Saving time in a space-efficient simulation algorithm

Systems Engineering Group
Department of Mechanical Engineering
Eindhoven University of Technology
PO Box 513
5600 MB Eindhoven
The Netherlands
http://se.wtb.tue.nl/

SE Report: Nr. 2011-03

Saving Time in a Space-Efficient Simulation Algorithm

J. Markovski

ISSN: 1872-1567

SE Report: Nr. 2011-03
Eindhoven, April 2011
SE Reports are available via http://se.wtb.tue.nl/sereports

Abstract

We present an efficient algorithm for computing the simulation preorder and equivalence for labeled transition systems. The algorithm improves an existing space-efficient algorithm and improves its time complexity by employing a variant of the stability condition and exploiting properties of the underlying relations and partitions. It has comparable space and time complexity with the most efficient counterpart algorithms for Kripke structures.

1 Introduction

The importance of the simulation preorder and equivalence in compositional verification has been stated on more than one occasion [1, 2, 3, 4, 5, 6]. It has been shown a natural preorder for matching implementations and specifications, when the preservation of the branching structure is important [7]. Moreover, it preserves existential and universal fragments of CTL* and standard modal μ-calculus [8]. Since the main application of minimization by simulation is to battle state-space explosion in verification, most of the algorithms are developed for Kripke structures. Notably, it is considered that they can be easily adjusted for labeled transition systems [9, 10]. The effect of such translations is typically neglected, even though efficient translations preserving predefined behavior may not be obvious [11].

Suppose that the underlying system to be minimized, be it a Kripke structure or a labeled transition system, has a set of states $\mathcal{S}$, a transition relation $\rightarrow$, a set of action labels $\mathcal{A}$, and simulation classes contained in partition $\mathcal{P}$. Then, the most computationally-efficient algorithm for computing the simulation preorder of Kripke structures has time complexity of $\mathcal{O}(|\mathcal{P}||\rightarrow)$ [4]. Unfortunately, this algorithm suffered from quadratic space complexity in the number of simulation classes, which was improved upon in [12, 6] to $\mathcal{O}(|\mathcal{P}||\mathcal{S}|\log(|\mathcal{S}|))$. The space complexity of $\mathcal{O}(|\mathcal{S}|\log(|\mathcal{P}|)+|\mathcal{P}|^2)$ for minimizing Kripke structures by simulation equivalence is achieved in [2], an algorithm later shown flawed and mended in [5]. This complexity is considered optimal when representing the simulation preorder as a partition-relation pair by keeping similar states in same partition classes, while representing the preorder as a relation between the classes. Unfortunately, this algorithm has an inferior time complexity of $\mathcal{O}(|\mathcal{P}|^2|\rightarrow|)$ [2, 5]. The space complexity of [12], has been iteratively improved to $\mathcal{O}(|\mathcal{P}|^2\log(|\mathcal{S}|)+|\mathcal{S}|\log(|\mathcal{P}|))$ [10, 13], based on original algorithm of [4]. This improvement in space complexity led to a slight performance decrease as the time complexity increases to $\mathcal{O}(|\mathcal{P}||\rightarrow|\log(|\mathcal{P}|))$.

Our main motivation for developing a new algorithm for minimization by simulation, focused on labeled transitions systems, is ongoing research in process-theoretic approaches to automated generation of control software [14]. There, the underlying refinement relation between the implementation and specification is a so-called partial bisimulation preorder [14]. This relation lies between simulation and bisimulation, by requiring that the specification simulates all actions of the implementation, whereas in the other direction only a subset of the actions needs to be (bi)simulated. So, the stability conditions that identify when the partition-relation pair represents partial bisimulation are a combination of the stability conditions for simulation and bisimulation. Thus, in this paper, we rewrite the stability conditions for simulation to stability condition for bisimulation, which deals with the partitioning of states, and stability condition for the simulation preorder of the partition classes.

This allows us to take a different approach from others by improving the time complexity of the space-efficient algorithm of [2], instead of improving the space complexity of [4]. Unlike [2, 5], we employ splitters for our refinement operation in the vein of [15, 1]. Moreover, we employ the "process the smaller half" method, that enables efficient refinement of the partitions and we also exploit properties of the topological order induced by the preorder. As mentioned above, such an approach is a preparation for future work, where we intend to abstract uncontrolled systems for more efficient automated control software synthesis. Similar ideas regarding the use of splitters have been presented in [13], while building upon the work of [4]. The worst-case time complexity of our algorithm is $\mathcal{O}(|\mathcal{A}||\rightarrow|+|\mathcal{P}||\mathcal{S}|+|\mathcal{A}||\mathcal{P}|^3)$ for a given labeled transition system, while having a space complexity of $\mathcal{O}(|\mathcal{S}|\log(|\mathcal{P}|)+|\mathcal{A}||\mathcal{P}|^2\log(|\mathcal{P}|))$. For Kripke structures, the number of actions labels plays no role, so we consider this comparable to previous work as the upper

bounds both for $|\rightarrow|$ and $|\mathcal{A}||\mathcal{P}|^2$ amount to $|\mathcal{A}||\mathcal{S}|^2$.

The rest of this paper is organized as follows. First, we revisit the notion of simulation preorder in Section 2. Next, we introduce the notion of splitters and the refinement operator that will be used to compute the coarsest simulation preorder in Section 3. Finally, in Section 4 we discuss the algorithms for computing the refinement operator and the complexity. We finish with concluding remarks and a discussion on computing abstractions on labeled transition systems versus Kripke structures.

2 Simulation Preorder and Partition Pairs

The underlying models that we are going to consider are labeled transitions systems (with successful termination options) following the notation of [16]. A labeled transition system G is a tuple $G = (\mathcal{S}, \mathcal{A}, \downarrow, \rightarrow)$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ a set of event labels, $\downarrow \subseteq \mathcal{S}$ is a successful termination predicate, and $\rightarrow \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the labeled transition relation. For $p, q \in \mathcal{S}$ and $a \in \mathcal{A}$, we write $p \xrightarrow{a} q$ and $p\downarrow$.

Definition 2.1. A relation $R \subseteq \mathcal{S} \times \mathcal{S}$ is a simulation, if for all $p, q \in \mathcal{S}$ such that $(p, q) \in R$ it holds that:

1. if $p\downarrow$, then $q\downarrow$;
2. if $p \xrightarrow{a} p'$ for some $a \in \mathcal{A}$, then there exists $q' \in \mathcal{T}$ such that $q \xrightarrow{a} q'$ and $(p', q') \in R$;

If $(p, q) \in R$, we say that q simulates p and we write $p \preceq Bq$. If $q \preceq p$ holds as well, we write $p \leftrightarrow q$.

Note that $\preceq$ is a preorder relation that is also a simulation relation, making $\leftrightarrow$ an equivalence relation [14].

To compute the simulation preorder, we also need to compute the simulation equivalence and vice versa. We compute the simulation quotient using a partitioning algorithm for the states of the labeled transition system. To this end, we need to define a so-called little and big brother states. Let $p \xrightarrow{a} p'$ and $p \xrightarrow{a} p''$ with $p' \preceq p''$. Then we say that p' is the little brother of p'', or p'' is the big brother of p'. The big brothers play an important role in defining the quotient of a labeled transition graph as they are the only ones that we need to keep [2, 5, 1]. In the sequel, we represent the partial bisimilarity preorder by means of partition-relation pairs [2, 5]. The partition identifies similar states, whereas the relation identifies the little brother classes.

Let $G = (\mathcal{S}, \mathcal{L}, \downarrow, \rightarrow)$ and let $\mathcal{P} \subset 2^{\mathcal{S}}$. The set $\mathcal{P}$ is a *partition* over $\mathcal{S}$ if $\bigcup_{P \in \mathcal{P}} P = \mathcal{S}$ and for all $P, Q \in \mathcal{P}$, if $P \cap Q \neq \emptyset$, then $P = Q$. A *partition pair* over G is a pair $(\mathcal{P}, \sqsubseteq)$ where $\mathcal{P}$ is a partition over $\mathcal{S}$ and the (little brother) relation $\sqsubseteq \subseteq \mathcal{P} \times \mathcal{P}$ is a partial order, i.e., a reflexive, antisymmetric, transitive relation. We denote the set of partition pairs by $\mathbb{PP}$. The refinement operator, always produces partition pairs, with the little brother relation being a partial order, provided that the initial partition pair is a partial order [2, 5].

For all $P \in \mathcal{P}$, we have that $P\downarrow$ and $P\not\downarrow$, if for all $p \in P$ it holds $p\downarrow$ and $p\not\downarrow$, respectively. For $P' \in \mathcal{P}$ by $p \xrightarrow{a} P'$ we denote that there exists $p' \in P'$ such that $p \xrightarrow{a} p'$. We distinguish two types of (Galois) transitions between the partition classes [2]: $P \xrightarrow{a}_{\exists} P'$, if there exists

$p \in P$ such that $p \xrightarrow{a} P'$, and $P \xrightarrow{a}_\forall P'$, if for every $p \in P$, it holds that $p \xrightarrow{a} P'$. It is straightforward that $P \xrightarrow{a}_\forall P'$ implies $P \xrightarrow{a}_\exists P'$. Also, if $P \xrightarrow{a}_\forall P'$, then $Q \xrightarrow{a}_\forall P'$ for every $Q \subseteq P$. We define the stability conditions for the simulation preorder.

Definition 2.2. Let $G = (\mathcal{S}, \mathcal{L}, \downarrow, \rightarrow)$ be a labeled graph. We say that $(\mathcal{P}, \sqsubseteq) \in \mathbb{PP}$ over G is stable (with respect to $\downarrow$ and $\rightarrow$), if the following conditions are fulfilled:

a. For all $P \in \mathcal{P}$, it holds that $P\downarrow$ or $P\not\downarrow$.

b. For all $P, Q \in \mathcal{P}$, if $P \sqsubseteq Q$ and $P\downarrow$, then $Q\downarrow$.

c. For every $P, Q, P' \in \mathcal{P}$ and $a \in \mathcal{A}$, if $P \sqsubseteq Q$ and $P \xrightarrow{a}_\exists P'$, there exists $Q' \in \mathcal{P}$ with $P' \sqsubseteq Q'$ and $Q \xrightarrow{a}_\forall Q'$.

Given a relation $R \in S \times T$ on some sets S and T, define $R^{-1} \in T \times S$ as $R^{-1} = \{(t, s) \mid (s, t) \in R\}$. If R is a preorder, then $R \cap R^{-1}$ is an equivalence relation. The following theorem shows that every simulation preorder induces a stable partition pair [2, 9].

Theorem 2.3. *Let $G = (\mathcal{S}, \mathcal{A}, \downarrow, \rightarrow)$ and let R be a simulation preorder over $\mathcal{S}$. Let $\leftrightarrow \triangleq R \cap R^{-1}$. If $\mathcal{P} = \mathcal{S}/\leftrightarrow$ and for all $p, q \in \mathcal{S}$ it holds if $(p, q) \in R$, then $[p]_\leftrightarrow \sqsubseteq [q]_\leftrightarrow$, then $(\mathcal{P}, \sqsubseteq) \in \mathbb{PP}$ is stable.*

Vice versa, every stable partition pair induce a simulation preorder [2, 9].

Theorem 2.4. *Let $G = (\mathcal{S}, \mathcal{A}, \downarrow, \rightarrow)$ and $(\mathcal{P}, \sqsubseteq) \in \mathbb{PP}$. Define $R = \{(p, q) \in P \times Q \mid P \sqsubseteq Q\}$. If $(\mathcal{P}, \sqsubseteq)$ is stable, then R is a simulation preorder.*

Next, we define $\lhd \in \mathbb{PP} \times \mathbb{PP}$ that identifies when one partition pair is finer than the other with respect to inclusion.

Definition 2.5. Let $(\mathcal{P}, \sqsubseteq)$ and $(\mathcal{P}', \sqsubseteq')$ be partition pairs. We say that $(\mathcal{P}, \sqsubseteq)$ is finer than $(\mathcal{P}', \sqsubseteq')$, notation $(\mathcal{P}, \sqsubseteq) \lhd (\mathcal{P}', \sqsubseteq')$, if and only if for all P, Q such that $P \sqsubseteq Q$ there exist $P', Q' \in \mathcal{P}'$ such that $P \subseteq P'$, $Q \subseteq Q'$, and $P' \sqsubseteq' Q'$.

The relation $\lhd$ as given in Definition 2.5 is a partial order. The following theorem states that coarser partition pairs with respect to $\lhd$ produce coarser simulation preorders [9].

Theorem 2.6. *Let $G = (\mathcal{S}, \mathcal{A}, \downarrow, \rightarrow)$ and $(\mathcal{P}_1, \sqsubseteq_1)$, $(\mathcal{P}_2, \sqsubseteq_2) \in \mathbb{PP}$. Define $R_i = \{(p_i, q_i) \in P_i \times Q_i \mid P_i \sqsubseteq_i Q_i\}$ for $i \in \{1, 2\}$. Then $(\mathcal{P}_1, \sqsubseteq_1) \lhd \mathcal{P}_2, \sqsubseteq_2)$ if and only if $R_1 \subseteq R_2$.*

Next, for every two stable partition pairs with respect to a labeled graph, there exists a $\lhd$-coarser stable partition pair.

Theorem 2.7. *Let $G = (\mathcal{S}, \mathcal{L}, \downarrow, \rightarrow)$ and let $(\mathcal{P}_1, \sqsubseteq_1)$, $(\mathcal{P}_2, \sqsubseteq_2) \in \mathbb{PP}$ be stable. Then, there exists stable $(\mathcal{P}_3, \sqsubseteq_3) \in \mathbb{PP}$ and $(\mathcal{P}_1, \sqsubseteq_1) \lhd (\mathcal{P}_3, \sqsubseteq_3)$ and $(\mathcal{P}_2, \sqsubseteq_2) \lhd (\mathcal{P}_3, \sqsubseteq_3)$.*

Proof. Let $\mathcal{P}_3$ be the minimal partition over $\mathcal{S}$ such that for every $P_1 \in \mathcal{P}_1$ there exists $P_3 \in \mathcal{P}_3$ such that $P_1 \subseteq P_3$ and for every $P_2 \in \mathcal{P}_2$ there exists $P_3 \in \mathcal{P}_3$ such that $P_2 \subseteq P_3$. Let $P_3 \sqsubseteq_3 Q_3$ be defined for $P_3, Q_3 \in \mathcal{P}_3$ if there exist $P_1, Q_1 \in pcal_1$ such that $P_1 \subseteq P_3$, $Q_1 \subseteq Q_3$, and $P_1 \sqsubseteq_1 Q_1$ or there exist $P_2, Q_2 \in pcal_2$ such that $P_2 \subseteq P_3$, $Q_2 \subseteq Q_3$, and $P_2 \sqsubseteq_2 Q_2$. It should be clear that $(\mathcal{P}_1, \sqsubseteq_1) \lhd (\mathcal{P}_3, \sqsubseteq_3)$ and $(\mathcal{P}_2, \sqsubseteq_2) \lhd (\mathcal{P}_3, \sqsubseteq_3)$ by construction. We will show that $(\mathcal{P}_3, \sqsubseteq_3)$ is a stable partition pair with respect to $\downarrow$ and $\rightarrow$.

First we will show that for every $P_3 \in \mathcal{P}_3$, either $P_3\!\downarrow$ or $P_3 \not\downarrow$ holds. Suppose that there exist $p, q \in P_3$ such that $p \not\downarrow$ and $q\!\downarrow$. Obviously, they have to come from two different classes, say $p \in P_1$ and $q \in Q_1$ for $P_1, Q_1 \in \mathcal{P}_1$. Then, for every $Q_3 \in \mathcal{P}_3$ it holds that $Q_3 \not\downarrow$ or $Q_3\!\downarrow$. This implies that there exists a class $P_2 \in \mathcal{P}$ such that $P_2 \subseteq P_3$ and for some classes $P_1, Q_1 \in \mathcal{P}_1$ such that $P_1 \not\downarrow$ and $Q_1\!\downarrow$ we have that $P_1 \cap P_2 \neq \emptyset$ and $Q_1 \cap P_2 \neq \emptyset$, which leads to contradiction.

Next, we show that if for all $P_3, Q_3 \in \mathcal{P}_3$ such that $P_3 \sqsubseteq_3 Q_3$ if $P_3\!\downarrow$ then $Q_3\!\downarrow$. Without loss of generality we can assume that $P_3 \sqsubseteq_3 Q_3$ is because there exist $P_1, Q_1 \in \mathcal{P}_1$ such that $P_1 \subseteq P_3$, $Q_1 \subseteq Q_3$, and $P_1 \sqsubseteq_1 Q_1$. Now, suppose that $P_3\!\downarrow$, but $Q_3 \not\downarrow$. But then $P_1\!\downarrow$ and $Q_1 \not\downarrow$, which leads to contradiction.

Finally, we show that for all $P_3, Q_3, P_3' \in \mathcal{P}_3$ and $a \in \mathcal{A}$ such that $P_3 \sqsubseteq_3 Q_3$ if $P_3 \xrightarrow{a}_\exists Q_3$ then there exists a $Q_3' \in \mathcal{P}_3$ such that $Q_3 \xrightarrow{a}_\forall Q_3'$ and $P_3' \sqsubseteq_3 Q_3'$. Without loss of generality we can assume that $P_3 \sqsubseteq_3 Q_3$ is because there exist $P_1, Q_1 \in \mathcal{P}_1$ such that $P_1 \subseteq P_3$, $Q_1 \subseteq Q_3$, and $P_1 \sqsubseteq_1 Q_1$. From $P_3 \xrightarrow{a}_\exists P_3'$ we can conclude that there exists $R_1 \subseteq P_3$ such that $R_1 \xrightarrow{a}_\exists R_1'$ with $R_1' \subseteq P_3'$. Then, by applying the stability condition on $R_1 \sqsubseteq_1 R_1$, we have that there exists R_1' such that $R_1 \xrightarrow{a}_\forall R_1'$ and $P_1' \sqsubseteq_1 R_1'$. We can repeat the same thought process for every $P_2 \in \mathcal{P}_2$ such that $P_2 \subseteq P_3$ and $P_2 \cap R_1 \neq \emptyset$, and then again for the classes of $\mathcal{P}_1$ that have common elements with the above classes and so on until we exhaust all elements of P_3. This process leads to existence of $P_3'' \in \mathcal{P}_3$ such that $P_3 \xrightarrow{a}_\forall P_3''$ and moreover $P_3' \sqsubseteq_3 P_3''$ because of $R_1' \sqsubseteq_1 R_1''$. Now, suppose that $P_1 \xrightarrow{a}_\forall P_1''$ for some $P_1'' \subseteq P_3''$. For $P_1 \sqsubseteq_1 Q_1$ to be satisfied and repeating the reasoning from above for Q_3, we conclude that there exists $Q_3' \in \mathcal{P}_3$ such that $Q_3 \xrightarrow{a}_\forall Q_3'$ with $Q_1 \xrightarrow{a}_\forall Q_1'$ and $P_1'' \sqsubseteq_1 Q_1'$. This implies that $P_3'' \sqsubseteq_3 Q_3'$, finally leading to $P_3' \sqsubseteq_3 Q_3'$, which completes the proof. □

Theorem 2.7 implies that stable partition pairs form an upper lattice with respect to $\lhd$. Now, it is not difficult to observe that finding the $\lhd$-maximal stable partition pair over a labeled graph G coincides with the problem of finding the coarsest simulation preorder over G [2, 9].

Theorem 2.8. *Let $G = (\mathcal{S}, \mathcal{A}, \downarrow, \rightarrow)$. The $\lhd$-maximal stable $(\mathcal{P}, \sqsubseteq) \in \mathbb{PP}$ is induced by the simulation preorder $\preceq$, i.e., $\mathcal{P} = \mathcal{S}/\leftrightarrow$ and $[p]_\sim \sqsubseteq [q]_\sim$ if and only if $p \preceq q$.*

Theorem 2.8 supported by Theorem 2.7 induces an algorithm for computing the coarsest simulation preorder and equivalence over a labeled transition system $G = (\mathcal{S}, \mathcal{A}, \downarrow, \rightarrow)$ by computing the $\lhd$-maximal stable partition pair $(\mathcal{P}, \sqsubseteq)$ such that $(\mathcal{P}, \sqsubseteq) \lhd (\{\mathcal{S}\}, \{(\mathcal{S}, \mathcal{S})\})$. We develop an iterative refinement algorithm to compute the $\lhd$-maximal stable partition pair.

3 Refinement Operator

We refine the partitions by splitting the classes in the vein of [2, 2], i.e., we choose subsets of nodes that do not adhere to the stability conditions, referred to as *splitters*, in combination with the other nodes from the same class and, consequently, we place them in a separate class. To this end, we define *parent partitions* and splitters.

Definition 3.1. Let $(\mathcal{P}, \sqsubseteq) \in \mathbb{PP}$ be defined over $\mathcal{S}$. Partition $\mathcal{P}'$ is a parent partition of $\mathcal{P}$, if for every $P \in \mathcal{P}$, there exist $P' \in \mathcal{P}'$ with $P \subseteq P'$. The relation $\sqsubseteq$ induces a little

brother relation $\sqsubseteq'$ on $\mathcal{P}'$, defined by $P' \sqsubseteq' Q'$ for $P', Q' \in \mathcal{P}'$, if there exist $P, Q \in \mathcal{P}$ such that $P \subseteq P'$, $Q \subseteq Q'$, and $P \sqsubseteq Q$.

Let $S' \subseteq P'$ for some $P' \in \mathcal{P}'$ and put $T' = P' \setminus S'$. The set S' is a splitter of $\mathcal{P}'$ with respect to $\mathcal{P}$, if for every $P \subset P'$ either $P \subseteq S'$ or $P \cap S' = \emptyset$, where $S' \sqsubseteq' T'$ or S' and T' are unrelated. The splitter partition is $\mathcal{P}' \setminus \{P'\} \cup \{S', T'\}$.

A consequence of Definition 3.1 is that $(\mathcal{P}, \sqsubseteq) \lhd (\mathcal{P}', \sqsubseteq')$. Note that $\mathcal{P}'$ contains a splitter if and only if $\mathcal{P}' \neq \mathcal{P}$.

For implementation of the refinement operator we need the notion of a topological sorting. Topological sorting with respect to a preorder relation is a linear ordering of elements such that topologically "smaller" elements are not preorder-wise greater with respect to each other.

Definition 3.2. Let $(\mathcal{P}, \sqsubseteq) \in \mathbb{PP}$. We say that $\leq$ is a topological sorting over $\mathcal{P}$ induced by $\sqsubseteq$, if for all $P, Q \in \mathcal{P}$ it holds that $P \leq Q$ if and only if $Q \not\sqsubseteq P$.

Definition 3.2 implies that if $P \leq Q$, then either $P \sqsubseteq Q$ or P and Q are unrelated. In general, topological sorting are not uniquely defined. It can be represented as a list $\leq \triangleq [P_1, P_2, \ldots, P_n]$, for some $n \in \mathbb{N}$, where $\mathcal{P} = \{P_i \mid i \in \{1, \ldots, n\}\}$ and $P_i \leq P_j$ for $1 \leq i \leq j \leq n$.

The following property that provides for an efficient updating of the topological order.

Theorem 3.3. *Let $(\mathcal{P}_1, \sqsubseteq_1) \in \mathbb{PP}$ with $\mathcal{P}_1 = \{P_1, \ldots, P_n\}$ and let $\leq_1 = [P_1, P_2, \ldots, P_n]$ be a topological order over $\mathcal{P}_1$ induced by $\sqsubseteq_1$. Suppose that $P_k \in \mathcal{P}_1$ for some $1 \leq k \leq n$ is split to Q_1 and Q_2 such that $P_k = Q_1 \cup Q_2$ and $Q_1 \cap Q_2 = \emptyset$ resulting in $\mathcal{P}_2 = \mathcal{P}_1 \setminus \{P_k\} \cup \{Q_1, Q_2\}$ such that $(\mathcal{P}_2, \sqsubseteq_2) \lhd (\mathcal{P}_1, \sqsubseteq_1)$. Suppose that either $Q_1 \sqsubseteq_2 Q_2$ or Q_1 and Q_2 are unrelated. Then, $\leq_2 = [P_1, \ldots, P_{k-1}, Q_1, Q_2, P_{k+1}, \ldots, P_n]$ is a topological sorting over $\mathcal{P}_2$ induced by $\sqsubseteq_2$.*

Proof. To show this, recall that from $(\mathcal{P}_2, \sqsubseteq_2) \lhd (\mathcal{P}_1, \sqsubseteq_1)$, we have for all $P'', Q'' \in \mathcal{P}_2$ such that $P'' \sqsubseteq_2 Q''$, there exist $P', Q' \in \mathcal{P}_1$ such that $P'' \subseteq P'$, $Q'' \subseteq Q'$, and $P' \sqsubseteq_1 Q'$. So, if $P' \leq_1 Q'$ then $P'' \leq_2 Q''$ or P'' and Q'' are unrelated for all $P'', Q'' \in \mathcal{P}_2$ such that $P'' \neq Q_1$ and $Q'' \neq Q_2$. Since $Q_1 \sqsubseteq_2 Q_2$ or Q_1 and Q_2 are unrelated, we have that $\leq_2$ is a topological sorting. □

Theorem 3.3 enables us to update the topological sorting by locally replacing each class with the results of the splitting without having to re-compute the whole sorting in every iteration, as it is done in [2, 5]. As a result, the classes whose nodes belong to the same parent are neighboring with respect to the topological sorting. Moreover, it also provides us with a procedure for searching for a little or a big brother of a given class. All little brothers of a given class are topologically sorted in descendent to the left, and all the big brothers are topologically sorted ascendent to the right.

Now, we can define a refinement fix-point operator Rfn. It takes as input $(\mathcal{P}_i, \sqsubseteq_i) \in \mathbb{PP}$ and an induced parent partition pair $(\mathcal{P}'_i, \sqsubseteq'_i)$, with $(\mathcal{P}_i, \sqsubseteq_i) \lhd (\mathcal{P}'_i, \sqsubseteq'_i)$, for some $i \in \mathbb{N}$, which are stable with respect to each other. Its result are $(\mathcal{P}_{i+1}, \sqsubseteq_{i+1}) \in \mathbb{PP}$ and parent partition $\mathcal{P}'_{i+1}$ such that $(\mathcal{P}_{i+1}, \sqsubseteq_{i+1}) \lhd (\mathcal{P}_i, \sqsubseteq_i)$ and $(\mathcal{P}'_{i+1}, \sqsubseteq'_{i+1}) \lhd (\mathcal{P}'_i, \sqsubseteq'_i)$. Note that $\mathcal{P}'_i$ and $\mathcal{P}'_{i+1}$ differ only in one class, which is induced by the splitter that we employed to refine $\mathcal{P}_i$ to $\mathcal{P}_{i+1}$. This splitter comprises classes of $\mathcal{P}_i$, which are strict subsets from some class of $\mathcal{P}'_i$. The refinement stops, when a fix point is reached for $m \in \mathbb{N}$ with $\mathcal{P}_m = \mathcal{P}'_m$. In the following, we omit partition pair indices, when clear from the context.

Now, suppose that $(\mathcal{P}, \sqsubseteq) \in \mathbb{PP}$ has $\mathcal{P}'$ as parent with $(\mathcal{P}, \sqsubseteq) \lhd (\mathcal{P}', \sqsubseteq')$, where $\sqsubseteq'$ is induced by $\sqsubseteq$. Condition a of Definition 2.2 requires that all states in a class have or, alternatively, do not have termination options. We resolve this issue by choosing a stable initial partition pair, for $i = 0$, that fulfills this condition, i.e., for all classes $P \in \mathcal{P}_0$ it holds that either $P\!\downarrow$ or $P \not\downarrow$. For condition b, we specify $\sqsubseteq_0$ such that $P \sqsubseteq_0 Q$ with $P\!\downarrow$ holds, only if $Q\!\downarrow$ holds as well. Thus, following the initial refinement, we only need to ensure that the stability condition c is satisfied, as shown in Theorem 3.7 below. For convenience, we rewrite this stability condition for $(\mathcal{P}, \sqsubseteq)$ with respect to $(\mathcal{P}', \sqsubseteq')$.

Definition 3.4. Let $(\mathcal{P}, \sqsubseteq) \in \mathbb{PP}$ and let $(\mathcal{P}', \sqsubseteq')$ be its parent partition pair, where for all $P' \in \mathcal{P}'$ either $P' \not\downarrow$ or $(\downarrow P')$. Then, $(\mathcal{P}, \sqsubseteq)$ is stable with respect to $\mathcal{P}'$ if:

1. For all $P \in \mathcal{P}$, $a \in \mathcal{A}$, and $P' \in \mathcal{P}'$, if $P \xrightarrow{a}_{\exists} P'$, there exists $Q' \in \mathcal{P}'$ with $P' \sqsubseteq' Q'$ and $P \xrightarrow{a}_{\forall} Q'$.
2. For all $P, Q \in \mathcal{P}$, $a \in \mathcal{A}$, $P' \in \mathcal{P}'$, if $P \sqsubseteq Q$ and $P \xrightarrow{a}_{\forall} P'$, there exists $Q' \in \mathcal{P}'$ with $P' \sqsubseteq' Q'$ and $Q \xrightarrow{a}_{\forall} Q'$.

It is not difficult to observe that stability conditions 1 and 2 replace stability condition c of Definition 2.2. They are equivalent when $\mathcal{P} = \mathcal{P}'$, which is the goal of our fix point refinement operation. From now on, we refer to the stability conditions above instead of the ones in Definition 2.2. The form of the stability conditions is useful as condition 1 is employed to refine the splitters, whereas condition 2 is used to adjust the little brother relation. Moreover, if the conditions of Definition 3.4 are not fulfilled for $(\mathcal{P}, \sqsubseteq) \lhd (\mathcal{P}', \sqsubseteq')$, then the partition pair $(\mathcal{P}, \sqsubseteq)$ is not stable.

Theorem 3.5. *Let $(\mathcal{P}, \sqsubseteq) \in \mathbb{PP}$, let $\mathcal{P}'$ be a parent partition, and suppose that the conditions of Definition 3.4 do not hold. Then $(\mathcal{P}, \sqsubseteq)$ is not stable.*

Proof. For condition 1 of Definition 3.4 suppose that $P \in \mathcal{P}$ and $P' \in \mathcal{P}'$ are such that $P \xrightarrow{a}_{\exists} Q'$. Then there exists a class $Q \in \mathcal{P}$ such that $Q \subseteq Q'$ and $P \xrightarrow{a}_{\exists} Q$. Now suppose that there does not exist $R' \in \mathcal{P}'$ such that $Q' \xrightarrow{a}_{\forall} R'$ and $Q' \sqsubseteq' R'$, but there exists $R \in \mathcal{P}$ such that $P \xrightarrow{a}_{\forall} R$ and $Q \sqsubseteq R$. Suppose that $R \subseteq R'$ for some $R' \in \mathcal{P}'$. Then $P \xrightarrow{a}_{\forall} R'$ as well, whereas $Q \sqsubseteq R$ is conditioned by $Q' \sqsubseteq' R'$, which leads to a contradiction.

For condition 2 suppose that $P \sqsubseteq Q$ and $P \xrightarrow{a}_{\forall} P'$ for some $P, Q \in \mathcal{P}$ and $P' \in \mathcal{P}'$, but there does not exist $Q' \in \mathcal{P}'$ such that $Q \xrightarrow{a}_{\forall} Q'$ and $P' \sqsubseteq' Q'$. Then there exists a class $S \in \mathcal{P}$ such that $S \subseteq P'$ and $Q \xrightarrow{a}_{\exists} S$, implying that there exists a class $R \in \mathcal{P}$ such that $Q \xrightarrow{a}_{\forall} R$ and $S \sqsubseteq R$. Suppose that $R \subseteq R'$ for some $R' \in \mathcal{P}'$. Then $P \xrightarrow{a}_{\forall} R'$ as well, whereas $S \sqsubseteq R$ is conditioned by $P' \sqsubseteq' R'$, which leads to a contradiction. □

We note that the stability condition 1 of Definition 3.4 is actually the stability condition for bisimulation [15, 1], whereas stability condition 2 only employs the $\rightarrow_{\forall}$ transition relation. This is slightly different from the equivalent stability condition employed in [6, 10, 13]: For every $P \in \mathcal{P}$ and $a \in \mathcal{A}$, if $P \xrightarrow{a}_{\exists} P'$, then $P \xrightarrow{a}_{\forall} \bigcup_{Q \sqsubseteq Q'} Q'$. This stability condition directly incorporates stability condition 2 and it enables refinements with respect to splitters made up of the union of the big brothers [6].

The initial stable partition pair and parent partition are induced by the termination options and outgoing transitions of the comprising states. To this end, we define *the set of outgoing labels* of a state $p \in \mathcal{S}$ to be $\mathrm{OL}(p) \triangleq \{a \in \mathcal{A} \mid p \xrightarrow{a}\}$. Let $P \subseteq \mathcal{S}$. If for all $p, q \in P$ we have that $\mathrm{OL}(p) = \mathrm{OL}(q)$ we define $\mathrm{OL}(P) = \mathrm{OL}(p)$ for any $p \in P$.

Definition 3.6. Let $G = (\mathcal{S}, \mathcal{A}, \downarrow, \rightarrow)$, let $P'_{\not\downarrow} = \{p \in \mathcal{S} \mid p \not\downarrow\}$, and $P'_{\downarrow} = \mathcal{S} \setminus P'_{\not\downarrow}$. The initial parent partition is given by $\{P'_{\not\downarrow}, P'_{\downarrow}\}$, where $P'_{\not\downarrow}$ or $P'_{\downarrow}$ are omitted if empty.

The initial stable partition pair $(\mathcal{P}_0, \sqsubseteq_0)$ is defined as the coarsest stable partition pair, where for every $P \in \mathcal{P}_0$, either $P \not\downarrow$ or $P\downarrow$ holds, $\mathrm{OL}(P)$ is well-defined, and for every $P, Q \in \mathcal{P}_0$, $P \sqsubseteq_0 Q$ holds if and only if $\mathrm{OL}(P) \subseteq \mathrm{OL}(Q)$ and if $P\downarrow$, then $Q\downarrow$ as well.

For every stable $(\mathcal{P}, \sqsubseteq) \in \mathbb{PP}$, we have $(\mathcal{P}, \sqsubseteq) \lhd (\mathcal{P}_0, \sqsubseteq_0)$. In the opposite, some stability condition of Definition 2.2 fails.

Theorem 3.7. *Let $(\mathcal{P}, \sqsubseteq) \in \mathbb{PP}$, and let $(\mathcal{P}_0, \sqsubseteq_0)$ be given as in Definition 3.6. If $(\mathcal{P}, \sqsubseteq)$ is stable, then $(\mathcal{P}, \sqsubseteq) \lhd (\mathcal{P}_0, \sqsubseteq_0)$.*

Proof. For all $P \in \mathcal{P}$ it holds that either $P\downarrow$ or $P \not\downarrow$, and if $P\downarrow$ and $P \sqsubseteq Q$ for some $Q \in \mathcal{P}$, then $Q\downarrow$ as well, which is also respected by $(\mathcal{P}_0, \sqsubseteq_0)$. Now, suppose that $P \sqsubseteq Q$ and $P \xrightarrow{a}_\exists P'$ for some $P, P', Q \in \mathcal{P}$. Then, there exists $Q' \in \mathcal{P}$ such that $P' \sqsubseteq Q'$ and $Q \xrightarrow{a}_\forall Q'$. When we substitute $Q = P$, we have that there must always exist some $P'' \in \mathcal{P}$ such that $P \xrightarrow{a}_\forall P''$ for all $a \in \mathcal{A}$, implying that $\mathrm{OL}P$ is well-defined. Moreover, we have that $\mathrm{OL}(P) \subseteq \mathrm{OL}(Q)$. As $(\mathcal{P}_0, \sqsubseteq_0)$ is the coarsest partition pair satisfying these conditions, we have that $(\mathcal{P}, \sqsubseteq) \lhd (\mathcal{P}_0, \sqsubseteq_0)$. □

The fix-point refine operator Rfn will be applied iteratively to the initial stable partition pair $(\mathcal{P}_0, \sqsubseteq_0)$ and $\mathcal{P}'_0$.

Definition 3.8. Let $(\mathcal{P}, \sqsubseteq) \in \mathbb{PP}$ and let $\mathcal{P}'$ be a parent partition of $\mathcal{P}$ with $\mathcal{P} \neq \mathcal{P}'$. Let $\leq$ be a topological sorting over $P \in \mathcal{P}$ induced by $\sqsubseteq$. Let $S' \subset P'$ for some $P' \in \mathcal{P}'$ be a splitter for $\mathcal{P}'$ with respect to $\mathcal{P}$. Suppose that $P' = \bigcup_{i=1}^{k} P_i$ for some $P_i \in \mathcal{P}$ for $k > 1$ with $P_1 \leq \ldots \leq P_k$ and $S' = P_1 \cup \ldots \cup P_s$ for $1 \leq s < k$.

Define $\mathrm{Rfn}(\mathcal{P}, \sqsubseteq, \mathcal{P}', S') = (\mathcal{P}_r, \sqsubseteq_r)$, where $(\mathcal{P}_r, \sqsubseteq_r)$ is the coarsest partition pair $(\mathcal{P}_r, \sqsubseteq_r) \lhd (\mathcal{P}, \sqsubseteq)$ that is stable with respect to $\mathcal{P}' \setminus \{P'\} \cup \{S', P' \setminus S'\}$.

The existence of the coarsest partition pair $(\mathcal{P}_r, \sqsubseteq_r)$ is guaranteed by Theorem 2.7. Once a stable partition pair is reached, it is no longer refined.

Theorem 3.9. *Let $G = (\mathcal{S}, \mathcal{A}, \downarrow, \rightarrow)$ and let $(\mathcal{P}, \sqsubseteq) \in \mathbb{PP}$ over $\mathcal{S}$ be stable. For every parent partition $\mathcal{P}'$ such that $\mathcal{P}' \neq \mathcal{P}$ and every splitter S' of $\mathcal{P}'$ with respect to $\mathcal{P}$, it holds that $\mathrm{Rfn}(\mathcal{P}, \sqsubseteq, \mathcal{P}', S') = (\mathcal{P}, \sqsubseteq)$.*

Proof. We will show that $(\mathcal{P}, \sqsubseteq)$ is stable with respect to every parent partition $\mathcal{P}'$, implying that the refinement operator with not change $(\mathcal{P}, \sqsubseteq)$. Suppose that for some $P \in \mathcal{P}$ and $P' \in \mathcal{P}'$ it holds that $P \xrightarrow{a}_\exists P'$. Then, there exists $Q \in \mathcal{P}$ such that $Q \subseteq P'$ and $P \xrightarrow{a}_\exists Q$, implying that there exists $R \in \mathcal{P}$ such that $Q \sqsubseteq R$ and $P \xrightarrow{a}_\forall R$. Suppose that $R \subseteq R'$ for some $R' \in \mathcal{P}'$. Then $P' \sqsubseteq' R'$ and $P \xrightarrow{a}_\forall R'$.

Now, suppose that for some $P, Q \in \mathcal{P}$ and $P' \in pcal'$ such that $P \sqsubseteq Q$ it holds that $P \xrightarrow{a}_\forall P'$. Then, there exists some $R \subseteq P'$ such that $P \xrightarrow{a}_\exists R$, implying that there exists some $S \in \mathcal{P}$ such that $P \xrightarrow{a}_\forall S$ and $R \sqsubseteq S$. Suppose that $S \subseteq S'$ for some $S' \in \mathcal{P}'$. Since $P \sqsubseteq Q$ and $P \xrightarrow{a}_\forall S$, there exists $T \in \mathcal{P}$, such that $Q \xrightarrow{a}_\forall T$ and $S \sqsubseteq T$. Suppose that $T \subseteq T'$. Then, we have that $P' \sqsubseteq' S' \sqsubseteq' T'$ and $P \xrightarrow{a}_\forall T'$, which completes the proof. □

When refining two partition pairs $(\mathcal{P}_1, \sqsubseteq_1) \lhd (\mathcal{P}_2, \sqsubseteq_2)$ with respect to the same parent partition and splitter, the resulting partition pairs are also related by $\lhd$.

Theorem 3.10. *Let $(\mathcal{P}_1, \sqsubseteq_1), (\mathcal{P}_2, \sqsubseteq_2) \in \mathbb{PP}$ and $(\mathcal{P}_1, \sqsubseteq_1) \lhd (\mathcal{P}_2, \sqsubseteq_2)$. Let $\mathcal{P}'$ be a parent partition of $\mathcal{P}_2$ and let S' be a splitter of $\mathcal{P}'$ with respect to $\mathcal{P}_2$. Then* $\mathrm{Rfn}(\mathcal{P}_1, \sqsubseteq_1, \mathcal{P}', S') \lhd \mathrm{Rfn}(\mathcal{P}_2, \sqsubseteq_2, \mathcal{P}', S')$.

Proof. Since $\mathcal{P}'$ is a parent partition of $\mathcal{P}_2$, it is also a parent partition of $\mathcal{P}_1$ making $\mathrm{Rfn}(\mathcal{P}_1, \sqsubseteq_1, \mathcal{P}', S')$ and $\mathrm{Rfn}(\mathcal{P}_2, \sqsubseteq_2, \mathcal{P}', S')$ well defined. Suppose that $\mathrm{Rfn}(\mathcal{P}_1, \sqsubseteq_1, \mathcal{P}', S') = (\mathcal{P}_3, \sqsubseteq_3)$ and $\mathrm{Rfn}(\mathcal{P}_2, \sqsubseteq_2, \mathcal{P}', S') = (\mathcal{P}_4, \sqsubseteq_4)$. As $(\mathcal{P}_3, \sqsubseteq_3) \lhd (\mathcal{P}_1, \sqsubseteq_1)$, then it also holds that $(\mathcal{P}_3, \sqsubseteq_3) \lhd (\mathcal{P}_2, \sqsubseteq_2)$. Then, according to definition 3.8, $(\mathcal{P}_3, \sqsubseteq_3)$ is stable with respect to $(\mathcal{P}_2, \sqsubseteq_2)$ as well. As $(\mathcal{P}_4, \sqsubseteq_4)$ is the coarsest partition that is stable with respect to $(\mathcal{P}_2, \sqsubseteq_2)$, this implies that $(\mathcal{P}_3, \sqsubseteq_3) \lhd (\mathcal{P}_4, \sqsubseteq_4)$. □

The refinement operator ultimately produces the coarsest stable partition pair with respect to a labeled graph.

Theorem 3.11. *Let $G = (\mathcal{S}, \mathcal{A}, \downarrow, \rightarrow)$, let $(\mathcal{P}_0, \sqsubseteq_0)$ be the initial stable partition pair, and $\mathcal{P}'_0$ the initial parent partition as given by Definition 3.6, and S'_0 a splitter. Suppose that $(\mathcal{P}_c, \sqsubseteq_c)$ is the coarsest stable partition pair with respect to $\downarrow$, $\rightarrow$, and $B \subseteq \mathcal{A}$. Then, there exist partitions $\mathcal{P}'_i$ and splitters S'_i for $i \in \{1, \ldots, n\}$ such that* $\mathrm{Rfn}(\mathcal{P}_i, \sqsubseteq_i, \mathcal{P}'_i, S'_i)$ *are well defined with $\mathcal{P}_n = \mathcal{P}'_n$ and $(\mathcal{P}_n, \sqsubseteq_n) = (\mathcal{P}_c, \sqsubseteq_c)$.*

Proof. We put $\mathcal{P}'_{i+1} = \mathcal{P}'_i \setminus \{P'_i\} \cup \{S'_i, T'_i\}$, where $S'_i \subset P'_i$ and $T'_i = P'_i \setminus S'_i$ for some $P'_i \in \mathcal{P}'_i$ for $i \in \{0, \ldots, n\}$. It should be clear that n is a finite number. By Theorem 3.9 we have that $\mathrm{Rfn}(\mathcal{P}_c, \sqsubseteq_c, \mathcal{P}'_i, S'_i) = (\mathcal{P}_c, \sqsubseteq_c)$ for every $i = \{0, \ldots, n\}$. Since $(\mathcal{P}_c, \sqsubseteq_c) \lhd (\mathcal{P}_0, \sqsubseteq_0)$ by Theorem 3.7 we obtain that $(\mathcal{P}_c, \sqsubseteq_c \lhd (\mathcal{P}_n, \sqsubseteq_n)$. Since $\mathcal{P}_n = \mathcal{P}'_n$ we have that $(\mathcal{P}_n, \sqsubseteq_n)$ is stable with respect to $\downarrow$ and $\rightarrow$. By definition $(\mathcal{P}_c, \sqsubseteq_c)$ is the coarsest such partition pair, directly implying that $(\mathcal{P}_n, \sqsubseteq_n) \lhd (\mathcal{P}_c, \sqsubseteq_c)$. This implies that $(\mathcal{P}_c, \sqsubseteq_c) = (\mathcal{P}_n, \sqsubseteq_n)$ which completes the proof. □

We can summarize the high-level algorithm for computing the coarsest partition pair in Algorithm 1.

Algorithm 1: Algorithm for computing the coarsest stable partition pair for $G = (\mathcal{S}, \mathcal{A}, \downarrow, \rightarrow)$

```
1 Compute initial stable partition pair (P, ⊑) and parent partition P' over S with respect to ↓
  and →;
2 while P ≠ P' do
3     Find splitter S' for P' with respect to P;
4     (P, ⊑) := Rfn(P, ⊑, P', S');
5     P' := P' \ {P'} ∪ {S', P' \ S'};
```

The algorithm implements the refinement steps by splitting every class in $\mathcal{P}$ with respect to the splitters S' and $P' \setminus S'$ for some parent $P' \in \mathcal{P}'$ in order to satisfy the stability conditions of Definition 3.4. The minimized labeled transition system has states $P \in \mathcal{P}$ with $P \xrightarrow{a} Q$ for $a \in \mathcal{A}$, if there does not exist $R \neq Q$ such that $Q \sqsubseteq R$ and $P \xrightarrow{a}_{\forall} Q$.

Next, we discuss the algorithm for computing the fix-point refinement operator.

4 Minimization Algorithm

We give alternative representations of the sets and relations required for computation of the refinement operator in order to provide a computationally efficient algorithm. The partition is represented as a list of states that preserves the topological order induced by $\sqsubseteq$, whereas the parent partition is a list of partition classes. Given two lists L_1 and L_2, by L_1L_2, we denote their concatenation. The little brother relation $\sqsubseteq$ is given as a table, whereas for $\sqsubseteq'$, we use a counter $\mathrm{cnt}_{\sqsubseteq}(P', Q')$ that keeps the number of pairs (P, Q) for $P, Q \in \mathcal{P}$ such that $P \subseteq P'$, $Q \subseteq Q'$, $P \neq Q$, and $P \sqsubseteq Q$. When splitting P' to S' and T', $\mathrm{cnt}_{\sqsubseteq}(P', P') = \mathrm{cnt}_{\sqsubseteq}(S', S') + \mathrm{cnt}_{\sqsubseteq}(S', T') + \mathrm{cnt}_{\sqsubseteq}(T', T')$. We keep only one Galois relation $\rightarrow_{\exists\forall} = \rightarrow_{\forall} \cup \rightarrow_{\exists}$, with a counter $\mathrm{cnt}_{\forall}(P, a, P')$ for $P \in \mathcal{P}$, $P' \in \mathcal{P}'$ and $a \in \mathcal{A}$, where $\mathrm{cnt}_{\forall}(P, a, P')$ keeps the number of $Q' \in \mathcal{P}'$ with $P' \sqsubseteq' Q'$ and $P \xrightarrow{a}_{\forall} Q'$. In this way we can check the conditions of Definition 3.4 efficiently. For example, if $P \xrightarrow{a}_{\exists\forall} P'$ and $\mathrm{cnt}_{\forall}(P, a, P') = 0$, then P is not stable with respect to P', so it has to be split. Also, if $P \sqsubseteq Q$ and $\mathrm{cnt}_{\forall}(P, a, P') > 0$, but $\mathrm{cnt}_{\forall}(Q, a, P') = 0$, then $P \sqsubseteq Q$ cannot hold, and it must be erased. By $:=$ we denote assignment, and for compactness we use $Y\,op = X$ instead of $Y := Y\,op\,X$ for $op \in \{+, -, \setminus, \cup\}$. We note that a similar approach is also taken in [6] to efficiently represent the splitter as the union of the big brothers.

To efficiently split the classes in the vein of [17, 15], the algorithm keeps track of the count of labeled transitions to the parents. Then, to split a class $P \in \mathcal{P}$ with respect to a splitter $S' \subseteq P' \in \mathcal{P}'$ and the remainder $T' = P' \setminus S'$, we just need to compute this count for the smaller splitter and deduce it in one step for the other. To this end, we define a function $\mathrm{cnt}_{\rightarrow} \colon \mathcal{S} \times \mathcal{A} \times \mathcal{P}' \to \mathbb{N}$. Now, for every $p \in P$ and $a \in \mathcal{A}$, if we know $\mathrm{cnt}_{\rightarrow}(p, a, P')$ and compute $\mathrm{cnt}_{\rightarrow}(p, a, S')$, then we have that $\mathrm{cnt}_{\rightarrow}(p, a, P' \setminus S') = \mathrm{cnt}_{\rightarrow}(p, a, P') - \mathrm{cnt}_{\rightarrow}(p, a, S')$. We deduce the following:

0. If $\mathrm{cnt}_{\rightarrow}(p, a, P') = \mathrm{cnt}_{\rightarrow}(p, a, S') = 0$, then $p \xnrightarrow{a} S'$, $p \xnrightarrow{a} T'$, and $\mathrm{cnt}_{\rightarrow}(p, a, T') = 0$;
1. If $\mathrm{cnt}_{\rightarrow}(p, a, P') > 0$ and $\mathrm{cnt}_{\rightarrow}(p, a, S') = 0$, then $p \xnrightarrow{a} S'$, $p \xrightarrow{a} T'$, and $\mathrm{cnt}_{\rightarrow}(p, a, P' \setminus S') = \mathrm{cnt}_{\rightarrow}(p, a, P')$;
2. If $\mathrm{cnt}_{\rightarrow}(p, a, P') = \mathrm{cnt}_{\rightarrow}(p, a, S') > 0$, then $p \xrightarrow{a} S'$, $p \xnrightarrow{a} T'$, and $\mathrm{cnt}_{\rightarrow}(p, a, T') = 0$;
3. If $\mathrm{cnt}_{\rightarrow}(p, a, P') > 0$, $\mathrm{cnt}_{\rightarrow}(p, a, S') > 0$, and $\mathrm{cnt}_{\rightarrow}(p, a, S') \neq \mathrm{cnt}_{\rightarrow}(p, a, P')$, then $p \xrightarrow{a} S'$, $p \xrightarrow{a} T'$, and $\mathrm{cnt}_{\rightarrow}(p, a, T') = \mathrm{cnt}_{\rightarrow}(p, a, P') - \mathrm{cnt}_{\rightarrow}(p, a, S')$.

Using the updated counters, we easily deduce if $P \xnrightarrow{a}_{\exists} P'$, $P \xrightarrow{a}_{\exists} P'$, or $P \xrightarrow{a}_{\forall} P'$ for $P \in \mathcal{P}$, $P' \in \mathcal{P}'$, and $a \in \mathcal{A}$.

The initial stable partition pair is computed in three steps. We assume that the partition pair $(\mathcal{P}, \sqsubseteq)$, the parent partition $\mathcal{P}'$, the states $\mathcal{S}$, the transition relation $\rightarrow$, and the supporting counters $\mathrm{cnt}_{\rightarrow}$, $\mathrm{cnt}_{\sqsubseteq}$, and $\mathrm{cnt}_{\forall}$ are globally accessible. Local data is initialized inside the algorithms. The first step, given by Algorithm 2, groups the nodes into classes according to their outgoing labels. This algorithm is also used to compute the initial partition when performing minimization by bisimulation [15, 1]. It employs a binary tree to decide in which class to place a state by encoding that children in the left subtree do not have the associated labeled transition as outgoing, whereas the one in the right subtree do. We assume that the action labels are given by a set $\mathcal{A} = \{a_1, \ldots, a_n\}$, so the tree has height $|\mathcal{A}|$. The leaves of the tree contain the states of the corresponding classes. The algorithm makes decisions on going left or right for the first $n-1$ levels, whereas the leaves at level n contain the nodes. Traversing the binary tree in inorder fashion results in a topological sorting with respect to the outgoing labels.

Algorithm 2: `SortStatesByOL()` - Sorts states by the labels of the outgoing transitions

```
1  new(root);
2  for p ∈ S do
3      move := root;
4      for i := 1, ..., n − 1 do
5          if a_i ∈ OL(p) then
6              if move.right = null then new(move.right);
7              move := move.right;
8          else
9              if move.left = null then new(move.left);
10             move := move.left;
11     if a_n ∈ OL(p) then
12         if move.right = null then new(move.right);
13         move := move.right; move.set := move.set ∪ {p};
14     else
15         if move.left = null then new(move.left);
16         move := move.left; move.set := move.set ∪ {p};
17 Return root;
```

Once the binary tree is computed, we need compute the little brother pairs as given in Algorithm 3. Recall that the left subtree at level i for $1 \leq i \leq n$ leads to classes that do not have action a_i in their outgoing labels, whereas the right subtree leads to classes that have a_i in their outgoing labels. The initial little brother relation is based on inclusion of outgoing label sets. Thus, if we traverse the tree inorder, i.e., if we recursively first visit the left subtree, then the root, and finally the right subtree, and keep track of corresponding subtrees that comprise the same or bigger sets of outgoing labels, we can fill in the little brother pairs. The algorithm also computes the initial partition as two globally accessible partitions $\mathcal{P}_{\not\downarrow}$ and $\mathcal{P}_{\downarrow}$ that contain classes that do not and do successfully terminate, respectively. The parent classes are denoted by $P'_{\not\downarrow}$ and $P'_{\downarrow}$, respectively.

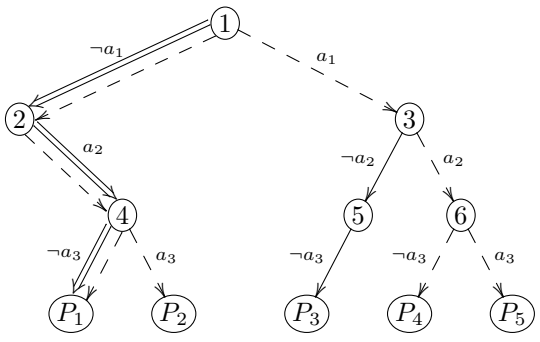

Figure 1: Computing little brother relation $\sqsubseteq$

Example 4.1. To clarify Algorithm 3, we give an instance of its execution, depicted in Fig. 1. By double lines we denote the preorder traversal to P_1, and by dashed lines the computation of the potential big brothers. The nodes are marked by number for ease of reference. At level 0, the potential big brothers start from the root $\{1\}$, and we continue the preorder traversal to the left. As the left subtree leads to classes that do not have a_1 as an outgoing label, the potential big brothers may, but do not have to contain a_1, given by $\{2, 3\}$. At node 2 there is no left subtree, so we continue with the right subtree

Algorithm 3: `InitialPP`(*move*, *BBNodes*) - Computes initial little brother relation $\sqsubseteq$

```
1  if move.left = null and move.right = null then
2      P↯ := {p ∈ move.set | p ↯}; P↓ := move.set \ P↯;
3      if P↯ ≠ ∅ then
4          Make new class P↯; par(P↯) := P'↯;
5          P'↯ ∪= {P↯}; 𝒫↯ := 𝒫↯ · [P↯];
6          for b ∈ BBNodes do
7              if b ≠ move then b.LBClasses ∪= {P↯};
8          for Q ∈ move.LBClasses such that Q ↯ do
9              ⊑ ∪= {(Q, P↯)}; cnt⊑(P'↯, P'↯) += 1;
10     if P↓ ≠ ∅ then
11         Make new class P↓; par(P↓) := P'↓;
12         P'↓ ∪= {P↓}; 𝒫↓ := 𝒫↓ · [P↓];
13         for b ∈ BBNodes do
14             if b ≠ move then b.LBClasses ∪= {P↓};
15         for Q ∈ move.LBClasses do
16             ⊑ ∪= {(Q, P↓)};
17             if Q ↯ then cnt⊑(P'↯, P'↓) += 1;
18             else cnt⊑(P'↓, P'↓) += 1;
19 newBBNodes := ∅;
20 if move.left ≠ null then
21     for b ∈ BBNodes do
22         if b.left ≠ null then newBBNodes ∪= {b.left};
23         if b.right ≠ null then newBBNodes ∪= {b.right};
24     InitialPP(move.left, newBBNodes);
25 if move.right ≠ null then
26     for b ∈ BBNodes do
27         if b.right ≠ null then newBBNodes ∪= {b.right};
28     InitialPP(move.right, newBBNodes);
```

Algorithm 4: `Initialize` - Compute initial stable partition pair and initializes supporting data

```
1  root := SortStatesByOL();
2  InitialPP(root, {root});
3  𝒫 := 𝒫↯𝒫↓; 𝒫' := [];
4  if P'↯ ≠ ∅ then 𝒫' := 𝒫'[P'↯];
5  if P'↓ ≠ ∅ then 𝒫' := 𝒫'[P'↓];
6  Compute cnt→, →∃∀, and cnt∀ for P'↯ and P'↓;
7  if P'↯ ≠ ∅ and P'↓ ≠ ∅ then Refine(P'↯, P'↓, ∅);
```

with root node 4. At this point the big brothers cannot be classes that do not comprise a_2, so the candidate set is $\{4, 6\}$. We proceed, with the left subtree of node 4, leading to the leaf node comprising the class of states P_1. The big brothers follow form the left and right subtrees of $\{4, 6\}$. We obtain that P_1 is the little brother of P_1, P_2, P_4, and P_5, which can be directly verified. If we now go back to node 4 and continue to the right leaf P_2, we obtain P_2 and P_5 as its big brothers, and so on.

Now that we have sorted the states of the labeled graph according to their outgoing labels and we have computed the little brother pairs, we compute the initial partition pair, its parent partition, and supporting counters $\mathrm{cnt}_{\rightarrow}$, $\mathrm{cnt}_{\sqsubseteq}$, and $\mathrm{cnt}_{\forall}$. Note that for the initial partition we have to split the classes obtained by Algorithm 2 according to the termination options. The parent class comprises only two classes $P'_{\not\downarrow}$ and $P'_{\downarrow}$. Recall that the parent classes comprise partition classes of nodes. We do not keep reflexive little brother pairs of the form $P \sqsubseteq P$. The little brother pairs also depend on the termination options as given in Definition 3.6. For that purpose we split each class P to $P_{\not\downarrow}$ and $P_{\downarrow}$. Recall that by traversing the binary tree inorder we obtain a topological order of the classes with respect to the little brother relation. To encode the topological sorting we treat $\mathcal{P}$ as a list comprised of $\mathcal{P}_{\not\downarrow}\mathcal{P}_{\downarrow}$. Note that $\mathrm{cnt}_{\sqsubseteq}$ is computed while forming the little brother relation, whereas $\mathrm{cnt}_{\rightarrow}$ is initialized using $\rightarrow^{-1}$ for $P'_{\downarrow}$ and $P'_{\not\downarrow}$ in a standard way, where we treat $P'_{\downarrow}$ as the complete set of nodes and $P'_{\not\downarrow}$ as its splitter in order to confirm to the refinement operator, see below. From $\mathrm{cnt}_{\rightarrow}$ we compute $\rightarrow_{\exists\forall}$ and $\mathrm{cnt}_{\forall}$ accordingly. We note that if $P'_{\not\downarrow}$ and $P'_{\downarrow}$ are both nonempty, then we have to refine the initial partition pair.

Algorithm 5: `FindSplitter` - Finds a splitter and updates supporting counters

```
1  Find a splitter S' for some P' ∈ P' with respect to P or
2  otherwise Return (∅, ∅, ∅);
3  Make new parent S' and set par(P) := S' for P ∈ S';
4  P' := P' \ S'; Insert S' ≤'-before P' in P';
5  Compute cnt⊑(S', S') and cnt⊑(S', P');
6  cnt⊑(P', P') −= cnt⊑(S', S') + cnt⊑(S', P');
7  for P ∈ P do
8      for a ∈ A do cnt∀(P, a, S') := 0; cnt∀(P, a, P') := 0;
9  for Q' ∈ P' do
10     if cnt⊑(P', Q') > 0 then
11         Compute cnt⊑(S', Q'); cnt⊑(P', Q') −= cnt⊑(S', Q');
12         for P ∈ P do
13             for a ∈ A do
14                 if cnt∀(P, a, Q') > 0 then
15                     if cnt⊑(S', Q') > 0 then cnt∀(P, a, S') += 1;
16                     if cnt⊑(P', Q') > 0 then cnt∀(P, a, P') += 1;
17 L := {Q' ∈ P' | cnt⊑(Q', P') > 0};
18 for Q' ∈ P' such that cnt⊑(Q', P') > 0 do
19     Compute cnt⊑(Q', S'); cnt⊑(Q', P') −= cnt⊑(Q', S');
20 if |S'| < |P'| then tS := S'; else tS := P';
21 for p' ∈ ⋃_{X∈tS} X do
22     for p ∈ →^{-1}(p') do cnt→(p, a, tS) += 1;
23 for q ∈ →^{-1}(⋃_{X∈S'} X) do
24     cnt→(p, a, P') −= cnt→(p, a, P') − cnt→(p, a, tS);
25 Return (S', P', L);
```

After computing the initial stable partition pair, we can begin the refinement. First, we have to find a splitter and update the supporting counters as given by Algorithm 5. We note that $\mathrm{cnt}_{\forall}$ can be updated correctly for every $Q' \in \mathcal{P}'$ such that $P' \sqsubseteq' Q'$. However, to compute $\mathrm{cnt}_{\forall}$ for $Q'' \in \mathcal{P}'$ such that $Q'' \sqsubseteq' P'$, we have to update it for the splitters first, which will be computed by the refinement operator. This is an additional complication,

Algorithm 6: `Refine`(S', P', L) - Refines the partition for a given choice of splitters

```
1  F := ∅;
2  for a ∈ A do
3      for P ∈ P do SplitClass (P, a, P');
4      for Q' ∈ P' do
5          if cnt⊑(Q', P') > 0 then
6              for P ∈ P do
7                  if cnt∀(P, a, P') > 0 then cnt∀(P, a, P') += 1;
8      for P ∈ P do SplitClass (P, a, S');
9      for Q' ∈ P' do
10         if cnt⊑(Q', S') > 0 then
11             for P ∈ P do
12                 if cnt∀(P, a, S') > 0 then cnt∀(P, a, Q') += 1;
13 for a ∈ A do
14     for Q' ∈ L do
15         for P ∈ P do
16             UpdateCount∀(P, a, Q');
17 while F ≠ ∅ do
18     tmpF := F; F := ∅;
19     for a ∈ A do
20         for (Q', R') ∈ tmpF do
21             cnt⊑(Q', R') := 0;
22             for P ∈ P such that cnt∀(P, a, R') > 0 do
23                 UpdateCount∀(P, a, Q');
```

so for that reason, we keep such Q'' in the set of local little brother dependent classes L. Note that this is mandatory, as $\sqsubseteq'$ is adapted with respect to the splitters S' and $P' \setminus S'$. To update the $\text{cnt}_{\rightarrow}$ counters, we use the "process the smaller half paradigm", i.e., we choose the smaller of the two splitters S' and $P' \setminus S'$ and we update $\text{cnt}_{\rightarrow}$ as discussed above.

After choosing a splitter, we have to refine the partition $\mathcal{P}$ by employing Algorithm 6. The refinement is executed in two steps. First, we refine the partition in order to stabilize it with respect to $P' \setminus S'$ and, afterwards, we refine with respect to S'. We note that we can do the refinement in one step, like in [15, 1], but the procedure gets quite complicated due to the possible combinations regarding little brother relation between S' and $P' \setminus S'$. Moreover, this does not change asymptotic time complexity, as we have to perform some operations twice, so for the sake of clarity of presentation, we refine the partition separately for both splitters. The main reason that there is no gain in time complexity is that, unlike the bisimulation case, we do not always have to split a class with respect to every splitter. Whether there is need to split a class is deduced from the stability conditions, i.e., if there exists a stable big brother, there is no splitting. After updating the $\text{cnt}_{\forall}$ counter for some class, we also need to update its little brothers, as we introduce an additional stable big brother for them. What follows is the updating of the counter $\text{cnt}_{\forall}$ with respect to the little brothers of the parent that has been split. Finally, we take into consideration the failed little brother parent pairs, which need to be eliminated, and have an effect on $\text{cnt}_{\forall}$.

The refinement operator employs Algorithm 7 to split a single class in order to make it stable with respect to a splitter. If the class has a stable big brother, then there is no need for it to be split. Otherwise, we check if the any of the nodes of the class have transitions

Algorithm 7: `SplitClass` (P, a, R') - Splits P to make it stable with respect to R

```
1  if cnt_∀(P, a, R') = 0 then
2      if P -a->_∃∀ R' then
3          P_♯ := {p ∈ P | p -a-/-> R'}; P := P \ P_♯;
4          if P_♯ ≠ ∅ and P ≠ ∅ then
5              Make new class P_♯; ⊑ ∪= (P_♯, P);
6              cnt_⊑(par(P), par(P)) += 1;
7              Copy →_∃∀, cnt_∀, ⊑, and par from P to P_♯;
8              →_∃∀ \= {(P, a, R')};
9              cnt_∀(P_♯, a, R') := 0; cnt_∀(P, a, R') = 1;
10             Insert P_♯ ≤-before P in 𝒫;
11             UpdateLittleBros_♯ (P_♯, a, R');
12         else if P_♯ ≠ ∅ then
13             P := P_♯; →_∃∀ \= (P, a, R');
14             UpdateLittleBros_♯ (P, a, R');
15         else cnt_∀(P, a, R') = 1;
16     else UpdateLittleBros_♯ (P, a, R');
```

to the splitter. If they do, then we proceed with splitting the class and adjusting the little brothers, whereas in the other case we just update the little brother relation. The updating is correct as all topologically smaller classes have already been updated together with their big brothers.

Algorithm 8: `UpdateLittleBros`$_\sharp$ (P, a, R') - Updates the little brothers of a split class

```
1  for Q ∈ 𝒫 such that Q ⊑ P do
2      if cnt_∀(Q, a, R') > 0 then
3          ⊑ \= {(Q, P)}; cnt_⊑(par(Q), par(P)) -= 1;
4          if cnt_⊑(par(Q), par(P)) = 0 then
5              cnt_⊑(par(Q), par(P)) := 1;
6              ℱ ∪= (par(Q), par(P));
```

To update the little brother relation of split classes we employ Algorithm 8. The algorithm checks if the stability condition 2 of Definition 3.4 is violated by comparing the $\text{cnt}_\forall$ counters. We note that for every little brother pair that no longer holds, we have to update the $\text{cnt}_\sqsubseteq$ counters. If they become zero, the parent little brother relation no longer holds. These pairs are then kept in $\mathcal{F}$ for global update of the little brother relation. We note that for consistency we have to keep the pairs as if they are still little brothers and update all of them later in the last part of Algorithm 6.

To update the $\text{cnt}_\forall$ counters when some little brother parent pairs are deleted, we employ Algorithm 9. If the $\text{cnt}_\forall$ counter decreases to zero, we have to update little brother pairs and the $\text{cnt}_\forall$ counters for those pairs. If in addition, there exists a $\rightarrow_{\exists\forall}$ transition, then it has to be stabilized.

Finally, we put all pieces together in Algorithm 10. Following the initialization, we refine the initial partition until there are no more splitters, i.e., $\mathcal{P} = \mathcal{P}'$. When the partition is stable, we compute the simulation quotient by traversing the partition in reverse topological order and only keeping the big brothers.

Algorithm 9: `UpdateCount`$_\forall(P, a, R')$ - Updates $\mathrm{cnt}_\forall$ by reducing it by one

```
1 cnt_∀(P, a, R') −= 1;
2 SplitClass(P, a, R');
3 if cnt_∀(P, a, R') = 0 then
4 |  K := {Q' ∈ P' | cnt_⊑(Q', R') > 0};
5 |  while Q' ∈ K do
6 |  |  K \= {Q'}; cnt_∀(P, a, Q') −= 1;
7 |  |  SplitClass(P, a, Q');
8 |  |  if cnt_∀(P, a, Q') = 0 then
9 |  |  |  K ∪= {Q'' ∈ P' | cnt_⊑(Q'', Q') > 0};
```

Algorithm 10: `Minimization` - Computes the simulation quotient of $G = (\mathcal{S}, \mathcal{A}, \downarrow, \rightarrow)$

```
1 Initialize;
2 (S', P', L) := FindSplitter;
3 while S' ≠ ∅ do
4 |  Refine(S', P', L);
5 |  (S', P', L) := FindSplitter;
6 for P ∈ P in reverse topological order do
7 |  for Q ∈ P in reverse topological order do
8 |  |  if Q ⊑ P then P \= {Q};
```

We can split the time cost of Algorithm 10 on the cost for the initialization and the cost for refinement. By $|\mathcal{P}|$ we denote that number of classes in the minimized graph. The time complexity of Algorithm 2 is known to be $\mathcal{O}(|\mathcal{A}||\rightarrow|)$ [1]. For Algorithm 3 we have $\mathcal{O}(|\mathcal{A}||\mathcal{P}|^2)$ as the depth of the tree is $|\mathcal{A}|$, whereas there are at most $|\mathcal{P}|$ little brothers for each of the $|\mathcal{P}|$ classes. The initialization of Algorithm 4 then costs $\mathcal{O}(|\mathcal{A}||\rightarrow| + |\mathcal{A}||\mathcal{P}|^2)$. The loop in Algorithm 10 will be executed $|\mathcal{P}|$ time, as this is the number of classes. The updating of counters in Algorithm 5 then costs $\mathcal{O}(|\mathcal{A}||\mathcal{P}|^3 + |\rightarrow| \log(|\mathcal{P}|))$ [17, 2]. For the refinement, we spend $\mathcal{O}(|\mathcal{P}||\mathcal{S}|)$ for splitting the classes, and $\mathcal{O}(|\mathcal{A}||\mathcal{P}|^3)$ for updating the counters. For the failed little brother parent pairs, we note that $\mathcal{F}$ can contain at most $|\mathcal{P}|^2$ pairs during the whole algorithm execution as this is the maximum of little brother pairs that may exist and if once a little brother pair has been deleted, it cannot appear again. Thus, this update takes in total $\mathcal{O}(|\mathcal{A}||\mathcal{P}|^3)$ as well. The computation of the simulation quotient costs $\mathcal{O}(|\mathcal{P}|^2)$. Thus, the time complexity of Algorithm 10 amounts to $\mathcal{O}(|\mathcal{A}||\rightarrow| + |\mathcal{P}||\mathcal{S}| + |\mathcal{A}||\mathcal{P}|^3)$.

For space complexity we have $\mathcal{O}(|\mathcal{P}|^2)$ for the little brother relation [2, 5], $\mathcal{O}(|\mathcal{A}||\mathcal{P}|^2 \log(|\mathcal{P}|))$ needed for the counters, and $\mathcal{O}(|\mathcal{S}| \log(|\mathcal{P}|))$ for the partition [2], which amounts to $\mathcal{O}(|\mathcal{S}| \log(|\mathcal{P}|) + |\mathcal{A}||\mathcal{P}|^2 \log(|\mathcal{P}|))$.

5 Concluding Remarks and Discussion

We have enhanced the algorithm of [2, 5] with a local update of the topological order, more efficient splitting of the classes based on the "process the smaller half method", and efficient update of the little brother relation. This resulted in an algorithm with time complexity of $\mathcal{O}(|\mathcal{A}||\rightarrow| + |\mathcal{P}||\mathcal{S}| + |\mathcal{A}||\mathcal{P}|^3)$, which is comparable with the fastest known bound of $\mathcal{O}(|\mathcal{P}||\rightarrow|)$ [6]. The addition of counters slightly increased the space complexity

bounds from $\mathcal{O}(|\mathcal{S}|\log(|\mathcal{P}|)+|\mathcal{P}|^2))$ to $\mathcal{O}(|\mathcal{S}|\log(|\mathcal{P}|)+|\mathcal{P}|^2\log(|\mathcal{P}|))$. Asymptotically our results match the ones from [13], as the worst-case bounds both for $|\rightarrow|$ and $|\mathcal{P}|^2|\mathcal{A}|$ amount to $|\mathcal{A}||\mathcal{S}|^2$.

Nonetheless, this brief analysis leaves us with an open question regarding transformations from labeled transition systems to Kripke structures and vice versa. Namely, the effect of the action labels $|\mathcal{A}|$ disappears when considering Kripke structures, but we were unable to find out how much does such a translation cost, as the number of states of the Kripke structure does increase. Now, if the factor of increase of states is less than $\sqrt[3]{|\mathcal{A}|}$, then one can profit from performing the minimization on Kripke structures instead of labeled transitions systems, whereas if the factor is greater, the minimization on labeled transition systems is faster. Note that we do not take into consideration the cost of the translation, which we hope to be linear in the number of states. We intend to investigate this question further, as space-efficient transformation that preserve certain semantics between Kripke structures and labeled transition systems may prove useful, when computing the abstractions depends on the number of action labels.

We intend to employ the presented algorithm as basis for a minimization algorithm for the partial bisimulation preorder, which bisimulates only a subset of the labeled transitions, whereas it simulates the rest. We will employ this minimization to reduce the size of the uncontrolled system in a supervisory control framework that automatically synthesizes control software [14]. Furthermore, the algorithm can also serve as basis for computing minimization with respect to so-called covariant-contravariant simulations, which have recently been shown related to the notion of modal transition systems [18].

Bibliography

[1] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.

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

[3] D. Bustan and O. Grumberg, "Simulation-based minimization," *ACM Transactions on Computational Logic*, vol. 4, pp. 181–206, 2003.

[4] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke, "Computing simulations on finite and infinite graphs," in *IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society Press, 1996, pp. 453–462.

[5] R. J. v. Glabbeek and B. Ploeger, "Correcting a space-efficient simulation algorithm," in *Proceedings of CAV*, ser. Lecture Notes in Computer Science, vol. 5123. Springer, 2008, pp. 517–529.

[6] F. Ranzato and F. Tapparo, "An efficient simulation algorithm based on abstract interpretation," *Information and Computation*, vol. 208, pp. 1–22, 2010.

[7] R. J. v. Glabbeek, "The linear time–branching time spectrum I," *Handbook of Process Algebra*, pp. 3–99, 2001.

[8] D. Dams, O. Grumberg, and R. Gerth, "Generation of reduced models for checking fragments of CTL," in *Proceedings of CAV 1993*, ser. Lecture Notes in Computer Science. Springer, 1993, vol. 697, pp. 479–490.

[9] B. Ploeger, "Improved verification methods for concurrent systems," Ph.D. dissertation, Eindhoven University of Technology, 2009.

[10] S. Crafa, F. Ranzato, and F. Tapparo, "Saving space in a time efficient simulation algorithm," in *Proceedings of ACSD 2009*. IEEE Computer Society, 2009, pp. 60–69.

[11] M. A. Reniers and T. A. C. Willemse, "Folk theorems on the correspondence between state-based and event-based systems," in *Proceedings of SOFSEM 2011*, ser. Lecture Notes in Computer Science. Springer, 2011, vol. 6543, pp. 494–505.

[12] F. Ranzato and F. Tapparo, "A new efficient simulation equivalence algorithm," in *Proceedings of LICS 2007*. IEEE Computer Society, 2007, pp. 171–180.

[13] ——, "A time and space efficient simulation algorithm," in *Proceedings of LICS 2009*. IEEE, 2009, A short talk.

[14] J. C. M. Baeten, D. A. van Beek, B. Luttik, J. Markovski, and J. E. Rooda, "A process-theoretic approach to supervisory control theory," in *Proceedings of ACC 2011*. IEEE, 2011, available from: `http://se.wtb.tue.nl`.

[15] J.-C. Fernandez, "An implementation of an efficient algorithm for bisimulation equivalence," *Science of Computer Programming*, vol. 13, no. 2-3, pp. 219–236, 1990.

[16] J. C. M. Baeten, T. Basten, and M. A. Reniers, *Process Algebra: Equational Theories of Communicating Processes*, ser. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2010, vol. 50.

[17] R. Paige and R. E. Tarjan, "Three partition refinement algorithms," *SIAM Journal on Computing*, vol. 16, no. 6, pp. 973–989, 1987.

[18] L. Aceto, I. Fabregas, D. de Frutos Escrig, A. Ingolfsdottir, and M. Palomino, "Relating modal refinements, covariant-contravariant simulations and partial bisimulations," in *Proceedings of FSEN 2011*, ser. Lecture Notes in Computer Science. Springer, 2011, to appear.