. However, its presence is essential as it allows us to split sequentially, with different letters, the current relation with the same splitter. Furthermore, it will give us more freedom in the choice of the splitter to use.

From now on, we consider that $T = (Q, \Sigma, \to)$ is a DLTS, $\mathscr{R}$ is an equivalence relation on Q and (L_1, L_2) is a splitter of $\mathscr{R}$. We define for $X \subseteq Q$:

$$\mathrm{Split}(X, \mathscr{R}) \triangleq \mathscr{R} \setminus \cup_{q \in X} \{(q, q'), (q', q) \in Q \times Q \,\big|\, q' \in \mathscr{R}(q) \wedge q' \notin X\} \ .$$

$\mathrm{Split}(X, \mathscr{R})$ amounts to split all blocks C such that $C \cap X \neq \emptyset$ and $C \setminus X \neq \emptyset$ in two blocks $C_1 = C \cap X$ and $C_2 = C \setminus X$, see Fig. 1(a). When in an equivalence relation $\mathscr{R}$ we just split some blocks C in two parts, the resulting relation is still an equivalence relation. Therefore, $\mathrm{Split}(X, \mathscr{R})$ is still an equivalence relation. Furthermore, a block of $\mathscr{R}$ whose all elements are in X or for which no element is in X is still a block in $\mathrm{Split}(X, \mathscr{R})$.



(a) From $\mathscr{R}$ to $\mathscr{U}$.

(b) Contradiction of $\mathscr{S} \subseteq \mathscr{R}$ and $\mathscr{S} \nsubseteq \mathscr{U}$.

Figure 1: $\mathscr{U} = \mathrm{Split}(\mathrm{pre}_a(L_1), \mathscr{R})$.
Blocks of $\mathscr{R}$ are dashed, blocks of $\mathscr{U}$ are dotted, $\mathrm{pre}_a(L_1)$ is in gray.

Let $\mathscr{U} = \mathrm{Split}(\mathrm{pre}_a(L_1), \mathscr{R})$ and let us consider a block C of $\mathscr{R}$ which contains an element q in $\mathrm{pre}_a(L_1)$ and an element q' not in $\mathrm{pre}_a(L_1)$, see Fig. 1(a). Since $C \xrightarrow{a} (L_1 \cup L_2)$, by definition of a splitter, we have $C \subseteq \mathrm{pre}_a(L_1 \cup L_2)$ and thus q' is in $\mathrm{pre}_a(L_2)$. This implies that $\mathrm{Split}(\mathrm{pre}_a(L_1), \mathscr{R}) = \mathrm{Split}(\mathrm{pre}_a(L_2), \mathscr{R})$ and in no block of $\mathscr{U}$ there is a state belonging in $\mathrm{pre}_a(L_1)$ and another state not belonging in $\mathrm{pre}_a(L_1)$. Therefore, $\mathscr{U} \circ \mathrm{pre}_a(L_1) \subseteq \mathrm{pre}_a(L_1)$ and, by symmetry, $\mathscr{U} \circ \mathrm{pre}_a(L_2) \subseteq \mathrm{pre}_a(L_2)$. This is illustrated by the right part of Fig. 1(a).

Now, let us suppose there is a bisimulation $\mathscr{S}$ included in $\mathscr{R}$ but not included in $\mathscr{U}$, see Fig. 1(b). This means that there are two states q and q' of the same block of $\mathscr{R}$ such that $q \,\mathscr{S}\, q'$, $q \in \mathrm{pre}_a(L_1)$ and $q' \notin \mathrm{pre}_a(L_1)$. Let q_1 be the state of L_1 such that $q \in \mathrm{pre}_a(q_1)$. We thus have $q' \in \mathscr{S} \circ \mathrm{pre}_a(q_1)$. With the fact that $\mathscr{S}$ is a simulation, we infer that $q' \in \mathrm{pre}_a \circ \mathscr{S}(q_1)$ which implies the existence of a state q'_1 such that $q_1 \,\mathscr{S}\, q'_1$

and $q' \in \mathrm{pre}_a(q'_1)$. But, remember, we have $\mathscr{S} \subseteq \mathscr{R}$, $q_1 \in L_1$ and $\mathscr{R}(L_1) = L_1$. Therefore, $q'_1 \in L_1$ and $q' \in \mathrm{pre}_a(L_1)$. Which contradicts $q' \notin \mathrm{pre}_a(L_1)$.

From what precedes, we infer the following theorem.

Theorem 6. *Let $T = (Q, \Sigma, \to)$ be a DLTS and (L_1, L_2) be a splitter of $\mathscr{R}$ an equivalence relation on Q. Let $\mathscr{U} = \mathrm{Split}(\mathrm{pre}_a(L_1), \mathscr{R})$. Then, $\mathscr{U}$ is an equivalence relation and any bisimulation included in $\mathscr{R}$ is also included in $\mathscr{U}$. Furthermore: $\mathscr{U} \circ \mathrm{pre}_a(L_1) \subseteq \mathrm{pre}_a(L_1)$ and $\mathscr{U} \circ \mathrm{pre}_a(L_2) \subseteq \mathrm{pre}_a(L_2)$.*

Let us come back to equation (1) and consider Fig. 1(a). We have $\mathscr{R} \circ \mathrm{pre}_a(L_1) \subseteq \mathrm{pre}_a(L_1) \cup \mathrm{Remove}_a(L_1)$ with $\mathrm{Remove}_a(L_1) = \mathrm{pre}_a(L_2)$ which clearly depends on a. With equation (2) we have: $\mathscr{R} \circ \mathrm{pre}_a(L_1) \subseteq \mathrm{pre}_a(L_1 \cup \mathrm{NotRel}(L_1))$ with $\mathrm{NotRel}(L_1) = L_2$ which does not depend on any letter. This was our goal.

From last theorem, the algorithm will maintain a set of potential splitters. At each main iteration, we choose L one of them. Then, we choose in L a block B, no matter which one. We set L_1 to be the smallest set between B and $L \setminus B$; we set L_2 to be the other one. Then, we iteratively split the current partition, with the non empty $\mathrm{pre}_a(L_1)$. The magical thing is that (L_1, L_2) remains a splitter during these iterations even if some blocks in L_1 or in L_2 are split. Management of the letters being aside, the resulting algorithm is quite similar to the one in [9]. However, knowing that the LTS being deterministic, there is no need of counters like in [9].

4 The Bisimulation Algorithm

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

1 $P := \mathrm{copy}(P_{\text{init}})$; **forall** $a \in \Sigma$ **do** $a.\mathsf{Pre} := \emptyset$;
2 **forall** $q \xrightarrow{a} q' \in \to$ **do** $a.\mathsf{Pre} := a.\mathsf{Pre} \cup \{q\}$
3 **forall** $a \in \Sigma$ **do** $(P, \_) :=$ `Split`$(a.\mathsf{Pre}, P)$;
4 **return** (P)

In the remainder of the paper, all LTS are finite and deterministic. Given a DLTS $T = (Q, \Sigma, \to)$ and an initial partition P_{init} of Q, inducing an equivalence relation $\mathscr{R}_{\text{init}}$, the algorithm manages a set S of potentials splitters to iteratively refine the partition P initially equals to P_{init}. At the end, P represents P_{bis} the partition whose induced equivalence relation $\mathscr{R}_{\text{bis}}$ is the coarsest bisimulation over T included in $\mathscr{R}_{\text{init}}$.

The partition P is a set of blocks. A block is assimilated with its set of states. The set S is a set of subsets of Q. We will see that S is a set of potential splitters. To each letter $a \in \Sigma$ we associate a set of states noted $a.\mathsf{Pre}$ since it corresponds to $\mathrm{pre}_a(Q)$ after the **forall** loop at line 2 of `Init` and to $\mathrm{pre}_a(\mathit{Smaller})$ after the **forall** loop at line 9 of `DBisim`.

The main function `DBisim` uses two others functions: `Split`, and `Init`. Function `Split`(X, P) splits each block C of P, having at least one element in X and another one not in X, in two blocks $C_1 = C \cap X$ and $C_2 = C \setminus X$, and returns the resulting partition and the list of blocks C that have been split. It is mainly an implementation of function

Function `DBisim`(T, P_{init}) with $T = (Q, \Sigma, \to)$

```
1  P := Init(T, P_init)};
2  if |P| = 1 then return (P) /* nothing more has to be done          */};
3  S := {Q}; alph := ∅;  forall a ∈ Σ do a.Pre := ∅;
4  while ∃L ∈ S do
5      /* Assert :   alph = ∅ ∧ (∀a ∈ Σ . a.Pre = ∅)                   */;
6      Let B be any block of L;
7      if there are only two blocks in L then S := S \ {L} else L := L \ B;
8      if |B| ≤ |L \ B| then Smaller := B else Smaller := L \ B;
9      forall q -a-> q_1 ∈-> such that q_1 ∈ Smaller do
10         alph := alph ∪ {a}; a.Pre := a.Pre ∪ {q};
11     forall a ∈ alph do
12         (P, Splitted) := Split(a.Pre, P);
13         forall C ∈ Splitted do
14             if C ⊈ ∪S then S := S ∪ {C};
15     forall a ∈ alph do a.Pre := ∅;
16     alph := ∅;
17 P_bis := P; return (P_bis)
```

Split$(X, \mathscr{R}_P)$ seen in the previous section. Function `Init`, also uses `Split` and returns the partition whose induced equivalence relation is InitRefine$(\mathscr{R}_{\text{init}})$.

4.1 Correctness

Let us first consider function `Init`(T, P_{init}). All line numbers in this paragraph refer to function `Init`. At line 2, we identify for each $a \in \Sigma$ the states which have an outgoing transition labelled by a. Then, at line 3, for each $a \in \Sigma$ we separate in all the blocks the states which have an outgoing transition labelled by a and the states that do not have an outgoing transition labelled by a. Clearly, the resulting partition corresponds to the relation InitRefine(R_{init}).

Let us consider function `DBisim`(T, P_{init}). From now on, all line numbers refer to function `DBisim`. Let $S' = \{B \in P \,\big|\, B \not\subseteq \cup S\}$. We prove, by an induction, that the following property is an invariant of the **while** loop of `DBisim`. The relation $\mathscr{R}$ is the equivalence relation induced by the current partition P.

$$L \in S \Rightarrow \mathscr{R}(L) = L \text{ and } L \text{ contains at least two blocks of } \mathscr{R} \ . \tag{4}$$

Just before the execution of the **while** loop, S contains only one element: Q. Thanks to the test at line 2, Q is made of at least two blocks, and we obviously have $\mathscr{R}(Q) = Q$, which satisfies property (4).

Let us assume (4) is satisfied before an iteration of the **while** loop. The set S can be modified only at lines 7 and 14. Let L be the element of S chosen at line 4. By induction hypothesis, L is composed of at least two blocks and $\mathscr{R}(L) = L$. If L is withdrawn from S, at line 7, this is because L is composed of exactly two blocks that are implicitly added in S'. If L is not withdrawn from S then a block, B, is withdrawn from L which is

composed, induction hypothesis and condition of the **if** at line 7, of at least three blocks. From the hypothesis that $\mathscr{R}(L) = L$ and from $\mathscr{R}(B) = B$ since B is a block, we also have $\mathscr{R}(L \setminus B) = L \setminus B$. Property (4) is therefore not modified by line 7. At line 14, a block C which has been split at line 12, and thus implicitly withdrawn from S', is added into S. Since C has just been split, and thus contains two complete blocks, property (4) is still true and thus is an invariant of the **while** loop.

Now, let us consider the two following properties:

$$\mathscr{S} \text{ is a bisimulation over } T \text{ included in } \mathscr{R}_{\text{init}} \Rightarrow \mathscr{S} \subseteq \mathscr{R} \ . \tag{5}$$

$$L \in S \cup S' \Rightarrow \forall a \in \Sigma \,.\, \mathscr{R} \circ \text{pre}_a(L) \subseteq \text{pre}_a(L) \ . \tag{6}$$

Thanks to function `Init` and Lemma 4, these properties are satisfied before the **while** loop. Remember that $S = \{Q\}$ and $S' = \emptyset$ at this moment. Let us suppose they are satisfied before an iteration of the **while** loop. Let L be the element of S chosen at line 4. After line 8, $L = L_1 \cup L_2$ with $L_1 = \mathit{Smaller}$ and $L_2 = L \setminus \mathit{Smaller}$. Since $\mathit{Smaller}$ is a block and $\mathscr{R}(L) = L$ by (4), we have: $\mathscr{R}(L_1) = L_1$ and $\mathscr{R}(L_2) = L_2$. This implies that (L_1, L_2) is a splitter of $\mathscr{R}$. Note that, during the iteration, $\mathscr{R}$ can only be refined (by function `Split`, line 12). This implies that (L_1, L_2) stays a splitter of $\mathscr{R}$ during the iteration. Furthermore, after each iteration of the **forall** loop at line 11, from Theorem 6, we have $\mathscr{S} \subseteq \mathscr{R}$ since this is the case, by induction hypothesis, before the iteration. The fact that, $\mathscr{R}$ can only be refined also implies that property (6) is still satisfied after the iteration for all elements of $S \cup S'$ different from L_1 and L_2. Let us consider their cases. Let $\mathscr{R}_c$ be the value of $\mathscr{R}$ after the iteration of the **forall** loop at line 11 for $a = c$. Then, from Theorem 6 we have $\mathscr{R}_c \circ \text{pre}_c(L_1) \subseteq \text{pre}_c(L_1)$ and $\mathscr{R}_c \circ \text{pre}_c(L_2) \subseteq \text{pre}_c(L_2)$. At the end of the iteration of the while loop, we obviously have $\mathscr{R} \subseteq \mathscr{R}_c$ and thus: $\forall a \in \mathit{alph} \,.\, (\mathscr{R} \circ \text{pre}_a(L_1) \subseteq \text{pre}_a(L_1) \wedge \mathscr{R} \circ \text{pre}_a(L_2) \subseteq \text{pre}_a(L_2))$. Let $a \notin \mathit{alph}$ this means that $\text{pre}_a(L_1) = \text{pre}_a(\mathit{Smaller}) = \emptyset$ and thus $\mathscr{R} \circ \text{pre}_a(L_1) \subseteq \text{pre}_a(L_1)$. Therefore, we have, with $\mathscr{R}'$ the value of $\mathscr{R}$ before the iteration: $\mathscr{R}' \circ \text{pre}_a(L_2) = \mathscr{R}' \circ \text{pre}_a(L)$ which is included, by induction hypothesis of (6), in $\text{pre}_a(L) = \text{pre}_a(L_2)$ and thus $\mathscr{R} \circ \text{pre}_a(L_2) \subseteq \text{pre}_a(L_2)$ at the end of the iteration. In summary, properties (5) and (6) are invariants of the **while** loop.

We postpone to the next section the proof of termination of `DBisim`. This will be done by a complexity argument. For the moment, note that the execution of the `while` loop ends when $S = \emptyset$ and thus, by definition of S', P as a set of blocs is included in S'. With (6), all of that implies, with $\mathscr{R}_{\text{bis}}$ the last value of $\mathscr{R}$: $\forall B \in P \, \forall a \in \Sigma \,.\, \mathscr{R}_{\text{bis}} \circ \text{pre}_a(B) \subseteq \text{pre}_a(B)$. By Proposition 2 this means that $\mathscr{R}_{\text{bis}}$ is a bisimulation, by (5) it is the coarsest one included in $\mathscr{R}_{\text{init}}$. Furthermore, since $\mathscr{R}_{\text{bis}} = \mathscr{R}_{P_{\text{bis}}}$ with P_{bis} a partition, the last value of P, it is an equivalence relation.

4.2 Complexities

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 element of X can be identified with its index in this array. Let $T = (Q, \Sigma, \rightarrow)$ be a LTS, an encoding of T is said *normalized* if the encodings of Q, Σ and $\rightarrow$ 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 $O(|\rightarrow|)$. If $|\Sigma|$ is not in $O(|\rightarrow|)$, we can

restrict Σ to its really used part $\Sigma' = \{a \in \Sigma \mid \exists q, q' \in Q \,.\, q \xrightarrow{a} q' \in \rightarrow\}$ whose size is less than $|\rightarrow|$. To do this, we can use hash table techniques, sort the set $\rightarrow$ with the keys being the letters labelling the transitions, or more efficiently use a similar technique of the one we used in the algorithm to distribute a set of transitions relatively to its labels (see, as an example, the **forall** loop at line 9 of `DBisim`). This is done in $O(|\Sigma| + |\rightarrow|)$ time and uses $O(|\Sigma|)$ space. We learned that this may be done in $O(|\rightarrow|)$ time, still with $O(|\Sigma|)$ space, by using a technique presented in [11] and which is also called "weak sorting" according to [1]. If $|Q|$ is not in $O(|\rightarrow|)$ this means there are states that are not involved in any transition. In general, these states are ignored. In fact, just after our initialization phase done by function `Init`, these states are in blocks that will not changed during the execution of the algorithm. Therefore, we can also restrict Q to its useful part $\{q \in Q \mid \exists q' \in Q \; \exists a \in \Sigma \,.\, q \xrightarrow{a} q' \in \rightarrow \vee\, q' \xrightarrow{a} q \in \rightarrow\}$ whose size is in $O(|\rightarrow|)$. This is done like for Σ.

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

Let us assume the following hypotheses:

- scanning the elements of one of the following sets is done in time proportional to its size: $\rightarrow$, $a.\mathsf{Pre}$ for any $a \in \Sigma$, L an element of S, B a block of P, the set of transitions leading to an element L of S or to a block B of P, and *Splitted*.
- `Split`$(a.\mathsf{Pre}, P)$ is executed in time proportional to the size of $a.\mathsf{Pre}$.
- function copy(P_{init}) is executed in $O(|Q|)$ time.
- all the other individual instructions in functions `Init` and `DBisim` are done in constant time or amortized constant time.
- all the data structures use only $O(|\rightarrow|)$ space.

Let us consider the time complexity of function `Init`. The **forall** loop at line 1 is done in time $O(|\Sigma|)$ and thus $O(|\rightarrow|)$ since T is supposed to be normalized. The **forall** loop at line 2 is done in time $O(|\rightarrow|)$. The **forall** loop at line 3 is also done in time $O(|\rightarrow|)$ since we have: $\Sigma_{a\in\Sigma}|a.\mathsf{Pre}| \leq \Sigma_{a\in\Sigma}|\xrightarrow{a}| \leq |\rightarrow|$. Therefore, `Init` is done in $O(|\rightarrow|)$-time.

Let us now consider function `DBisim`. Let us first remark that during an iteration of the `forall` loop at line 11, $|Splitted| \leq |a.\mathsf{Pre}|$. Furthermore, during an iteration of the **while** loop, $|alph|$ and $\Sigma_{a\in alph}|a.\mathsf{Pre}|$ are less than the number of transitions scanned during the `forall` loop at line 9. This implies that the time complexity of function `DBisim` is proportional to the overall number of transitions scanned at line 9. But thanks to line 8, each time a transition $q \xrightarrow{a} q'$ is used at line 9, q_1 belongs to the set *Smaller* whose size is less than half of the size of the previous time.

From all of this, we get the following theorem.

Theorem 7. *Let $T = (Q, \Sigma, \rightarrow)$ be a DLTS and P_{init} be an initial partition of Q inducing an equivalence relation $\mathscr{R}_{\text{init}}$. Function `DBisim` computes P_{bis} the partition whose corresponding equivalence relation $\mathscr{R}_{\text{bis}}$ is the coarsest bisimulation over T included in $\mathscr{R}_{\text{init}}$ in: $O(|\rightarrow| \log |Q|)$-time and $O(|\Sigma| + |\rightarrow| + |Q|)$-space.*

The $O(|Q| + |\Sigma|)$ part in the space complexity of the theorem is due to the normalization of T as explained at the beginning of this sub section.

4.3 Implementation

The data structures can be similar to what is classically used in minimization and bisimulation algorithms like those in [7, 9]. However we do prefer some ideas found in [11] and rediscovered in [2]. Instead of a doubly linked list to represent the states of a block, both papers distribute the indexes of the states in an auxiliary array A such that states in a same block form a subarray of A. Furthermore, when a block C has to be split in C_1 and C_2, the states of C_1 and C_2 stay in the same subarray of A corresponding to C. The only modification is that states of C_1 are put on the left side of that subarray and thus states of C_2 on the right side. Therefore, to encode a block, we just have to memorize a left and a right index. A key advantage for the present paper, is that to represent an element L of S we also just have to memorize a left and a right index in A. Therefore, when the blocks of L are split, we do not have to update these variables. The other elements are classic: each state knows the block to which it belongs, each block and each element of S maintains its size, each block maintains a boolean to know whether it belongs to $\cup S$ (this information is transmitted to the sub block when a block is split), S may be encoded by a file or a by a stack. To choose a block in L at line 6 we just choose the left or the right block in the subarray of A corresponding to L. Thus, it is easy to perform the instruction $L := L \setminus B$ in line 7 and to scan the elements of $Smaller$. The transitions are also initially sorted by a counting sort in order to have the transitions leading to the same state form a subarray of the array of all the transitions. This allows us to scan the transitions leading to a given state in time proportional to their number.

The variable *Splitted* is used just for the clarity of the presentation. Indeed, when a block C not in one of the potential splitters of S is split, this is detected during the call of function `Split` and C is directly added into S.

The implementation of function `Split`(X, P) is not original (see for example [2, Sect. 6] where a version which also returns the blocks in X is given). The time complexity of a call is $O(|X|)$ when the elements of X may be scan in time proportional to its size.

In [2, Sect. 6] an encoding, satisfying the hypotheses, of *alph* and the $a.\mathsf{Pre}$ is given. However, their role is just to distribute the set of transitions leading to $Smaller$ in function of their label. Then, for each label found, we scan the corresponding transitions to do the split. The distribution of the transitions can simply be done by a kind of counting sort. It runs in $O(|Smaller|)$-time and uses $O(|{\rightarrow}|)$-space. The "weak sorting" technique used in [1] can also be used.

In summary, the data structures that we use satisfy the hypotheses given at the second paragraph of Sect. 4.2 under the assumption of a normalized DLTS.

5 Main Application and Future Works

The main application of the coarsest bisimulation problem over finite DLTS is the minimization of deterministic automata (LTS with the precision of an initial state and a set of final states). It is well known that, when there is no useless state (a state q is useless if there is no path from the initial state to q or if there is no path from q to a final state) this amounts of finding the coarsest bisimulation included in the following equivalence relation: $\mathscr{R}_{\text{init}} = \{(q, q') \in Q \times Q \,|\, q \in F \Leftrightarrow q' \in F\}$ with $F \subseteq Q$ the set of final states. The blocks of this relation are: F, if it is not empty (but in that case the minimal au-

tomata is the empty one), and $(Q \setminus F)$ if it is also not empty. Therefore, the algorithm presented in the present paper can be used to minimize a deterministic automata with the complexities announced.

Although this was not the purpose, a good piece of news is that `DBisim` is very similar to the algorithm in [9]. The main difference being that our LTS are deterministic. An extension of the present paper to the coarsest bisimulation problem over LTS will be done. This will yield an algorithm with the same time and space complexities of [10] but simpler.

References

[1] Marie-Pierre Béal and Maxime Crochemore. Minimizing incomplete automata. In *Finite-State Methods and Natural Language Processing (FSMNLP'08)*, Joint Research Center, pages 9–16, 2008.

[2] Gérard Cécé. Three Simulation Algorithms for Labelled Transition Systems.

[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] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[5] Jean-Claude Fernandez. An implementation of an efficient algorithm for bisimulation equivalence. *Sci. Comput. Program.*, 13(2-3):219–236, 1990.

[6] Orna Grumberg and David E. Long. Model checking and modular verification. *ACM Trans. Program. Lang. Syst.*, 16(3):843–871, 1994.

[7] John E. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. Technical Report CS-TR-71-190, Stanford University, Department of Computer Science, January 1971.

[8] Timo Knuutila. Re-describing an algorithm by hopcroft. *Theor. Comput. Sci.*, 250(1-2):333–363, 2001.

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

[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] Antti Valmari and Petri Lehtinen. Efficient minimization of dfas with partial transition. In Susanne Albers and Pascal Weil, editors, *STACS*, volume 1 of *LIPIcs*, pages 645–656. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2008.