References

[ASSB94] A. Aziz, V. Singhal, G.M. Swamy, and R.K. Brayton. Minimizing interacting finite state machines: A compositional approach to language containment. In *In Proc. of Intl. Conf. on Computer Design*, 1994.

[ASSSV94] A. Aziz, V. Singhal, T.R. Shiple, and A.L. Sangiovanni-Vincentelli. Formula-dependent equivalence for compositional ctl model checking. In *In Proc. of Conference on Computer-Aided Verification*, 1994.

[BdS92] Amar Bouali and Robert de Simone. Symbolic bisimulation minimisation. In G. V. Bochmann and D. K. Probst, editors, *Proceedings of the 4th Conference on Computer-Aided Verification*, volume 663 of *LNCS*, pages 96–108. Springer Verlag, July 1992.

[BFH90] A. Bouajjani, J.-C. Fernandez, and N. Halbwachs. Minimal model generation. In E.M Clarke and R.P. Kurshan, editors, *Computer-Aided Verification*, pages 197–203, New York, June 1990. Springer-Verlag.

[CE81] E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In *Logic of Programs: Workshop, Yorktown Heights, NY, May 1981*, volume 131 of *lncs*. sv, 1981.

[CGP99] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT press, December 1999.

[FV98] K. Fisler and M. Vardi. Bisimulation minimization in an automata-theoretic verification framework. In *CAV 98 papers*, 1998.

[GL94] O. Grumberg and D.E. Long. Model checking and modular verification. *ACM Trans. on Programming Languages and Systems*, 16(3):843–871, 1994.

[HHK95] M.R. Henzinger, T.A. Henzinger, and P.W. Kopke. Computing simulation on finite and infinite grafs. In *Proc. Symp. Foundations of Computer Science*, pages 453–462, 1995.

[LY92] D. Lee and M. Yannakakis. Online minimization of transition systems. In *STOC 92 papers*, 1992.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *In proceedings of the 2nd International Joint Conference on Artificial Intelligence*, pages 481–489, September 1971.

[Par81] D. Park. Concurrency and automata on infinite sequences. In *5th GI-Conference on Theoretical Computer Science*, 1981.

[PB96] R. Paige and B. Bloom. Transformational design and implementation of new efficient solution to the ready simulation problem. In *Science of Computer Programming*, volume 24, pages 189–220, 1996.

Proof Sketch of Invariant 3**:** We show that if $([s_1],[s_2]) \in H_{i+1}$ and $([s_2],[s_3]) \in H_{i+1}$ then for every successor t_1 of s_1 there is a successor t_3 of s_3 such that $([t_1],[t_3]) \in H_i$. Thus, by Invariant 2 $([s_1],[s_3]) \in H_{i+1}$.

5.3 Equivalence Classes

In this section we show that when the algorithm terminates after k iterations, $\leq_k$ is the maximal simulation relation over $M \times M$ and Σ_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.

Theorem 20. *When the algorithm terminates, $\leq_k$ is the maximal simulation over $M \times M$ and Σ_k is the set of equivalence classes of the simulation equivalence relation.*

Invariant 2 and $\leq_k = \leq_{k_1}$ imply that $\leq_k$ is a simulation relation. For maximality of $\leq_k$ we prove by induction on i that if $([s_1],[s_2]) \in H_i$ then (s_1, s_2) is in the maximal simulation over $M \times M$.

Theorem 21. *H_k is the maximal simulation relation over $M_q \times M_q$.*

5.4 Space Complexity

The space complexity of the Partitioning Algorithm depends on the size of Σ_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 Σ_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 $O(|S| \cdot log(|\Sigma_i|))$.

In the ith 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 Σ_i, Σ_{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.

5.5 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 $O(|S|^2 \cdot |\Sigma_k|^2 \cdot (|\Sigma_k|^2 + |R|))$.

where $\alpha_1 = \{0\}, \alpha_2 = \{1,2\}, \beta_1 = \{3\}, \beta_2 = \{4\}, \beta_3 = \{5\}, \gamma_1 = \{6\}, \gamma_2 = \{7\}, \delta_0 = \{8,9\}$.

– The third iteration results in the relations:
$\Sigma_3 = \Sigma_2, H_3 = H_2$ - *change* $= false$.
The equivalence classes are:
$\alpha_1 = \{0\}, \alpha_2 = \{1,2\}, \beta_1 = \{3\}, \beta_2 = \{4\}, \beta_3 = \{5\}, \gamma_1 = \{6\}, \gamma_2 = \{7\}, \delta_0 = \{8,9\}$

Since the third iteration results in no change to the computed partition or ordering relation, the algorithm terminates. Σ_2 is the final set of equivalence classes which constitutes the set S_q of states of M_q. H_2 is the maximal simulation relation over $M_q \times M_q$.

5.2 The Correctness of the Partitioning Algorithm

In order to prove the correctness of the Partitioning Algorithm , we prove first the three invariants mentioned before. We will prove these invariants by induction on i. The base case $(i = 0)$ for all three invariants follows from the definitions of the initial relations Σ_0 and H_0. We assume that for every $j \leq i$, the invariants hold for j. We prove that the invariants hold for $i + 1$.

Theorem 18. *1. For all states $s_1, s_2 \in S$, s_1 and s_2 are in the same class in Σ_{i+1} iff s_1 and s_2 are $i+1-$equivalent.*
2. $([s_1]^{i+1}, [s_2]^{i+1}) \in H_{i+1}$ iff $s_1 \leq_{i+1} s_2$.
3. H_{i+1} is transitive.

Proof Sketch of Invariant 1: We sketch here only the proof for the second direction. We prove that $s_2 \in [s_1]$. Since $s_1 \in [s_1]$, $s_1 \in GT$ and $s_1 \in LT$. Based on that and on the transitivity of H_i (by induction hypothesis for Invariant 3), we show that $s_2 \in GT \cap LT$ and therefore $s_2 \in [s_1]$.

Proof Sketch of Invariant 2: We next sketch the proof of Invariant 2 for H_{i+1}. Since the construction of H_{i+1} is based on both Σ_i and Σ_{i+1}, we need to distinguish between classes in these sets. We use $[s]^i$ and $[s]^{i+1}$ to denote equivalence classes in Σ_i and Σ_{i+1} respectively.

The following lemma implies the first direction of the invariant. The proof of the second direction uses similar arguments.

Lemma 19. *Let $([s_1]^{i+1}, [s_2]^{i+1}) \in H_{i+1}$. Then for every successor t_1 of s_1, there exists a successor t_2 of s_2 such that $([t_1]^i, [t_2]^i) \in H_i$.*

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

Since s_2, s_3 are in the same class in Σ_{i+1}, by invariant 1 s_2 and s_3 are $(i+1)-$equivalent. Thus, there exists a successor t_2 of s_2 such that $([t_3]^i, [t_2]^i) \in H_i$. By Invariant 3, H_i is transitive and therefore $([t_1]^i, [t_2]^i) \in H_i$. $\square$

H_i it is sufficient to check $(\alpha'_1, \alpha'_2) \in H_i$ only in case $\alpha_2 \supseteq \alpha'_2$, $\alpha_1 \supseteq \alpha'_1$, and $(\alpha_1, \alpha_2) \in H_{i-1}$.

For suitable α'_1 and α'_2, we first construct the set Φ of classes that are "smaller" than the classes in $\Pi(\alpha'_2)$. By checking if $\Phi \supseteq \Pi(\alpha'_1)$ we determine whether every class in $\Pi(\alpha'_1)$ is "smaller" than some class in $\Pi(\alpha'_2)$, in which case (α'_1, α'_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_q.

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

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 $| \leq_0 |$, 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, Σ_i will not change but H_i will, and in a later iteration $j > i$, Σ_j will change again.

Example: In this example we show how the Partitioning Algorithm is applied to the Kripke structure presented in Figure 5 .

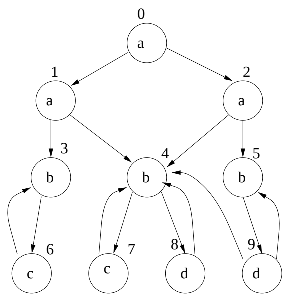

Fig. 5. An example structure

- We initialize the algorithm as follows:
  $\Sigma_0 = \{\alpha_0, \beta_0, \gamma_0, \delta_0\}$, $H_0 = \{(\alpha_0, \alpha_0), (\beta_0, \beta_0), (\gamma_0, \gamma_0), (\delta_0, \delta_0)\}$,
  where $\alpha_0 = \{0, 1, 2\}, \beta_0 = \{3, 4, 5\}, \gamma_0 = \{6, 7\}, \delta_0 = \{8, 9\}$.
- The first iteration results in the relations:
  $\Sigma_1 = \{\alpha_1, \alpha_2, \beta_1, \beta_2, \beta_3, \gamma_0, \delta_0\}$,
  $H_1 = \{(\alpha_1, \alpha_1), (\alpha_2, \alpha_2), (\beta_1, \beta_1), (\beta_2, \beta_2), (\beta_3, \beta_3), (\beta_1, \beta_2), (\beta_3, \beta_2), (\gamma_0, \gamma_0), (\delta_0, \delta_0)\}$
  where $\alpha_1 = \{0\}, \alpha_2 = \{1, 2\}, \beta_1 = \{3\}, \beta_2 = \{4\}, \beta_3 = \{5\}, \gamma_0 = \{6, 7\}, \delta_0 = \{8, 9\}$.
- The second iteration results in the relations:
  $\Sigma_2 = \{\alpha_1, \alpha_2, \beta_1, \beta_2, \beta_3, \gamma_1, \gamma_2, \delta_0\}$,
  $H_2 = \{(\alpha_1, \alpha_1), (\alpha_2, \alpha_2), (\beta_1, \beta_1), (\beta_2, \beta_2), (\beta_3, \beta_3),$
  $(\beta_1, \beta_2), (\beta_3, \beta_2), (\gamma_1, \gamma_1), (\gamma_2, \gamma_2), (\gamma_1, \gamma_2), (\delta_0, \delta_0)\}$,

```
Initialize the algorithm:
    change := true
    for each label $ap \in 2^{AP}$ construct $\alpha_{ap} \in \Sigma_0$ such that $s \in \alpha_{ap} \Leftrightarrow L(s) = ap$.
    $H_0 = \{(\alpha, \alpha) | \alpha \in \Sigma_0\}$
    while $change = true$ do begin
        $change := false$
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 $s_p \in \alpha$
                $GT := \{s_g | s_g \in \alpha \wedge \forall t_p \in succ(s_p)\ \exists t_g \in succ(s_g).\ ([t_p], [t_g]) \in H_i\}$
                $LT := \{s_l | s_l \in \alpha \wedge \forall t_l \in succ(s_l)\ \exists t_p \in succ(s_p).\ ([t_l], [t_p]) \in H_i\}$
                $\alpha' := GT \cap LT$
                if $\alpha \neq \alpha'$ then $change := true$
                $\alpha := \alpha \setminus \alpha'$
                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
                $\Phi = \{\phi | \exists \xi \in \Pi(\alpha_2')\ (\phi, \xi) \in H_i\}$
                if $\Phi \supseteq \Pi(\alpha_1')$ then
                    insert $(\alpha_1', \alpha_2')$ to $H_{i+1}$
                else
                    change := true
            end
        end
    end
```

Fig. 4. The Partitioning Algorithm

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.

Σ_i is a set of equivalence classes with respect to the $i-$equivalence relation. In the ith iteration we split the equivalence classes of Σ_{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 α that are $i-$equivalent to s_p. These states form an $i-$equivalence class α' that is inserted to Σ_i.

α' is constructed in two steps. First we calculate the set of states $GT \subseteq \alpha$ that contains all states s_g such that $s_p \leq_i s_g$. 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 α that are $i-$equivalent to s_p.

H_i captures the partial order $\leq_i$, i.e., $s_1 \leq_i s_2$ iff $([s_1], [s_2]) \in H_i$. Note 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

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

Given a structure M, we would like to build the equivalence classes of the simulation equivalence relation, without first calculating H_M. Our algorithm, called the *Partitioning Algorithm* , starts with a *partition* Σ_0 of S to classes. The classes in Σ_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 Σ_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 (Σ_{i-1} and Σ_i) and the previous ordering relation (H_{i-1}). Initially, H_0 includes only the identity pairs (of classes).

In the algorithm, we use *succ(s)* for the set of successors of s. Whenever Σ_i is clear from the context, $[s]$ is used for the equivalence class of s. We also use a function Π that associates with each class $\alpha \in \Sigma_i$ the set of classes $\alpha' \in \Sigma_{i-1}$ that contain a successor of some state in α.

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

We use English letters to denote states, capital English letters to denote sets of states, Greek letters to denote equivalence classes, and capital Greek letters to denote sets of equivalence classes. The Partitioning Algorithm is presented in Figure 4 .

Definition 17. *The partial order $\leq_i$ on S is defined by: $s_1 \leq_i s_2$ implies, $L(s_1) = L(s_2)$ and if $i > 0$, $\forall t_1[(s_1,t_1) \in R \rightarrow \exists t_2[(s_2,t_2) \in R \wedge ([t_1],[t_2]) \in H_{i-1}]]$. In case $i = 0$, $s_1 \leq_0 s_2$ iff $L(s_1) = L(s_2)$.*
*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 three invariants 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.

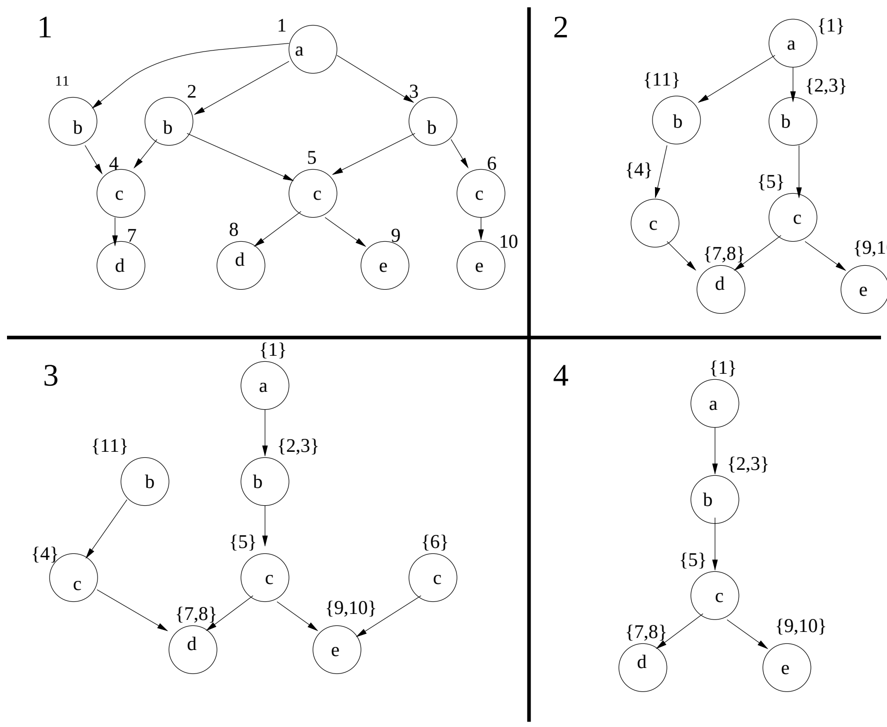

Fig. 3. An example of the Minimizing Algorithm

2. Part 2 presents the $\forall-$structure M_q. The maximal simulation relation H_M is (not including the trivial pairs):
   $H_M = \{(\{11\}, \{2,3\}), (\{4\}, \{5\}), (\{6\}, \{5\})\}$.
3. $\{11\}$ is a little brother of $\{2,3\}$ and $\{1\}$ is their father. Part 3 presents the structure after the removal of the edge $(\{1\}, \{11\})$.
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 naive 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 $O(|S|^3)$.

We proved that the result M' of the Disconnecting Algorithm is simulation equivalent to the original structure M. Note that M' has the same set of states as M. We now show that the maximal simulation relation over M is identical to the maximal simulation relations for all intermediate structures M'' (including M'), computed by the Disconnecting Algorithm. Since there are no simulation equivalent states in M, there are no such states in M' as well.

Lemma 16. *Let $M' =< S, R', s_0, L >$ be the result of the Disconnecting Algorithm on M and let $H' \subseteq S' \times S'$ be the maximal simulation over $M' \times M'$. Then, $H_M = H'$.*

The lemma is proved by induction on the number of iterations.

As a result of the last lemma, 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:

$$R' = R - \{(s_1, s_2) | \exists s_3 : (s_1, s_3) \in R \wedge (s_2, s_3) \in H_M \wedge (s_3, s_2) \notin H_M\}.$$

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 $H_M$ over $M_q \times M_q$.
2. $R' = R_q - \{(s_1, s_2) | \exists s_3 : (s_1, s_3) \in R_q \wedge (s_2, s_3) \in H_M\}$
3. Remove all unreachable states.
```

Fig. 2. The Minimizing Algorithm

Note that, in the second step we eliminate the check $(s_3, s_2) \notin H_M$. 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 8 is correct.

Figure 3 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 : $\{\{1\}, \{2,3\}, \{11\}, \{4\}, \{5\}, \{6\}, \{7,8\}, \{9,10\}\}$.

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 M' are simulation equivalent.
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 1 we present an iterative algorithm which disconnects little brothers and results in M'.

```
change := true
while (change = true) do
    Compute the maximal simulation relation $H_M$
    change := false
    If there are $s_1, s_2, s_3 \in S$ 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. 1. The Disconnecting Algorithm.

Since in each iteration of the algorithm one edge 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 15. *Let $M' =< S', R', s'_0, L' >$ be the result of the Disconnecting Algorithm on M. Then M and M' are simulation equivalent.*

Proof Sketch : 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'' be the result of the first i iterations and H'' be the maximal simulation over $M'' \times M''$. Let M' be the result of the $(i+1)$th iteration where $R' = R'' \setminus \{(s''_1, s''_2)\}$. Assume that M and M'' are simulation equivalent. It is straight forward to see that $H' = \{(s'_1, s''_2) | (s''_1, s''_2) \in H''\}$ is a simulation relation over $M' \times M''$. Thus, $M' \preceq M''$.

To show that $M'' \preceq M'$ we prove that $H' = \{(s''_1, s'_2) | (s''_1, s''_2) \in H''\}$ is a simulation relation. Clearly, $(s''_0, s'_0) \in H'$ and $L''(s''_1) = L'(s'_2)$.

Suppose $(s''_1, s'_2) \in H'$ and t''_1 is a successor of s''_1. Since H'' is a simulation relation, there exists a successor t''_2 of s''_2 such that $(t''_1, t''_2) \in H''$. This implies that $(t''_1, t'_2) \in H'$. If $(s'_2, t'_2) \in R'$ then we are done. Otherwise, (s''_2, t''_2) is removed from R'' because t''_2 is a little brother of some successor t''_3 of s''_2. Since (s''_2, t''_2) is the only edge removed at the $(i+1)$th iteration, $(s'_2, t'_3) \in R'$. Because t''_2 is a little brother of t''_3 then $(t''_2, t''_3) \in H''$. By transitivity of the simulation relation, $(t''_1, t''_3) \in H''$, thus $(t''_1, t'_3) \in H'$. □

The transitions in M_q are $\forall$-transitions, in which there is a transition between two equivalence classes iff *every* state of the one has a successor in the other. We could also define $\exists$-transitions, in which there is a transition between classes if there exists a state in one with a successor in the other. Both definitions result in a simulation equivalent structure. However, the former has smaller transition relation and therefore it is preferable.

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 11. *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 $(s_m, s) \in H_M$ and $(s, s_m) \notin H_M$.*

Definition 12. *Let α be a state of M_q, s_1 a state in α and t_1 a successor of s_1. The set $G(\alpha, t_1)$ is defined as follow:*

$$G(\alpha, t_1) = \{t_2 \in S | \exists s_2 \in \alpha \wedge (s_2, t_2) \in R \wedge (t_1, t_2) \in H_M\}.$$

Intuitively, $G(\alpha, t_1)$ is the set of states that are greater than t_1 and are successors of states in α. Notice that since all state in α are simulation equivalent, every state in α has at least one successor in $G(\alpha, t_1)$.

Lemma 13. *Let α, t_1 be as defined in Definition 12 . Then for every maximal states t_m in $G(\alpha, t_1)$, $[t_m]$ is a successor of α.*

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 α.

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

Theorem 14. *The structures M and M_q are simulation equivalent.*

Proof Sketch : It is straight forward to show that $H' = \{(\alpha, s) | s \in \alpha\}$ is a simulation relation over $M_q \times M$. Thus, $M_q \preceq M$.

In order to prove that $M \preceq M_q$ we choose $H' = \{(s_1, \alpha) |$ there exists a state $s_2 \in \alpha$ such that $(s_1, s_2) \in H_M\}$. Clearly, $(s_0, s_{0_q}) \in H'$ and $L(s_1) = L_q(\alpha)$.

Assume $(s_1, \alpha_1) \in H'$ and let t_1 be a successor of s_1. We prove that there exists a successor α_2 of α_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 13 implies that $(\alpha_1, [t_m]) \in R_q$. Since t_m is maximal in $G(\alpha_1, t_1)$, $(t_1, t_m) \in H_M$ 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 $(s_1, s_2) \in H_M$. Since $(s_1, s_2) \in H_M$ there is a successor t_2 of s_2 such that $(t_1, t_2) \in H_M$. The first case implies that there exists an equivalence class α_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 $(t_2, t_3) \in H_M$. By transitivity of simulation $(t_1, t_3) \in H_M$. Thus, $(t_1, \alpha_2) \in H'$. $\Box$

initial state. (since all states are reachable, the distance is bounded by $|S|$). Again we use the composed relation $H_{MM'M}$ to show that if f is not onto then M' is not reduced.

Similarly, we can show that f^{-1} is onto and therefore f is total. $\Box$

Lemma 7. *For all $s' \in S'$, $L'(s') = L(f(s'))$. Furthermore, for all $s'_1, s'_2 \in S'$, $(s'_1, s'_2) \in R'$ iff $(f(s'_1), f(s'_2)) \in R$.*

Thus, we conclude Theorem 4 .

Theorem 8. *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 8 , we present in the next sections an algorithm that receives a Kripke structure M and computes a reduce Kripke structure M', which is simulation equivalent to $|M|$, such that $|M'| \leq |M|$. Moreover, if M is not reduced then $|M'| < |M|$.

Lemma 9. *Let M' be a reduced Kripke structure. For every M that is simulation equivalent to $|M'|$, if M and M' are not isomorphic then $|M'| < |M|$.*

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 model 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 $\forall-$*quotient structure* with respect to the simulation equivalence relation. We fix M to be the original Kripke structure. We denote by $[s]$ the equivalence class which includes s.

Definition 10. *The $\forall-$quotient structure $M_q =< S_q, R_q, s_{0_q}, L_q >$ 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).*
- *$R_q = \{(\alpha_1, \alpha_2) | \forall s_1 \in \alpha_1 \ \exists s_2 \in \alpha_2 . \ (s_1, s_2) \in R\}$*
- *$s_{0_q} = [s_{0_q}]$.*
- *$L_q([s]) = L(s)$.*

structure always exists. Furthermore, we show that all reduced structures of M are *isomorphic* to each other.

Let M be a Kripke structure. The *maximal simulation relation* over $M \times M$ always exists and is denoted by H_M. We need the following two definitions in order to characterize reduced structures.

Two states $s_1, s_2 \in M$ are *simulation equivalent* iff $(s_1, s_2) \in H_M$ and $(s_2, s_1) \in H_M$.
A state s_1 is a *little brother* of a state s_2 iff there exists a state s_3 such that:

- $(s_3, s_2) \in R$ and $(s_3, s_1) \in R$.
- $(s_1, s_2) \in H_M$ and $(s_2, s_1) \notin H_M$.

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 straight forward. 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 $H_{MM'}$ and $H_{M'M}$ to denote the maximal simulation relations over $M \times M'$ and $M' \times M$ respectively. The *composed* relation $H_{MM'M} \subseteq S \times S$ is defined by $H_{MM'M} = \{(s_1, s_2) | \exists s' \in S'.\ (s_1, s') \in H_{MM'} \wedge (s', s_2) \in H_{M'M}\}$.

Lemma 5. *The composed relation $H_{MM'M}$ is a simulation relation.*

For the reduced Kripke structures M and M', we define the *matching relation* $f \subseteq S' \times S$ as follows:

$$(s', s) \in f \text{ iff } (s', s) \in H_{M'M} \text{ and } (s, s') \in H_{MM'}.$$

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.

Lemma 6. *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 Sketch : First we need to prove that f is a function from S' to S. We 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 $(s', s_2) \in f$. We show that $(s_1, s_2) \in H_{MM'M}$ and $(s_2, s_1) \in H_{MM'M}$. Since $H_{MM'M}$ is included in H_M, this contradicts the assumption that M is reduced. 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 $s \in S$ from the 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 $L : S \to 2^{AP}$ 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* [Mil71] 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

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

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). For lack of space the formal definition is omitted. It 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' iff $M \preceq M'$ 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 show that a reduced 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 respect 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 Σ_0 of the state space to classes whose states are equally labeled. It also initializes a preorder H_0 over the classes in Σ_0. At iteration $i+1$, Σ_{i+1} is constructed by splitting classes in Σ_i. The relation H_{i+1} is updated based on Σ_i, Σ_{i+1} and H_i.

When the algorithm terminates (after k iterations) Σ_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 $O(|\Sigma_k|^2+|S|\cdot log(|\Sigma_k|))$. We assume that in most cases $|\Sigma_k| << |S|$, thus this complexity is significantly smaller than that of the Minimizing Algorithm . Unfortunately, time complexity will probably become worse (depending on the size of Σ_k). It is bounded by $O(|S|^2 \cdot |\Sigma_k|^2 \cdot (|\Sigma_k|^2 + |R|))$. 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 Σ_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 [FV98] a bisimulation minimization is applied to the intersection of the system automaton and the specification automaton. The algorithm from [LY92] is used.

Several works minimize a structure in a compositional way, preserving language containment [ASSB94] or a given CTL formula [ASSSV94]. Minimizing with respect

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. Relations of this type 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.

In this paper we investigate the *simulation equivalence* relation that is weaker than bisimulation but stronger than the simulation preorder. This relation strongly preserves ACTL. It also strongly preserves ACTL*, which contains the linear-time temporal logic LTL. Both ACTL and LTL are widely used for model checking in practice.

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

Simulation Based Minimization

Doron Bustan and Orna Grumberg

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

Abstract. [1] 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] The full version of this paper including proofs of correctness can be found in http://www.cs.technion.ac.il/users/orna/publications.html