Simulation Based Minimization

Doron Bustan and Orna Grumberg

Computer Science Dept.
Technion, Haifa 32000, Israel
orna@cs.technion.ac.il

Abstract. This work presents a minimization algorithm. The algorithm receives a Kripke structure $M$ and returns the smallest structure that is simulation equivalent to $M$. The simulation equivalence relation is weaker than bisimulation but stronger than the simulation preorder. It strongly preserves ACTL and LTL (as sub-logics of ACTL$^*$).

We show that every structure $M$ has a unique up to isomorphism reduced structure that is simulation equivalent to $M$ and smallest in size.

We give a Minimizing Algorithm that constructs the reduced structure. It first constructs the quotient structure for $M$, then eliminates transitions to little brothers and finally deletes unreachable states.

The first step has maximal space requirements since it is based on the simulation preorder over $M$. To reduce these requirements we suggest the Partitioning Algorithm which constructs the quotient structure for $M$ without ever building the simulation preorder. The Partitioning Algorithm has a better space complexity but might have worse time complexity.
1 Introduction

Temporal logic model checking is a method for verifying finite-state systems with respect to propositional temporal logic specifications. The method is fully automatic and quite efficient in time, but is limited by its high space requirements. Many approaches to beat the state explosion problem of model checking have been suggested, including abstraction, partial order reduction, modular methods, and symmetry ([CGP99]). All are aimed at reducing the size of the model (or Kripke structure) to which model checking is applied, thus, extending its applicability to larger systems.

Abstraction methods, for instance, hide some of the irrelevant details of a system and then construct a reduced structure. The abstraction is required to be weakly preserving, meaning that if a property is true for the abstract structure then it is also true for the original one. Sometimes we require the abstraction to be strongly preserving so that, in addition, a property that is false for the abstract structure, is also false for the original one.

In a similar manner, for modular model checking we construct a reduced abstract environment for a part of the system that we wish to verify. In this case as well, properties that are true (false) of the abstract environment should be true (false) of the real environment.

It is common to define equivalence relations or preorders on structures in order to reflect strong or weak preservation of various logics. For example, language equivalence (containment) strongly (weakly) preserves the linear-time temporal logic LTL. Other relations that are widely used are the bisimulation equivalence [Par81] and the simulation preorder [Mil71]. The former guarantees strong preservation of branching-time temporal logics such as CTL and CTL* [CE81]. The latter guarantees weak preservation of the universal fragment of these logics (ACTL and ACTL* [GL94]).

Bisimulation has the advantage of preserving more expressive logics. However, this is also a disadvantage since it requires the abstract structure to be too similar to the original one, thus allowing less powerful reductions. The simulation preorder, on the other hand, allows more powerful reductions, but it provides only weak preservation. Language equivalence provides strong preservation and large reduction, however, its complexity is exponential while the complexity to compute bisimulation and simulation is polynomial.

In this paper we investigate the simulation equivalence relation that is weaker than bisimulation but stronger than the simulation preorder and language equivalence. Simulation equivalence strongly preserves ACTL*, and also strongly preserves LTL and ACTL as sublogics of ACTL*. Both ACTL and LTL are widely used for model checking in practice.

As an equivalence relation that is weaker than bisimulation, it can derive smaller minimized structure. For example, the structure in part 2 of Figure 1 is minimized with respect to simulation equivalence. In comparison, the minimized structure with respect to bisimulation is the structure in part 1 of Figure 1 and the minimized structure with respect to language equivalence is the structure in part 3 of Figure 1.
Fig. 1. Different minimized structures with respect to different equivalence relations

Given a Kripke structure $M$, we would like to find a structure $M'$ that is simulation equivalent to $M$ and is the smallest in size (number of states and transitions).

For bisimulation this can be done by constructing the quotient structure in which the states are the equivalence classes with respect to bisimulation. Bisimulation has the property that if one state in a class has a successor in another class then all states in the class have a successor in the other class. Thus, in the quotient structure there will be a transition between two classes if every (some) state in one class has a successor in the other. The resulting structure is the smallest in size that is bisimulation equivalent to the given structure $M$.

The quotient structure for simulation equivalence can be constructed in a similar manner. There are two main difficulties, however. First, it is not true that all states in an equivalence class have successors in the same classes. As a result, if we define a transition between classes whenever all states of one have a successor in the other, then we get the $\forall$-quotient structure. If, on the other hand, we have a transition between classes if there exists a state of one with a successor in the other, then we get the $\exists$-quotient structure. Both structures are simulation equivalent to $M$, but the $\forall$-quotient structure has fewer transitions and therefore is preferable.

The other difficulty is that the quotient model for simulation equivalence is not the smallest in size. Actually, it is not even clear that there is a unique smallest structure that is simulation equivalent to $M$.

The first result in this paper is showing that every structure has a unique up to isomorphism smallest structure that is simulation equivalent to it. This structure is reduced, meaning that it contains no simulation equivalent states, no little brothers (states that are smaller by the simulation preorder than one of their brothers), and no unreachable states.

Our next result is presenting the Minimizing Algorithm that given a structure $M$ constructs the reduced structure for $M$. Based on the maximal simulation relation over $M$, the algorithm first builds the $\forall$-quotient structure with re-
spect to simulation equivalence. Then it eliminates transitions to little brothers. Finally, it removes unreachable states. The time complexity of the algorithm is $O(|S|^3)$. Its space complexity is $O(|S|^2)$ which is due to the need to hold the simulation preorder in memory.

Since our main concern is space requirements, we suggest the Partitioning Algorithm which computes the quotient structure without ever computing the simulation preorder. Similarly to [LY92], the algorithm starts with a partition $\Sigma_0$ of the state space to classes whose states are equally labeled. It also initializes a preorder $H_0$ over the classes in $\Sigma_0$. At iteration $i+1$, $\Sigma_{i+1}$ is constructed by splitting classes in $\Sigma_i$. The relation $H_{i+1}$ is updated based on $\Sigma_i$, $\Sigma_{i+1}$ and $H_i$.

When the algorithm terminates (after $k$ iterations) $\Sigma_k$ is the set of equivalence classes with respect to simulation equivalence. These classes form the states of the quotient structure. The final $H_k$ is the maximal simulation preorder over the states of the quotient structure. Thus, the Partitioning Algorithm replaces the first step of the Minimizing Algorithm. Since every step in the Minimizing Algorithm further reduces the size of the initial structure, the first step handles the largest structure. Therefore, improving its complexity influences most the overall complexity of the algorithm.

The space complexity of the Partitioning Algorithm is $O(|\Sigma_k|^3 + |S| \cdot \log(\Sigma_k))$. We assume that in most cases $|\Sigma_k| \ll |S|$, thus this complexity is significantly smaller than that of the Minimizing Algorithm. Unfortunately, time complexity will probably become worse (depending on the size of $\Sigma_k$). It is bounded by $O(|S|^3 \cdot |\Sigma_k|^2 \cdot (|\Sigma_k|^2 + R))$). However, since our main concern is the reduction in memory requirements, the Partitioning Algorithm is valuable.

Other works also suggest minimization algorithms. In [LY92], the quotient structure with respect to bisimulation is constructed without first building the bisimulation relation. We follow a similar approach. However, in our case states may remain in the same class even when they do not have successors in the same classes. Thus, our analysis is more complicated and requires both $\Sigma_i$ and $H_i$. Symbolic bisimulation minimization is suggested in [BdS92]. In [BFH90] a minimized structure with respect to bisimulation is generated directly out of the text. In [FV89] a bisimulation minimization is applied to the intersection of the system automaton and the specification automaton. The algorithm from [LY92] is used. [2] shows that eliminating little brothers results in a simulation equivalent structure. However, the paper does not consider the minimization problem.

Several works minimize a structure in a compositional way, preserving language containment [ASSB94] or a given CTL formula [ASSSV94]. Minimizing with respect to a given formula may result in a more power reduction, however it requires to determine the checked formula in advance.

The rest of the paper is organized as follows. Section 2 gives our basic definitions. Section 3 defines reduced structures and shows that every structure has a unique simulation equivalent reduced structure. Section 4 presents the Minimizing Algorithm. Finally, Section 5 describes the Partitioning Algorithm and discusses its space and time complexity.
2 Preliminaries

Let $AP$ be a set of atomic propositions. A Kripke structure $M$ over $AP$ is a four tuple $M = (S, s_0, R, L)$ where $S$ is a finite set of states; $s_0 \in S$ is the initial state; $R \subseteq S \times S$ is the transition relation that must be total, i.e., for every state $s \in S$ there is a state $s' \in S$ such that $R(s, s')$; and $L : S \rightarrow 2^{AP}$ is a function that labels each state with the set of atomic propositions true in that state.

The size $|M|$ of a Kripke structure $M$ is the pair $(|S|, |R|)$. We say that $|M| \leq |M'|$ if $|S| \leq |S'|$ or $|S| = |S'|$ and $|R| \leq |R'|$.

Given two structures $M$ and $M'$ over $AP$, a relation $H \subseteq S \times S'$ is a simulation relation [Mil91] over $M \times M'$ iff the following conditions hold:

1. $(s_0, s'_0) \in H$.
2. For all $(s, s') \in H$, $L(s) = L'(s')$ and

$$\forall t \exists t' [(s, t) \in R \rightarrow (s', t') \in R' \land (t, t') \in H]].$$

We say that $M'$ simulates $M$ (denoted by $M \preceq M'$) if there exists a simulation relation $H$ over $M \times M'$.

The logic $ACTL^*$ [GL94] is the universal fragment of the powerful branching-time logic $CTL^*$. $ACTL^*$ consists of the temporal operators $X$ (next-time), $U$ (until) and $R$ (release) and the universal path quantifier $A$ (for all paths). The formal definition is omitted and can be found in [CGP99].

The following lemma and theorem have been proven in [GL94].

Lemma 1. $\preceq$ is a preorder on the set of structures.

Theorem 2. Suppose $M \preceq M'$. Then for every $ACTL^*$ formula $f$, $M' \models f$ implies $M \models f$.

Given two Kripke structures $M, M'$, we say that $M$ is simulation equivalent to $M'$ if $M \preceq M'$ and $M' \preceq M$. It is easy to see that this is an equivalence relation.

A simulation relation $H$ over $M \times M'$ is maximal iff for all simulation relations $H'$ over $M \times M'$, $H' \subseteq H$.

In [GL94] it has been shown that if there is a simulation relation over $M \times M'$ then there is a unique maximal simulation over $M \times M'$.

3 The Reduced Structure

Given a Kripke structure $M$, we would like to find a reduced structure, that will be simulation equivalent to $M$ and smallest in size. In this section we prove that a reduced structure always exists. Furthermore, we show that all reduced structures of $M$ are isomorphic to each other.

Let $M$ be a Kripke structure and $H$ be the maximal simulation relation over $M \times M$. We need the following definitions in order to characterize reduced structures.

Two states $s_1, s_2 \in M$ are simulation equivalent iff $(s_1, s_2) \in H$ and $(s_2, s_1) \in H$.

A state $s_1$ is a little brother of a state $s_2$ iff there exists a state $s_3$ such that:
- \((s_0, s'_0) \in R\) and \((s_0, s_1) \in R\).
- \((s_1, s_2) \in H\) and \((s_2, s'_1) \notin H\).

**Definition 3.** A Kripke structure \(M\) is reduced if:

1. There are no simulation equivalent states in \(M\).
2. There are no states \(s_1, s_2\) such that \(s_1\) is a little brother of \(s_2\).
3. All states in \(M\) are reachable from \(s_0\).

**Theorem 4.** Let \(M, M'\) be two reduced Kripke structures. Then the following two statements are equivalent:

1. \(M\) and \(M'\) are simulation equivalent.
2. \(M\) and \(M'\) are isomorphic.

The proof that 2 implies 1 is straightforward. In the rest of this section we assume that \(M\) and \(M'\) are reduced Kripke structures. We will show that if \(M \preceq M'\) and \(M' \preceq M\) then \(M\) and \(M'\) are isomorphic.

We use \(H_{MM}\) to denote the maximal simulation over \(M \times M'\), and \(H_{M'M}\) to denote the maximal simulation over \(M' \times M\). The composed relation \(H_{MM' \subseteq S \times S}\) is defined by

\[
H_{MM' \subseteq S \times S} = \{(s_1, s_2) | \exists s' \in S'. (s_1, s') \in H_{MM} \land (s', s_2) \in H_{M'M}\}
\]

**Lemma 5.** The composed relation \(H_{MM' \subseteq S \times S}\) is a simulation relation.

**Proof:**

- \((s_0, s'_0) \in H_{MM}\) and \((s'_0, s_0) \in H_{M'M}\) implies \((s_0, s_0) \in H_{MM' \subseteq S \times S}\).
- \((s_1, s_2) \in H_{MM' \subseteq S \times S}\) implies that there exists a state \(s'\) in \(M'\) such that \((s_1, s') \in H_{MM}\) and \((s', s_2) \in H_{M'M}\). Thus, \(L(s_1) = L'(s') = L(s_2)\).
- Let \((s_1, s_2) \in H_{MM' \subseteq S \times S}\) and let \(t_1\) be a successor of \(s_1\). We will show that there exists a successor \(t_2\) of \(s_2\) such that \((t_1, t_2) \in H_{M'M}\).
  - \((s_1, s_2) \in H_{MM' \subseteq S \times S}\) implies that there exists \(s'\) such that \((s_1, s') \in H_{MM}\) and \((s', s_2) \in H_{M'M}\).
  - \((s_1, s') \in H_{MM}\) implies that there exists a successor \(t' \in S'\) of \(s'\) such that \((t_1, t') \in H_{MM}\).
  - \((s_1, s') \in H_{M'M}\) implies that there exists a successor \(t_2 \in S\) of \(s_2\) such that \((t', t_2) \in H_{M'M}\).
  - By the above, \((t_1, t_2) \in H_{M'M}\). \(\square\)

Given two reduced Kripke structures \(M\) and \(M'\), we will define a matching relation \(f\) over \(S' \times S\) based on the two simulation relations between the structures. We show that \(f\) is an isomorphism between \(M'\) and \(M\), i.e., \(f\) is an one to one and onto total function that preserves the state labeling and the transition relation.

**Definition 6.** The matching relation \(f \subseteq S' \times S\) is defined as follow: \((s', s) \in f\) iff \((s', s) \in H_{M'M}\) and \((s, s') \in H_{M'M}\).
Lemma 7. Let \( f \subseteq S' \times S \) be the matching relation. Then \( f \) is an one to one, onto, and total function from \( S' \) to \( S \).

Proof: First we prove that \( f \) is a function from \( S' \) to \( S \). Assume to the contrary that there are different states \( s_1, s_2 \) in \( S \) and \( s' \) in \( S' \) such that \( (s', s_1) \in f \) and \( (s', s_2) \in f \). Let \( H_{MM}' \) be the composed relation. Since \( H_{MM}' \) is a simulation relation, it is included in the maximal simulation over \( M \times M \). We will show that \( (s_1, s_2) \in H_{MM}' \) and \( (s_2, s_1) \in H_{MM}' \), which contradicts the assumption that \( M \) is reduced.

- \( (s', s_1) \in f \) implies \( (s', s_1) \in H_{MM}' \) and \( (s_1, s') \in H_{MM}' \).
- \( (s', s_2) \in f \) implies \( (s', s_2) \in H_{MM}' \) and \( (s_2, s') \in H_{MM}' \).
- \( (s_1, s') \in H_{MM}' \) and \( (s', s_2) \in H_{MM}' \) implies \( (s_1, s_2) \in H_{MM}' \).
- \( (s_2, s') \in H_{MM}' \) and \( (s', s_1) \in H_{MM}' \) implies \( (s_2, s_1) \in H_{MM}' \).

The proof that \( f^{-1} \) is a function from \( S \) to \( S' \) is similar. Thus, we conclude that \( f \) is one to one.

Next, we prove that \( f \) is onto, i.e. for every state \( s \) in \( S \) there exists a state \( s' \) in \( S' \) such that \( (s', s) \in f \). The proof is by induction on the distance of \( s \) from the initial state. (since all states are reachable, the distance is bounded by \(|S|\)).

- Base: The case where the distance is 0 follows from the fact that simulation relations relate initial states to each other. Thus, \( (s_0, s_0) \in H_{MM}' \).
- Induction step: Assume that the induction hypothesis holds for every state with distance less than or equal to \( n \). We prove it for states with distance \( n+1 \). Let \( t_1 \in S \) be a state with distance \( n \). Then there is a state \( s \) with distance \( n \) such that \( (s, t_1) \in R \). By the induction hypothesis, there exists a state \( s' \) in \( S' \) such that \( (s, s') \in H_{MM}' \) and \( (s', s) \in H_{MM}' \). By the definition of simulation, for every successor of \( s \), in particular \( t_1 \), there exists a successor \( t_1' \) of \( s' \) in \( S' \) such that \( (t_1, t_1') \in H_{MM}' \). If in addition \( (t_1, t_1') \in H_{MM}' \) then \( (t_1, t_1') \in f \) and we are done. Assume to the contrary that \( (t_1, t_1') \notin H_{MM}' \), then \( (s', s) \in H_{MM}' \) implies that there exists \( t_2 \) such that \( (s, t_2) \in R \) and \( (t_1, t_2) \in H_{MM}' \). Let \( H_{MM}' \) be the composed simulation relation. Then, \( H_{MM}' \) is included in the maximal simulation over \( M \times M \). \( (t_1, t_1') \in H_{MM}' \) and \( (t_1', t_2) \in H_{MM}' \) implies \( (t_1, t_2) \in H_{MM}' \). However \( t_1, t_2 \) are both successors of \( s \). This implies that either \( t_1, t_2 \) are simulation equivalent or \( t_1 \) is a little brother of \( t_2 \), contradicting the assumption that \( M \) is reduced.

A similar proof can be applied to show that \( f^{-1} \) is onto which implies that \( f \) is total. \( \square \)

Lemma 8. Let \( s', t' \in S' \) be states, then \( (s', t') \in R' \) iff \( (f(s'), f(t')) \in R \). 

Proof: We prove that if \( (s', t'_1) \in R \) then \( (f(s'), f(t'_1)) \in R \). The proof of the other direction is similar. Let \( s', t'_1 \in S' \) be two states such that \( (s', t'_1) \in R' \).
and let \( s, t_1 \in S \) be states such that \( f(s') = s \) and \( f(t'_1) = t_1 \). Assume to the contrary that \((s, t_1) \notin R\). Then \((s', s) \in H_{M \cdot M'}\) implies that there exists \( t_2 \) such that \((s, t_2) \in R\) and \((t'_1, t_2) \in H_{M \cdot M'}\). Moreover \((s', s) \in H_{M \cdot M'}\) implies that there exists \( t'_2 \) such that \((s', t'_2) \in R'\) and \((t_2, t'_2) \in H_{M \cdot M'}\). We distinguish between two cases:

1. If \( t'_2 = t'_1 \) then \( f(t'_1) = t_2 \), contradicting the assumption that \( f \) is a function.
2. Otherwise, let \( H_{M' \cdot M' \cdot M} \) be the composed simulation relation over \( M' \times M' \).

Therefore, it is included in the maximal simulation over \( M' \times M' \): \((t'_1, t_2) \in H_{M' \cdot M' \cdot M} \) and \((t_2, t'_2) \in H_{M' \cdot M' \cdot M} \). This implies that either \( t'_1, t'_2 \) are simulation equivalent or \( t'_1 \) is a little brother of \( t'_2 \), contradicting the assumption that \( M' \) is reduced.

\( \square \)

**Proposition 9.** For all \( s' \in S' \), \( L'(s') = L(f(s')) \).

**Proof:** immediate by definition of \( f \).

We showed that for reduced structures \( M \) and \( M' \), if they are simulation equivalent then there exists a one to one, onto, total function \( f : S' \to S \) such that for every \( s' \), \( L'(s') = L(f(s')) \) and for every \( s', t', (s', t') \in R' \) iff \((f(s'), f(t')) \in R\). Thus, we conclude Theorem 4.

**Theorem 10.** Let \( M \) be a non-reduced Kripke structure, then there exists a reduced Kripke structure \( M' \) such that \( M, M' \) are simulation equivalent and \( |M'| < |M| \).

In order to prove Theorem 10, we present in the next sections an algorithm that receives a Kripke structure \( M \) and computes a reduced Kripke structure \( M' \), which is simulation equivalent to \(|M|\), such that \(|M'| \leq |M|\). Moreover, if \( M \) is not reduced then \(|M'| < |M|\).

The following lemma shows that the reduced structures are strictly smaller than any other structure that is simulation equivalent to them.

**Lemma 11.** Let \( M' \) be a reduced Kripke structure. For every \( M \) that is simulation equivalent to \(|M'|\), if \( M \) and \( M' \) are not isomorphic then \(|M'| < |M|\).

**Proof:** By Theorem 4, since \( M \) is not isomorphic to \( M' \), \( M \) is not reduced. By Theorem 10 there exists a reduced Kripke structure \( M'' \) which is simulation equivalent to \( M \) and \(|M''| < |M|\). \( M'' \) and \( M' \) are both simulation equivalent to \( M \) and therefore are simulation equivalent to each other. Since they are reduced, they are also isomorphic and therefore \(|M'| = |M''|\). Thus, \(|M'| < |M|\). \( \square \)

### 4 The Minimizing Algorithm

In this section we present the Minimizing Algorithm that gets a Kripke structure \( M \) and computes a reduced Kripke structure \( M' \) which is simulation equivalent to \( M \) and \(|M'| \leq |M|\). If \( M \) is not reduced then \(|M'| < |M|\).
The algorithm consists of three steps. First, a quotient structure is constructed in order to eliminate equivalent states. The resulting quotient structure is simulation equivalent to \( M \) but may not be reduced. The next step disconnects little brothers and the last one removes all unreachable states.

In each step of the algorithm, if the resulting structure differs from the original one then the resulting one is strictly smaller than the original structure.

### 4.1 The \( \forall \)-quotient Structure

In order to compute a simulation equivalent structure that contains no equivalent states, we compute the quotient structure with respect to the simulation equivalence relation. The states of the structure are the equivalence classes and the labeling function is straightforward (all states in a given equivalence class have the same labeling, so we use this label for the class as well). However, the transition relation is not uniquely defined. We can have a transition between two equivalence classes if from every state of one there is a transition to some state of the other (\( \forall \)-transitions). We can also have a transition in case there exists a state in one with a transition to some state of the other (\( \exists \)-transitions). Both definitions will result in a simulation equivalent structure. However, the former has smaller transition relation and therefore it is preferable.

In the rest of this section we present the \( \forall \)-quotient structure and prove that it is simulation equivalent to the original structure. If the quotient structure is not isomorphic to the original one, then it is strictly smaller in size.

For the rest of this section we fix \( M \) to be the original Kripke structure and \( H \) to be the maximal simulation relation over \( M \times M \). We denote by \( [s] \) the equivalence class which includes \( s \).

**Definition 12.** The \( \forall \)-quotient structure \( M_q = \langle S_q, R_q, s_{0_q}, L_q \rangle \) of \( M \) is defined as follow:

- \( S_q \) is the set of the equivalence classes of the simulation equivalence. (We will use Greek letters to represent equivalence classes).
- \( R_q = \{ (s_1, s_2) \mid \forall s_1 \in a_1 \exists s_2 \in a_2 \cdot (s_1, s_2) \in R \} \)
- \( s_{0_q} = [s_{0}]. \)
- \( L_q([s]) = L(s). \)

Note that, \(|S_q| \leq |S|\) and \(|R_q| \leq |R|\). If \(|S_q| = |S|\), then every equivalence class contains a single state. In this case, \( R_q \) is identical to \( R \) and \( M_q \) is isomorphic to \( M \). Thus, when \( M \) and \( M_q \) are not isomorphic, \(|S_q| < |S|\).

Next, we show that \( M \) and \( M_q \) are simulation equivalent.

**Definition 13.** Let \( G \subseteq S \) be a set of states. A state \( s_m \in G \) is maximal in \( G \) iff there is no state \( s \in G \) such that \((s_m, s) \in H \) and \((s, s_m) \notin H \).

**Definition 14.** Let \( a \) be a state of \( M_q \), \( s_1 \) and \( t_1 \) a successor of some state in \( a \). The set \( G(a, t_1) \) is defined as follow:

\[
G(a, t_1) = \{ t_2 \in S | \exists s_2 \in a \land (s_2, t_2) \in R \land (t_1, t_2) \in H \}.
\]
Intuitively, \( G(\alpha, t_1) \) is the set of states that are greater than \( t_1 \) and are successors of states in \( \alpha \). Notice that since all state in \( \alpha \) are simulation equivalent, every state in \( \alpha \) has at least one successor in \( G(\alpha, t_1) \).

**Lemma 15.** Let \( \alpha, t_1 \) be as defined in Definition 14. Then for every maximal state \( t_m \) in \( G(\alpha, t_1) \), \([t_m]\) is a successor of \( \alpha \).

**Proof:** Let \( t_m \) be a maximal state in \( G(\alpha, t_1) \), and let \( s_m \in \alpha \) be a state such that \( t_m \) is a successor of \( s_m \). We prove that for every state \( s \in \alpha \), there exists a successor \( t \in [t_m] \), which implies that \([t_m]\) is a successor of \( \alpha \).

\( s, s_m \in \alpha \) implies \((s_m, s) \in H \). This implies that there exists a successor \( t \) of \( s \) such that \((t_m, t) \in H \). By transitivity of the simulation relation, \((t_1, t) \in H \). Thus \( t \in G(\alpha, t_1) \). Since \( t_m \) is maximal in \( G(\alpha, t_1) \), \((t, t_m) \in H \). Thus, \( t \) and \( t_m \) are simulation equivalent and \( t \in [t_m] \). □

**Theorem 16.** The structures \( M \) and \( M_q \) are simulation equivalent.

**Proof:** First we prove that \( M_q \preceq M \). Let \( H' \subseteq S_q \times S \) be the relation \( H' = \{(s, s) | s \in \alpha \} \). We prove that \( H' \) is a simulation relation.

- \((s_0, s_0) \in H' \) implies that \((s_0, s_0) \in H' \).
- By the definition of \( L_q \), \((\alpha, s) \in H' \) implies that \( L(s) = L_q(\alpha) \).
- Assume \((\alpha_1, s) \in H' \) and \( \alpha_2 \) be a successor of \( \alpha_1 \). Then by the definition of \( R_q \), there exists a successor \( t \) of \( \alpha_2 \) such that \( t \in \alpha_2 \). Thus, \((\alpha_2, t) \in H' \).

Second, we prove that \( M \preceq M_q \). Let \( H' \subseteq S \times S_q \) be the relation \( H' = \{(s_1, s_2) | \text{there exists a state } s_2 \in \alpha \text{ such that } (s_1, s_2) \in H \} \). We prove that \( H' \) is a simulation relation.

- \((s_0, s_0) \in H \) and \( s_0 \in s_0 \) imply that \((s_0, s_0) \in H' \).
- \((s_1, \alpha) \in H' \) implies that there exists a state \( s_2 \in \alpha \) such that \((s_1, s_2) \in H \). Thus, \( L(s_1) = L(s_2) = L_q(\alpha) \).
- Assume \((s_1, \alpha_1) \in H' \) and \( t_1 \) be a successor of \( s_1 \). We prove that there exists a successor \( \alpha_2 \) of \( \alpha_1 \) such that \((t_1, \alpha_2) \in H' \). We distinguish between two cases:

1. \( s_1 \in \alpha_1 \). Let \( t_m \) be a maximal state in \( G(\alpha_1, t_1) \), then Lemma 15 implies that \((\alpha_1, [t_m]) \in R_q \). Since \( t_m \) is maximal in \( G(\alpha_1, t_1) \), \((t_1, t_m) \in H \) which implies \((t_1, [t_m]) \in H' \).

2. \( s_1 \notin \alpha_1 \). Let \( s_2 \in \alpha_1 \) be a state such that \((s_1, s_2) \in H \). Since \((s_1, s_2) \in H \) there is a successor \( t_2 \) of \( s_2 \) such that \((t_1, t_2) \in H \). The first case implies that there exists an equivalence class \( \alpha_2 \) such that \((\alpha_1, \alpha_2) \in R_q \) and \((t_2, \alpha_2) \in H' \). By \((t_2, \alpha_2) \in H' \) we have that there exists a state \( t_3 \in \alpha_2 \) such that \((t_2, t_3) \in H \). By transitivity of simulation \((t_1, t_3) \in H \). Thus, \((t_1, \alpha_2) \in H' \). □
4.2 Disconnecting Little Brothers

Our next step is to disconnect the little brothers from their fathers. As a result of applying this step to a Kripke structure $M$ with no equivalent states, we get a Kripke structure $M'$ satisfying:

1. $M$ are equivalent to $M'$
2. There are no equivalent states in $M'$.
3. There are no little brothers in $M'$.
4. $|M'| \leq |M|$, and if $M$ and $M'$ are not identical, then $|M'| < |M|$.

In Figure 2, we present an iterative algorithm which disconnects little brothers and results in $M'$.

```plaintext
change := true
while (change = true) do
  Compute the maximal simulation relation $H$
  change := false
  If there are $s_1, s_2, s_3$ such that $s_1$ is a little brother of $s_2$
    and $s_3$ is the father of both $s_1$ and $s_2$ then
    change := true
    $R = R \setminus \{(s_3, s_1)\}$
  end
end

Fig. 2. The Disconnecting Algorithm.
```

Since in each iteration of the algorithm one transition is removed, the algorithm will terminate after at most $|R|$ iterations. We will show that the resulting structure is simulation equivalent to the original one.

**Lemma 17.** Let $M' = < S', R', s'_0, I' >$ be the result of the Disconnecting Algorithm on $M$. Then $M$ and $M'$ are simulation equivalent.

**Proof:** We prove the lemma by induction on the number of iterations.

- Base: at the beginning $M$ and $M$ are simulation equivalent.
- Induction step: Let $M'' = < S'', R'', s''_0, I'' >$ be the result of the first $i$ iterations and $H''$ be the maximal simulation over $M'' \times M''$. Let $M' = < S', R', s'_0, I' >$ be the result of the $(i+1)$th iteration where $R' = R'' \setminus \{(s''_0, s''_0)\}$. Assume that $M$ and $M''$ are simulation equivalent. We first prove that $M' \preceq M''$. We choose $H' \subseteq S' \times S'$ to be $H' = \{(s'_1, s'_2) | (s''_1, s''_2) \in H''\}$. Since $M'$ is obtained from $M''$ by removing one transition, clearly $H'$ is a simulation relation.
  We now show that $M'' \preceq M'$. Similarly to the previous case, we choose $H' \subseteq S' \times S'$ to be $H' = \{(s''_1, s''_2) | (s'_1, s'_2) \in H''\}$. We will prove that $H'$ is a simulation relation.
Lemma 18. Let $H \subseteq S \times S$ be the maximal simulation relation over $M \times M$. Let $M' \leq S, R', s_0, L >$ be the result of the Disconnecting Algorithm on $M$ and let $H' \subseteq S' \times S'$ be the maximal simulation relation over $M' \times M'$. Then, $H = H'$.

Proof: Since the Disconnecting Algorithm changes only the transition relation, we have for all intermediate structures $M'' S'' = S, s_0'' = s_0$ and $L'' = L$. We prove the lemma by induction on the number of iterations.

- Base: at the beginning $H'' = H''$.
- Induction step: Let $M'' \leq S, R'', s_0, L >$ be the result of the first $i$ iterations and let $H''$ be the maximal simulation relation over $M'' \times M''$. Assume that $H'' = H$. Let $M'$ be the result of the $(i+1)$th iteration and $H'$ be the maximal simulation relation over $M' \times M'$. We prove that $H' = H''$. First we prove that $H''$ is a simulation relation over $M' \times M'$, which implies that $H'' \subseteq H'$. ($H'$ is maximal over $M' \times M'$).

  - $(s_0, s_0) \in H''$ implies that $L(s_0) = L(s_0)$.
  - Let $s_1, s_2, t_1$ be states such that $(s_1, s_2) \in H''$ and $(s_1, t_1) \in R'$, $(s_1, s_2) \in H''$ implies that there exists a state $t_2$ such that $(s_2, t_2) \in R''$ and $(t_1, t_2) \in H''$. We distinguish between two cases:
    1. If $(s_2, t_2) \in R'$, we are done.
    2. If $(s_2, t_2) \notin R'$, then since $(s_2, t_2)$ is removed from $R''$, there must exist a state $t_3$ such that $(t_3, t_3) \in H''$ and $(s_2, t_3) \in R'$ (there is a little brother of $t_3$ and $s_2$ is the father of both states). Since only one transition is removed, $(s_2, t_3) \in R'$. By transitivity of $H''$, $(t_1, t_3) \in H''$. Thus, $H''$ is a simulation relation over $M' \times M'$.

Next we prove that $H'$ is a simulation relation over $M'' \times M''$, which implies that $H' \subseteq H''$. ($H''$ is maximal over $M' \times M'$).
• \((s_8, s_8) \in H'\).
• \((s_1, s_2) \in H'\) implies that \(L(s_1) = L(s_2)\).
• Let \(s_1, s_2, t_1\) be states such that \((s_1, s_2) \in H'\) and \((s_1, t_1) \in R''\). We distinguish between two cases:
  1. If \((s_1, t_1) \in R'\), then \((s_1, s_2) \in H'\) implies that there exists a state \(t_2\) such that \((s_2, t_2) \in R'\) and \((t_1, t_2) \in H'\). Thus, \((s_2, t_2) \in R''\).
  2. If \((s_1, t_1) \notin R'\), then since \((s_1, t_1)\) is removed from \(R''\), there exists a state \(t_3\) such that \((s_1, t_3) \in R''\) and \((t_1, t_3) \in H''\) \((t_1\) is a little brother of \(t_3\) and \(s_1\) is their father\). \((t_1, t_3) \in H''\) and \(H'' \subseteq H'\) implies \((t_1, t_3) \in H'\). Since \((s_1, t_1)\) is the only transition removed from \(R'\), \((s_1, t_3) \in R'\). This implies that there exists a state \(t_2\) such that \((s_2, t_2) \in R'\) and \((t_3, t_2) \in H'\). By transitivity of \(H'\), \((t_1, t_2) \in H'\). Thus, \((s_2, t_2) \in R''\) and \(H'\) is a simulation over \(M'' \times M''\).

\(\square\)

As a result of the last theorem, the Disconnecting Algorithm can be simplified significantly. The maximal simulation relation is computed once on the original structure \(M\) and is used in all iterations. If the algorithm is executed symbolically (with BDDs) then this operation can be performed efficiently in one step:

\[ R' = R - \{(s_1, s_2) \exists s_3 : (s_1, s_3) \in R \land (s_2, s_3) \in H \land (s_3, s_2) \notin H\}. \]

### 4.3 The Algorithm

We now present our algorithm for constructing the reduced structure for a given one.

1. Compute the \(\forall\)-quotient structure \(M_q\) of \(M\) and the maximal simulation relation \(H\) over \(M_q \times M_q\).
2. \(R' = R_q - \{(s_1, s_2) \exists s_3 : (s_1, s_3) \in R_q \land (s_2, s_3) \in H\}\)
3. Remove all unreachable states.

**Fig. 3. The Minimizing Algorithm**

Note that, in the second step we eliminate the check \((s_3, s_2) \notin H\). This is based on the fact that \(M_q\) does not contain simulation equivalent states. Removing unreachable states does not change the properties of simulation with respect to the initial states. The size of the resulting structure is equal to or smaller than the original one. Similarly to the first two steps of the algorithm, if the resulting structure is not identical then it is strictly smaller in size.

We have proved that the result of the Minimizing Algorithm \(M'\) is simulation equivalent to the original structure \(M\). Thus we can conclude that Theorem 10 is correct.

Figure 4 presents an example of the three steps of the Minimizing Algorithm applied to a Kripke structure.
1. Part 1 contains the original structure, where the maximal simulation relation is (not including the trivial pairs):
\{(2, 3), (3, 2), (11, 2), (11, 3), (4, 5), (6, 5), (7, 8), (8, 7), (9, 10), (10, 9)\}. The equivalence classes are : \{\{1\}, \{2, 3\}, \{11\}, \{4\}, \{5\}, \{6\}, \{7, 8\}, \{9, 10\}\}.

2. Part 2 presents the $\forall$–structure $M_2$. The maximal simulation relation $H$ is (not including the trivial pairs):
\[ H = \left\{ \left\{ \{11\}, \{2, 3\} \right\}, \left\{ \{4\}, \{5\} \right\}, \left\{ \{6\}, \{5\} \right\} \right\} \].

3. \{11\} is a little brother of \{2, 3\} and \{1\} is their father. Part 3 presents the structure after the removal of the transition \{(1), (11)\}.

4. Finally, part 4 contains the reduced structure, obtained by removing the unreachable states.

### 4.4 complexity

The complexity of each step of the algorithm depends on the size of the Kripke structure resulting from the previous step. In the worst case the Kripke structure does not change, thus all three steps depend on the original Kripke structure. Let $M$ be the given structure. We analyze each step separately (a naïve analysis):

1. First, the algorithm constructs equivalence classes. To do that it needs to compute the maximal simulation relation. [PB96,HHK95] showed that this
can be done in time $O(|S| \cdot |R|)$. Once the algorithm has the simulation relation, the equivalence classes can be constructed in time $O(|S|^2)$. Next, the algorithm constructs the transition relation. This can be done in time $O(|S| + |R|)$. As a whole, building the quotient structure can be done in time $O(|S| \cdot |R|)$.

2. Disconnecting little brothers can be done in $O(|S|^3)$.

3. Removing unreachable states can be done in $O(|R|)$.

As a whole the algorithm works in time $O(|S|^3)$.

The space bottle neck of the algorithm is the computation of the maximal simulation relation which is bounded by $|S|^2$.

5 Partition Classes

In the previous section, we presented the Minimizing Algorithm. The algorithm consists of three steps, each of which results in a structure that is smaller in size. Since the first step handles the largest structure, improving its complexity will influence most the overall complexity of the algorithm.

In this section we suggest an alternative algorithm for computing the set of equivalence class. The algorithm avoids the construction of the simulation relation over the original structure. As a result, it has a better space complexity, but its time complexity is worse. Since the purpose of the Minimizing Algorithm is to reduce space requirements, it is more important to reduce its own space requirement.

5.1 The Partitioning Algorithm

Let $M = < S, R, s_0, L >$ be a Kripke structure and $H$ be the maximal simulation over $M \times M$. We would like to build the equivalence classes of the simulation equivalence relation, without first calculating $H$. Our algorithm, called the Partitioning Algorithm, starts with a partition $\Sigma_0$ of $S$ to classes. The classes in $\Sigma_0$ differ from one another only by their state labeling. In each iteration, the algorithm refines the partition and forms a new set of classes. We use $\Sigma_i$ to denote the set of the classes obtained after $i$ iterations. In order to refine the partitions we build an ordering relation $H_i$ over $\Sigma_i \times \Sigma_i$ which is updated in every iteration according to the previous and current partitions ($\Sigma_{i-1}$ and $\Sigma_i$) and the previous ordering relation ($H_{i-1}$). Initially, $H_0$ includes only the identity pairs (of classes).

In the algorithm, we use $\text{succ}(s)$ for the set of successors of $s$. We use $[s]^i$ to denote the equivalence class of $s$ in $\Sigma_i$. $[s]$ is used whenever $\Sigma_i$ is clear from the context. We also use a function $\Pi$ that associates with each class $\alpha \in \Sigma_i$ the set of classes $\alpha^t \in \Sigma_{i-1}$ that contain a successor of some state in $\alpha$.

$$\Pi(\alpha) = \{[t]^{i-1} | \exists s \in \alpha. (s, t) \in R\}$$

We use the following notational convention:
The Partitioning Algorithm is presented in Figure 5.

**Initialize the algorithm:**

\[
\text{change} := \text{true}
\]

for each label \( \alpha \in 2^{AP} \) construct \( \alpha_0 \in \Sigma_0 \) such that \( s \in \alpha_0 \Leftrightarrow L(s) = \alpha \).

\( H_0 = \{(\alpha, \alpha) \mid \alpha \in \Sigma_0\} \)

while \( \text{change} = \text{true} \) do begin

\[
\text{change} := \text{false}
\]

**refine** \( \Sigma \):

\( \Sigma_{i+1} := \emptyset \)

for each \( \alpha \in \Sigma_i \) do begin

while \( \alpha \neq \emptyset \) do begin

choose \( s_p \) such that \( s_p \in \alpha \)

\( GT := \{s_0 | s_0 \in \alpha \land \forall t_p \in \text{succ}(s_p), \exists t_s \in \text{succ}(s_s). (\{t_s, t_p\}) \in H_i\} \)

\( LT := \{s_0 | s_0 \in \alpha \land \forall t_s \in \text{succ}(s_s), \exists t_p \in \text{succ}(s_p). (\{t_s, t_p\}) \in H_i\} \)

\( \alpha' := GT \cap LT \)

if \( \alpha \neq \alpha' \) then \( \text{change} := \text{true} \)

\( \alpha := \alpha \setminus \alpha' \)

Add \( \alpha' \) as a new class to \( \Sigma_{i+1} \).

end

end

**update** \( H \):

\( H_{i+1} := \emptyset \)

for every \( (\alpha_1, \alpha_2) \in H_i \) do begin

for each \( \alpha_2', \alpha_1' \in \Sigma_{i+1} \) such that \( \alpha_2 \supseteq \alpha_2', \alpha_1 \supseteq \alpha_1' \) do begin

\( \Phi = \{\phi | \exists \xi \in H(\alpha_2') \land (\phi, \xi) \in H_i\} \)

if \( \Phi \supseteq \Pi(\alpha_1') \) then

insert \( (\alpha_1', \alpha_2') \) to \( H_{i+1} \)

else

\( \text{change} := \text{true} \)

end

end

end

**Fig. 5.** The Partitioning Algorithm

**Definition 19.** The partial order \( \preceq_i \) on \( S \) is defined by: \( s_1 \preceq_i s_2 \) iff

- \( L(s_1) = L(s_2) \).
- If \( i > 0 \) then for every successor \( t_1 \) of \( s_1 \) there exists a successor \( t_2 \) of \( s_2 \) such that \( (t_1, t_2) \in H_{i-1} \).
In case \( i = 0, s_1 \leq_0 s_2 \) iff \( L(s_1) = L(s_2) \).

**Definition 26.** Two states \( s_1, s_2 \) are \( i \)-equivalent iff \( s_1 \leq_i s_2 \) and \( s_2 \leq_i s_1 \).

In the rest of this section we explain how the algorithm works. There are two invariants (formally proved later) which are preserved during the execution of the algorithm.

**Invariant 1:** For all states \( s_1, s_2 \in S \), \( s_1 \) and \( s_2 \) are in the same class \( \alpha \in \Sigma_i \) iff \( s_1 \) and \( s_2 \) are \( i \)-equivalent.

**Invariant 2:** For all states \( s_1, s_2 \in S \), \( s_1 \leq_i s_2 \) iff \( ([s_1], [s_2]) \in H_i \).

\( \Sigma_i \) is a set of equivalence classes with respect to the \( i \)-equivalence relation. In the \( i \)th iteration we split the equivalence classes of \( \Sigma_{i-1} \) so that only states that are \( i \)-equivalent remain in the same class.

A class \( \alpha \in \Sigma_{i-1} \) is repeatedly split by choosing an arbitrary state \( s_p \in \alpha \) (called the splitter) and identifying the states in \( \alpha \) that are \( i \)-equivalent to \( s_p \). These states form an \( i \)-equivalence class \( \alpha' \) that is inserted to \( \Sigma_i \).

\( \alpha' \) is constructed in two steps. First we calculate the set of states \( GT \subseteq \alpha \) that contains all states \( s_g \) such that \( s_g \leq_i s_p \). Next we calculate the set of states \( LT \subseteq \alpha \) that contains all states \( s_l \) such that \( s_l \leq_i s_p \). The states in the intersection of \( GT \) and \( LT \) are the states in \( \alpha \) that are \( i \)-equivalent to \( s_p \).

\( H_i \) captures the partial order \( \leq_i \), i.e., \( s_1 \leq_i s_2 \) iff \( ([s_1], [s_2]) \in H_i \). We later prove (Lemma 28 ) that the sequence \( \leq_0, \leq_1, \ldots \) satisfies \( \leq_0 \supseteq \leq_1 \supseteq \leq_2 \supseteq \ldots \). Therefore, if \( s_1 \leq_i s_2 \) then \( s_1 \leq_{i-1} s_2 \). Thus, \( ([s_1], [s_2]) \in H_i \) implies \( ([s_1], [s_2]) \in H_{i-1} \). Based on that, when constructing \( H_i \) it is sufficient to check \( (\alpha_1', \alpha_2') \in H_i \) only in case \( \alpha_2 \supseteq \alpha_1' \), \( \alpha_1 \supseteq \alpha_1' \), and \( (\alpha_1, \alpha_2) \in H_{i-1} \).

For suitable \( \alpha_1' \) and \( \alpha_2' \), we first construct the set \( \Phi \) of classes that are “smaller” than the classes in \( II(\alpha_1') \). By checking if \( \Phi \supseteq II(\alpha_1') \) we determine whether every class in \( II(\alpha_1') \) is “smaller” than some class in \( II(\alpha_2') \), in which case \( (\alpha_1', \alpha_2') \) is inserted to \( H_i \).

When the algorithm terminates, \( \leq_i \) is the maximal simulation relation and the \( i \)-equivalence is the simulation equivalence relation over \( M \times M \). Moreover, \( H_i \) is the maximal simulation relation over the corresponding quotient structure \( M_\Sigma \).

The algorithm runs until there is no change both in the partition \( \Sigma_i \) and in the relation \( H_i \). A change in \( \Sigma_i \) is the result of a partitioning of some class \( \alpha \in \Sigma_i \). The number of changes in \( \Sigma_i \) is bounded by the number of possible partitions, which is bounded by \( |S|^2 \).

A change in \( H_i \) results in the relation \( \leq_{i+1} \) which is contained in \( \leq_i \) and smaller in size, i.e., \( | \leq_i | > | \leq_{i+1} | \). The number of changes in \( H_i \) is therefore bounded by \( | \leq_i | \), which is bounded by \( |S|^2 \). Thus, the algorithm terminates after at most \( |S|^2 + |S| \) iterations. Note that, it is possible that in some iteration \( i \), \( \Sigma_i \) will not change but \( H_i \) will, and in a later iteration \( j > i \), \( \Sigma_j \) will change again.

**Example:** In this example we show how the Partitioning Algorithm is applied to the Kripke structure presented in Figure 6.
The Correctness of the Partitioning Algorithm

In order to prove the correctness of the Partitioning Algorithm, we prove three invariants, the first two invariants are already mentioned. The third invariant is necessary to prove the first two.

**Invariant 1:** For all states \( s_1, s_2 \in S \), \( s_1 \) and \( s_2 \) are in the same class \( \alpha \in \Sigma_1 \) iff \( s_1 \) and \( s_2 \) are \( i \)-equivalent.

Since the third iteration results in no change to the computed partition or ordering relations, the algorithm terminates. \( \Sigma_3 \) is the final set of equivalence classes which constitutes the set \( S_3 \) of states of \( M_q \). \( H_3 \) is the maximal simulation relation over \( M_q \times M_q \).

### 5.2 The Correctness of the Partitioning Algorithm

Fig. 6. An example structure
**Invariant 2:** For all states \( s_1, s_2 \in S \), \( s_1 \leq_i s_2 \) iff \([s_1], [s_2] \) \( \in H_i \).

**Invariant 3:** \( H_i \) is transitive.

We will prove these invariants by induction on \( i \).

**Base:**

1. \( s_1, s_2 \) in the same class in \( \Sigma_0 \) iff \( L(s_1) = L(s_2) \) iff \( s_1 \leq_0 s_2 \) and \( s_2 \leq_0 s_1 \) iff \( s_1 \) is \( 0 \)-equivalent to \( s_2 \).
2. \( ([s_1], [s_2]) \in H_0 \) iff \( [s_1] = [s_2] \) iff \( s_1, s_2 \) in the same class iff \( L(s_1) = L(s_2) \) iff \( s_1 \leq_0 s_2 \).
3. \( (a_1, a_2) \in H_0 \) iff \( a_1 = a_2 \). Thus, for every \( a_1, a_2, a_3 \) if \( (a_1, a_2) \in H_0 \) and \( (a_2, a_3) \in H_0 \) then \( a_1 = a_2 = a_3 \) which implies \( (a_1, a_3) \in H_0 \).

In the next three sections we prove the induction step. We assume that for every \( j \leq i \), the invariants hold for \( j \). We prove that the invariants hold for \( i+1 \).

### 5.3 Proving Invariant 1

In this section we fix \( s_p \) (the splitter) to be the state which was chosen in the partition of class \( a \), and the construction of class \( a' = GT \cap LT \).

**Proposition 21.** For every \( a' \in \Sigma_{i+1} \) there exists \( a \in \Sigma_i \) such that \( a' \subseteq a \).

We use \( a'_{\text{pre}} \) to denote the class \( a \in \Sigma_i \) which contains \( a' \in \Sigma_{i+1} \).

**Proposition 22.** Let \( a_1, a_2 \in \Sigma_{i+1} \), then \( (a_1, a_2) \in H_{i+1} \) implies that \( (a_1_{\text{pre}}, a_2_{\text{pre}}) \in H_i \).

**Corollary 23.** If states \( s_1 \) and \( s_2 \) are in the same class then \( L(s_1) = L(s_2) \).

**Lemma 24.** Let \( a' \) be a class in \( \Sigma_{i+1} \) and \( s_1 \) and \( s_2 \) be states in \( a' \), then \( s_1 \) and \( s_2 \) are \( (i+1) \)-equivalent.

**Proof:** Let \( a' \in \Sigma_{i+1} \), \( s_1, s_2 \in a' \). We prove that for every successor \( t_1 \) of \( s_1 \) there exists a successor \( t_2 \) of \( s_2 \) such that \([t_1], [t_2] \) \( \in H_i \). This implies that \( s_1 \leq_{i+1} s_2 \).

\( s_1 \in a' \) implies \( s_1 \in LT \). By the definition of \( LT \), there exists a successor \( t_p \) of \( s_p \) such that \([t_p], [t_p] \) \( \in H_i \). \( s_2 \in a' \) implies \( s_2 \in GT \). Then by the definition of \( GT \), there exists a successor \( t_2 \) of \( s_2 \) such that \([t_2], [t_2] \) \( \in H_i \). By Invariant 3, \( H_i \) is transitive, therefore \([t_1], [t_2] \) \( \in H_i \).

In a similar way we prove that \( s_2 \leq_{i+1} s_1 \). Thus \( s_1, s_2 \) are \( (i+1) \)-equivalent. \( \square \)

**Lemma 25.** Let \( s_1 \) and \( s_2 \) be \( (i+1) \)-equivalent states. Then \( s_1 \) and \( s_2 \) are in the same class in \( \Sigma_{i+1} \).

**Proof:** We will prove that \( s_2 \in [s_1] \). Let \( s_p \) be the splitter, used to construct \([s_1]\).
- Since \( s_1, s_2 \) are \((i + 1)\)-equivalent then for every successor \( t_2 \) of \( s_2 \), there exists a successor \( t_1 \) of \( s_1 \) such that \([t_2], [t_1] \in H_i\).
- Since \( s_1 \in [s_1] \) then \( s_1 \in LT \). By the definition of \( LT \), there exists a successor \( t_p \) of \( s_1 \) such that \([t_1], [t_p] \in H_i\).
- By Invariant 3, \( H_i \) is transitive. Therefore \([t_2], [t_p] \in H_i\). We proved that, for every successor \( t_2 \) of \( s_2 \), there exists a successor \( t_p \) of \( s_1 \) such that \([t_2], [t_p] \in H_i\). Thus, by definition of \( LT \), \( s_2 \in LT \).
- Since \( s_1 \in [s_1] \) then \( s_1 \in GT \). Then by the definition of \( GT \), for every successor \( t_p \) of \( s_p \), there exists a successor \( t_3 \) of \( s_1 \), such that \([t_3], [t_3] \in H_i\).
- \( s_1, s_2 \) are \((i + 1)\)-equivalent, there exists a successor \( t_4 \) of \( s_2 \) such that 
\([t_4], [t_4] \in H_i\).
- \( H_i \) is transitive, and therefore \([t_4], [t_4] \in H_i\). We proved that, for every successor \( t_p \) of \( s_1 \) there exists a successor \( t_4 \) of \( s_2 \) such that \([t_4], [t_4] \in H_i\).
- \( s_2 \in GT \) and \( s_2 \in LT \) implies \( s_2 \in [s_1] \).

By Lemma 25 and Lemma 24 we can conclude Invariant 1.

### 5.4 Proving Invariant 2

In this section we prove for \( H_{i+1} \) the property defined by Invariant 2. Since the construction of \( H_{i+1} \) is based on both \( \Sigma_i \) and \( \Sigma_{i+1} \), we need to distinguish between classes in these sets. We use \([s]_i \) and \([s]_i^+ \) to denote equivalence classes in \( \Sigma_i \) and \( \Sigma_{i+1} \) respectively.

**Lemma 26.** Let \(([s_1]_i^+, [s_2]_i^+) \in H_{i+1} \). Then for every successor \( t_1 \) of \( s_1 \), there exists a successor \( t_2 \) of \( s_2 \) such that \([t_2], [t_2] \in H_i\).

**Proof:** Let \(([s_1]_i^+, [s_2]_i^+) \in H_{i+1} \), and let \( t_1 \) be a successor of \( s_1 \). Then \([t_1]_i \in H \) \(([s_1]_i^+) \). Since \( H \) \(([s_1]_i^+) \subseteq \Phi \), then \([t_1]_i \in \Phi \). By definition of \( \Phi \), there is a state \( t_2 \) such that \([t_2]_i \) is in \( H \) \(([s_2]_i^+) \) and \(([t_1]_i, [t_2]_i) \in H_i\). \([t_2]_i \) is in \( H \) \(([s_2]_i^+) \) implies that \( t_3 \) is a successor of some state \( s_3 \) in \([s_2]_i^+ \).

Since \( s_1, s_2 \) are in the same class in \( \Sigma_{i+1} \), by Lemma 24 \( s_1 \) and \( s_2 \) are \((i + 1)\)-equivalent. Thus, there exists a successor \( t_2 \) of \( s_2 \) such that \([t_2], [t_2] \in H_i\). By Invariant 3, \( H_i \) is transitive and therefore \(([t_1]_i, [t_2]_i) \in H_i\).

**Corollary 27.** If \(([s_1]_i^+, [s_2]_i^+) \in H_{i+1} \) then \( s_1 \leq_{i+1} s_2 \).

**Lemma 28.** If \( s_1 \leq_{i+1} s_2 \) then \( s_1 \leq_i s_2 \).

**Proof:** First, \( s_1 \leq_{i+1} s_2 \) implies \( L(s_1) = L(s_2) \). Next, we distinguish between two cases:

1. \( i = 0 \), then \( L(s_1) = L(s_2) \) implies \( s_1 \leq_0 s_2 \).
2. Suppose \( i > 0 \). We will show that for every successor \( t_1 \) of \( s_1 \), there exists a successor \( t_2 \) of \( s_2 \) such that \([t_1]_i \leq [t_2]_i \).
   
   Let \( t_1 \) be a successor of \( s_1 \), then \( s_1 \leq_{i+1} s_2 \) implies that there exists a successor \( t_2 \) of \( s_2 \) such that \(([t_1], [t_2]) \in H_i\). Let \([t_1]_i = ([t_1]_i)_{pre} \) and \([t_2]_i = ([t_2]_i)_{pre} \). Then by Proposition 22 \(([t_1]_i \leq [t_2]_i) \in H_{i-1} \), as required.
Lemma 29. If $s_1 \leq_{i+1} s_2$, then $([s_1]_{i+1}^+, [s_2]_{i+1}^+) \in H_{i+1}$.

Proof: Assume $s_1 \leq_{i+1} s_2$.

- By Lemma 28 $s_1 \leq_2 s_2$.
- By induction hypothesis, Invariant 2 holds for $i$. Thus, $([s_1]_i^+, [s_2]_i^+) \in H_i$.
- Clearly, $[s_1]_{i+1}^+ \subseteq [s_1]_i^+$ and $[s_2]_{i+1}^+ \subseteq [s_2]_i^+$. Since $([s_1]_i^+, [s_2]_i^+) \in H_i$, the pair $([s_1]_{i+1}^+, [s_2]_{i+1}^+)$ is considered for inclusion in $H_{i+1}$ in the $i+1$th update step of the algorithm.
- In order to prove that $([s_1]_{i+1}^+, [s_2]_{i+1}^+) \in H_{i+1}$, we show that $H([s_1]_{i+1}^+) \subseteq \Phi$, i.e., every class $\alpha$ in $H([s_1]_{i+1}^+)$ is also in $\Phi$.
- Let $\alpha \in \Sigma_i$ be a class in $H([s_1]_{i+1}^+)$. Then there exists a state $s_3 \in [s_1]_{i+1}^+$ and a successor $t_3$ of $s_3$ such that $t_3 \in \alpha$.
- By Lemma 24, $s_1, s_3$ being in the same class of $\Sigma_{i+1}$ implies that, there exists a successor $t_1$ of $s_1$ such that $(\alpha, [t_1]_i^+) \in H_i$.
- Since $s_1 \leq_{i+1} s_2$, then there exists a successor $t_2$ of $s_2$ such that $([t_1]_i^+, [t_2]_i^+) \in H_i$.
- Since $H_i$ is transitive, $(\alpha, [t_2]_i^+) \in H_i$.
- The definition of $H([s_2]_{i+1}^+)$ implies that $[t_2]_i^+ \in H([s_2]_{i+1}^+)$. Hence, $(\alpha, [t_2]_i^+) \in H_i$ implies that $\alpha \in \Phi$.

Corollary 27 and Lemma 29 prove Invariant 2.

5.5 Proving Invariant 3

Lemma 30. $H_{i+1}$ is transitive.

Proof: Let $\alpha_1, \alpha_2, \alpha_3$ be classes in $\Sigma_{i+1}$ such that $(\alpha_1, \alpha_2) \in H_{i+1}$ and $(\alpha_2, \alpha_3) \in H_{i+1}$. We prove that $(\alpha_1, \alpha_3) \in H_{i+1}$. To do so, we show that for all states $s_1, s_3$ in $\alpha_1, \alpha_3$ respectively, the following holds. For every successor $t_1$ of $s_1$, there exists a successor $t_2$ of $s_3$ such that $([t_1]_i^+, [t_2]_i^+) \in H_i$. By Lemma 29 this implies $(\alpha_1, \alpha_3) \in H_{i+1}$.

Let $s_1, s_2, s_3$ be states in $\alpha_1, \alpha_2, \alpha_3$ respectively, and let $t_1$ be a successor of $s_1$. By Lemma 26, $(\alpha_1, \alpha_2) \in H_{i+1}$ implies that there exist a successor $t_2$ of $s_3$ such that $([t_1]_i^+, [t_2]_i^+) \in H_i$. By Lemma 26, $(\alpha_2, \alpha_3) \in H_{i+1}$ implies that there exists a successor $t_3$ of $s_3$ such that $([t_2]_i^+, [t_3]_i^+) \in H_i$. By the induction hypothesis, $([t_1]_i^+, [t_3]_i^+) \in H_i$. Thus, we conclude that $(\alpha_1, \alpha_3) \in H_{i+1}$.

This above lemma proves Invariant 3. This completes the proof of the three invariants.

5.6 Equivalence Classes

In this section we will show that when the algorithm terminates after $k$ iterations, $\leq_k$ is the maximal simulation relation over $M \times M$ and $\Sigma_k$ is the set of equivalence classes with respect to simulation equivalence over $M \times M$. Moreover, $H_k$ is the maximal simulation relation over the corresponding quotient structure $M_q$.
Lemma 31. For every $i \geq 0$ and every state $s$, $s \leq_i s$.

Proof We will prove it by induction on $i$:

- Base: For $i = 0$, $L(s) = L(s)$ implies $s \leq_0 s$.
- Induction step: Assume that the lemma holds for $i$. Let $t$ be a successor of $s$. The induction hypothesis implies that $t \leq_i t$. Based on Invariant 2 we then have, $([t], [t]) \in H_i$. Thus, for every successor $t$ of $s$, we choose $t$ as the successor of $s$ such that $([t], [t]) \in H_i$. By the definition of $\leq_{i+1}$, this implies $s \leq_{i+1} s$.

Proposition 32. When the algorithm terminates, $\leq_k = \leq_{k-1}$.

Lemma 33. $\leq_k$ is a simulation over $M \times M$.

Proof:

- By Lemma 31, $s_0 \leq_k s_0$.
- $(s_1, s_2) \in \leq_k$ implies $L(s_1) = L(s_2)$.
- $(s_1, s_2) \in \leq_k$ implies that for every successor $t_1$ of $s_1$ there exists a successor $t_2$ of $s_2$ such that $([t_1], [t_2]) \in H_k$. By Corollary 27 $t_1 \leq_{k-1} t_2$, thus, since $\leq_k = \leq_{k-1}$, $t_1 \leq_k t_2$.

Lemma 34. $\leq_k$ is the maximal simulation over $M \times M$.

Proof Let $H'$ be the maximal simulation over $M \times M$. We prove that $H' \subset \leq_k$. By Invariant 2 it is sufficient to prove that $(s_1, s_2) \in H'$ implies $([s_1], [s_2]) \in H_k$.

- Base: $(s_1, s_2) \in H'$ implies $L(s_1) = L(s_2)$. Therefore, $[s_1]^0 = [s_2]^0$ and $([s_1]^0, [s_2]^0) \in H_k$.
- Induction step: Assume that the lemma holds for $i - 1$. Let $(s_1, s_2)$ be in $H'$. Then for every successor $t_1$ of $s_1$ there exists a successor $t_2$ of $s_2$ such that $([t_1], [t_2]) \in H_k$. By the inductive hypothesis, $([t_1], [t_2]) \in H_{k-1}$ which by Lemma 29 implies that $([s_1], [s_2]) \in H_i$.

Theorem 35. When the algorithm terminates $\Sigma_k$ is the set of equivalence classes of the simulation equivalence relation.

Proof: States $s_1, s_2$ are simulation equivalent iff $(s_1, s_2) \in \leq_k$ and $(s_2, s_1) \in \leq_k$ iff $s_1, s_2$ are $k$-equivalent iff [by Invariant 1] $s_1, s_2$ are in the same class in $\Sigma_k$.

We proved that $\Sigma_k$ is the set of equivalence classes which are used as the set of states $S_0$ in the quotient structure $M_0$. Next, we show that $H_k$ is the maximal simulation relation over $M_0 \times M_0$. 

21
Lemma 36. $H_k$ is a simulation over $M_q \times M_q$.

Proof:
- By Invariant 2, $(s_0, s_0) \in \leq_k$ implies $([s_0], [s_0]) \in H_k$.
- Assume, $([s_1], [s_2]) \in H_k$. By Invariant 2, $(s_1, s_2) \in \leq_k$, thus $L(s_1) = L(s_2)$ which implies that $L_q([s_1]) = L_q([s_2])$.
- Let $([s_1], [s_2])$ be a pair in $H_k$ and $a$ a successor of $[s_1]$. By the definition of $R_i$ there exists a successor $t_1$ of $s_1$ in $a$. Since $\leq_k$ is a simulation relation, there is a successor $t_2$ of $s_2$ such that $(t_1, t_2) \leq_k$. Let $t_m$ be a maximal state in $G([s_2], t_2)$ (Definition 13). By Lemma 15 $[t_m]$ is a successor of $[s_2]$, $t_m$ is maximal in $G([s_2], t_2)$, hence $(t_2, t_m) \leq_k$. Since $\leq_k$ is transitive, $(t_1, t_m) \leq_k$. Thus, by Invariant 2, $(a, [t_m]) \in H_k$.

Theorem 37. $H_k$ is the maximal simulation relation over $M_q \times M_q$.

Proof: Let $H'$ be the maximal simulation relation over $M_q \times M_q$. We prove that the relation defined by $H = \{(s_1, s_2) | ([s_1], [s_2]) \in H' \}$ is the maximal simulation relation over $M \times M$. Thus, $H = \leq_k$. By Invariant 2, $\leq_k = \{(s_1, s_2) | ([s_1], [s_2]) \in H_k \}$, hence $H_k = H'$.

- By $H_k \subseteq H'$ we have $\leq_k \subseteq H$. Since $H$ includes the maximal simulation relation $\leq_k$, it is sufficient to show that $H$ is a simulation relation.
- By transitivity of $H'$, $H$ is transitive.
- Since, $([s_1], [s_2]) \in H'$, $(s_0, s_0) \in H$.
- Assume $(s_1, s_2) \in H$, then $L(s_1) = L_q([s_1]) = L_q([s_2]) = L(s_2)$.
- Suppose $(s_1, s_2) \in H$ and $t_1$ is a successor of $s_1$. Let $t_m$ be a maximal state in $G([s_1], t_1)$ (Definition 13). By Lemma 15 $[t_m]$ is a successor of $[s_1]$, $t_m$ is maximal in $G([s_1], t_1)$, hence $(t_1, t_m) \leq_k$. Because $\leq_k \subseteq H$, $(t_1, t_m) \in H$. Since, $H'$ is a simulation relation, there is a successor $a$ of $[s_2]$ such that $([t_m], a) \in H'$. By the definition of $R_i$ there exists a successor $t_2$ of $s_2$ in $a$. It follows from $([t_m], a) \in H'$ that $(t_m, t_2) \in H$ and by transitivity of $H$, $(t_1, t_2) \in H$. \qed

5.7 Space Complexity

The space complexity of the Partitioning Algorithm depends on the size of $\Sigma_i$. We assume that the algorithm applied to Kripke structures with some redundancy, thus $|\Sigma_i| << |S|$.

We measure the space complexity with respect to the size of the three following relations:

1. The relation $R$.
2. The relations $H_i$ whose size depends on $\Sigma_i$. We can bound the size of $H_i$ by $|\Sigma_i|^2$.
3. A relation that relates each state to its equivalence class. Since every state belongs to a single class, the size of this relation is $O(|S| \cdot log(|\Sigma_i|))$. 

22
In the $i$th iteration we do not need to keep all $H_0, H_1, \ldots$ and $\Sigma_0, \Sigma_1, \ldots$, since we only refer to $H_i, H_{i+1}$ and $\Sigma_i, \Sigma_{i+1}$. By the above we conclude that the total space complexity is $O(|R| + |\Sigma_k|^2 + |S| \cdot \log(|\Sigma_k|))$.

In practice, we often do not hold the transition relation $R$ in the memory. Rather we use it to provide, whenever needed, the set of successors of a given state. Thus, the space complexity is $O(|\Sigma_k|^2 + |S| \cdot \log(|\Sigma_k|))$. Recall that the space complexity of the naive algorithm for computing the equivalence classes of the simulation equivalence relation is bounded by $|S|^2$, which is the size of the simulation relation over $M \times M$. In case $|\Sigma_k| << |S|$, the Partitioning Algorithm achieve a much better space complexity.

5.8 Time Complexity

As we already mentioned, the algorithm runs at most $|S|^2$ iterations. In every iteration it performs one refine and one update. refine can be done in $O(|\Sigma_k|^3 + |\Sigma_k| \cdot |R|)$ and update can be done in $O(|\Sigma_k|^2 \cdot (|\Sigma_k|^2 + |R|))$. Thus the total time complexity is $O(|S|^2 \cdot |\Sigma|^2 \cdot (|\Sigma_k|^2 + |R|))$.

References


