Modular Minimization of Finite State Machines

Doron Bustan and Orna Grumberg
Computer Science Department
Technion, Haifa 32000, Israel
email: {orna,doron2}@cs.technion.ac.il

Abstract

This work presents a modular technique for minimizing a finite-state machine (FSM) while preserving its equivalence to the original system. Being modular, the minimization technique should consume less time and space. Preserving equivalence, the resulting minimized model can be employed in both temporal logic model checking and sequential equivalence checking, thus reducing their time and space consumption.

Most systems have a natural modular structure, and we suggest using this structure in the minimization of their model. The complexity of minimizing a single module can be exponentially smaller than that of minimizing the entire system. (The actual reduction in complexity depends on the module connectivity to the other parts of the system.) Thus the method has great potential.

This modular algorithm has been implemented on an Intel system in Intel Haifa, and it has been tested on real hardware designs.

1 Introduction

The fast development of the hardware and software industry has increase the need for formal verification tools and techniques. Two widely used formal verification methods are temporal logic model checking and sequential equivalence checking. Both model checking and equivalence checking are fully automatic. However, they both suffer from the *state explosion problem*, that is, their space requirements are high and limit their applicability to large systems.

Many approaches for overcoming the state explosion problem have been suggested, including abstraction, partial order reduction, modular verification methods, and symmetry [6]. All are aimed at reducing the size of the

model to which formal verification methods are applied, thus extending their applicability to larger systems. When reduction methods are applied, the verification technique has to be able to deduce properties of the system by verifying the reduced model. We therefore require the result of the reduction to be *equivalent* to the original model.

Two of the most commonly used equivalence relations are *language equivalence* and *bisimulation* [18]. The former is suitable for equivalence checking as well as model checking for the linear-time logic LTL [19]. The latter is suitable for model checking of the expressive μ-calculus [12] logic and the widely used logics CTL [3, 7] and LTL.

Minimizing a model with respect to language equivalence is PSPACE-complete [21], while minimizing a model with respect to bisimulation is polynomial. Thus, bisimulation minimization appears to be more tractable. However, computing bisimulation minimization in a naive way may still be quite costly in terms of time and space [9]. This motivated the development of more refined reduction methods for a variety of equivalence relations. We describe some of these works below.

The algorithm in [14] minimizes models with respect to bisimulation. In order to improve efficiency, the algorithm refers only to reachable states and computes equivalence classes for bisimulation instead of pairs of equivalent states. This appears to consume less memory for BDD-based [4] implementations. In [8], the algorithm presented in [14] is applied to the intersection of the model with an automaton representing the property that should be satisfied by the model. In [5], a reduction with respect to symmetry equivalence is performed. The symmetry equivalence is a bisimulation equivalence, but not necessarily the maximal one. [5] reports that computing this reduction is more efficient in the BDD framework than reduction with respect to bisimulation.

Other works exploit modularity for reduction. The modular reduction in [1] preserves a given formula which should be checked for truth in the model. This method can result in a small model; however, since it preserves a single formula, it cannot be used for equivalence checking. In [2], the equivalence relation is a combination of language equivalence and fairness constraints. Since computing this relation is PSPACE-complete, an approximation equivalence relation is computed and the quotient model is defined with respect to it. [10] presents an algorithm that constructs an abstract model of a system through a sequence of approximations, where the final approximation is equivalent to the original system with respect to the specification language. The approximations are constructed according

to *interface specifications* given by the user. [20] suggests decomposing the model, reducing each module in separate and composing the result.

1.1 Modular minimization

In this paper we present a new modular minimization algorithm that improves the *naive modular algorithm*. The naive modular algorithm [20] is based on partitioning the system into components. It minimizes the model in iterations. In each iteration two components are selected and composed. Then the result is minimized. This process is repeated until all components are composed to form the full minimized system. The advantages of this approach are:

- Time and space requirements of minimization algorithms depend on the size of the model to which they are applied. By minimizing components instead of the full system, we expect a better overall complexity. Moreover, we will be able to minimize a system in parts even when the problem of minimizing the full system is intractable due to its size.
- It is sometimes impossible to complete the construction of the minimized system due to the size of intermediate components. In such cases, it might still be possible to apply some formal verification procedures to a partially minimized model, composed of minimized and unminimized components.

The *improved algorithm* we present improves each iteration in the naive algorithm. Given two components, the improved algorithm constructs the minimized model without ever constructing the non-minimized result of the composition. Thus the algorithm avoids the bottleneck of the naive algorithm. [13] presents a similar approach in which, before composing and reducing two components, each component is reduced with respect to the other. However, unlike the improved algorithm, this approach uses the full composition for the reduction of each component. Moreover, the result of composing the two reduced components needs to be further reduced. We present two versions of the improved algorithm. The first is for deterministic systems and the second is for nondeterministic ones. While the version for nondeterministic systems is more general, it has worse complexity. Since deterministic systems are widely used in the hardware industry, a special, more efficient version for these systems is worth developing.

The paper includes an implementation of the improved algorithm, carried out on an Intel verification platform at the sequential equivalence verification

CAD group of Intel design technology in Haifa. We tested our algorithm on real designs. The results imply that this method has real potential in making bisimulation minimization practical.

The rest of the paper is organized as follows: In Section 2 we define the model, model composition, and bisimulation equivalence. Section 3 presents some properties of bisimulation and modularity. Section 4 presents the modular minimization algorithm for deterministic and nondeterministic FSMs. Section 5 describes the implementation and the experimental results. Section 6 presents some conclusions.

2 Basic definitions

We model systems as finite-state machines (FSMs) in the form of *Moore machines* in which the states are labeled with outputs and the edges are labeled with inputs. Such machines are commonly used for modeling hardware designs.

Definition 2.1 *[16] An FSM is a tuple* $M =< S, S_0, I, O, L, R >$ *where*

- S *is a finite set of states.*
- $S_0 \subseteq S$ *is a set of initial states.*
- I *is a finite set of input propositions.*
- O *is a finite set of output propositions.*
- $I \cap O = \emptyset$.
- L *is a labeling function that maps each state to the set of output propositions true in that state.*
- $R \subseteq S \times 2^I \times S$ *is the transition relation. We assume that for every* $s \in S$ *and* $i \subseteq I$ *there exists at least one state* s' *such that* $(s, i, s') \in R$.

An FSM is *deterministic* iff for every state s and $i \subseteq I$ there exists exactly one state s' such that $(s, i, s') \in R$ and $|S_0| = 1$.

Two FSMs are composed only if their outputs are disjoint. There is a transition from a pair of states in the composed FSM if and only if the output of each state matches the input on the transition leaving the other state. This models the input-output connections between the two machines.

Definition 2.2 *Let* $M_1 =< S_1, S_{01}, I_1, O_1, L_1, R_1 >$ *and* $M_2 =< S_2, S_{02}, I_2, O_2, L_2, R_2 >$ *be two FSMs such that* $O_1 \cap O_2 = \emptyset$. *The composition* $M = M_1 || M_2 =< S, S_0, I, O, L, R >$ *is an FSM such that:*

- $S = S_1 \times S_2$.
- $S_0 = S_{01} \times S_{02}$.
- $I = (I_1 \setminus O_2) \cup (I_2 \setminus O_1)$.
- $O = O_1 \cup O_2$.
- $L((s_1, s_2)) = L_1(s_1) \cup L_2(s_2)$.
- $((s_1, s_2), i, (s_1', s_2')) \in R$ *iff* $(s_1, (i \cup L_2(s_2)) \cap I_1, s_1') \in R_1$ *and* $(s_2, (i \cup L_1(s_1)) \cap I_2, s_2') \in R_2$.

Lemma 2.3 *Let* M_1 *and* M_2 *be deterministic FSMs. Then the composition* M *of* M_1 *and* M_2 *is deterministic as well.*

Proof : Obviously, $|S_0| = 1$. Let (s_1, s_2) be a state in S and $i \subseteq I$ be an input. Let $i_1 = (i \cup L_2(s_2)) \cap I_1$ and $i_2 = (i \cup L_1(s_1)) \cap I_2$. Since M_1 is deterministic, there exists exactly one state s_1' such that $(s_1, i_1, s_1') \in R_1$. Similarly, there exists exactly one state s_2' such that $(s_2, i_2, s_2') \in R_2$. By the definition of composition, (s_1', s_2') is the only state such that $((s_1, s_2), i, (s_1', s_2')) \in R$. □

We now define the basic notion of equivalence that we use in this work, namely, *bisimulation.*

Definition 2.4 *Let* $M_1 =< S_1, S_{01}, I_1, O_1, L_1, R_1 >$ *and* $M_2 =< S_2, S_{02}, I_2, O_2, L_2, R_2 >$ *be two FSMs such that* $O_1 \cap O_2 \neq \emptyset$ *and* $I_1 = I_2$. *We say that* M_1 *and* M_2 *are bisimulation equivalent with respect to* $O' \subset O_1 \cap O_2$ *iff there exists a relation* $H \subseteq S_1 \times S_2$ *(called a bisimulation relation) such that:*

- *For every state* $s_{01} \in S_{01}$ *there exists a state* $s_{02} \in S_{02}$ *such that* $(s_{01}, s_{02}) \in H$ *and for every state* $s_{02} \in S_{02}$ *there exists a state* $s_{01} \in S_{01}$ *such that* $(s_{01}, s_{02}) \in H$.
- *For every pair* (s_1, s_2) *in* H *the following three conditions hold:*
  - $L_1(s_1) \cap O' = L_2(s_2) \cap O'$.

- *For every $i \subseteq I_1$ (recall that $I_1 = I_2$), and for every state s'_1 such that $(s_1, i, s'_1) \in R_1$, there exists a state s'_2 such that $(s_2, i, s'_2) \in R_2$ and $(s'_1, s'_2) \in H$.*
- *For every $i \subseteq I_2$, and for every state s'_2 such that $(s_2, i, s'_2) \in R_2$ there exists a state s'_1 such that $(s_1, i, s'_1) \in R_1$ and $(s'_1, s'_2) \in H$.*

Proposition 2.5 *For every FSM M, let s be a state in M that is not reachable from any initial state. The result of removing s from M is bisimulation equivalent to M.*

Consequently, we refer only to FSMs where all the states are reachable from the initial states.

Bisimulation is an equivalence relation over FSMs. [15] shows that for every two FSMs M_1 and M_2, there exists a maximal bisimulation relation that contains every relation satisfying the conditions of Definition 2.4. The maximal bisimulation relation $H \subseteq S \times S$ over the states of an FSM M is an equivalence relation over S. As such, it induces a partition of S to equivalence classes. These classes can be used to form the *quotient FSM* of M, which is the minimal FSM that is bisimulation equivalent to M.
We will denote by $[s]$ the equivalence class of a state s.

Definition 2.6 *Let $M =< S, S_0, I, O, L, R >$ be an FSM and let $H \subseteq S \times S$ be the maximal bisimulation relation with respect to $O' \subseteq O$ over M. The quotient FSM $M_Q =< S_Q, S_{0_Q}, I_Q, O_Q, L_Q, R_Q >$ of M with respect to H is defined as follows:*

- $S_Q = \{\alpha | \alpha$ *is an equivalence class in* $H\}$.
- $S_{0_Q} = \{[s_0] | s_0 \in S_0\}$.
- $I_Q = I$.
- $O_Q = O'$.
- *For $\alpha \in S_Q$, $L_Q(\alpha) = L(s) \cap O'$, for some (all) states $s \in \alpha$.*
- $R_Q = \{(\alpha, i, \alpha') |$ *there are states* $s \in \alpha, s' \in \alpha'$ *such that* $(s, i, s') \in R\}$.

Definition 2.7 *An FSM M is minimized iff it is isomorphic to its quotient FSM.*

3 Properties of modularity and reduction

The improved algorithm uses both modularity and bisimulation-based reduction. In the following we present some properties of the bisimulation relation, bisimulation reduction, and the relationships between bisimulation and modularity. The proofs for these claims are given in Appendix A.

Lemma 3.1 *Let M be an FSM, and let M_Q be the quotient FSM of M. Let (α, i, α') be an element in R_Q. Then for every state s in α there exists a state s' in α' such that $(s, i, s') \in R$.*

Proposition 3.2 *If M is deterministic, then M_Q is deterministic.*

Lemma 3.3 *M is minimized iff the maximal bisimulation relation over $M \times M$ contains exactly the identity pairs.*

Lemma 3.4 *Let M be an FSM and M_Q be the quotient FSM of M with respect to O'. Then M and M_Q are bisimulation equivalent with respect to O'.*

Lemma 3.5 *Let M be an FSM and M_Q be the quotient FSM of M with respect to O'. Then M_Q is the smallest (in number of states and transitions) FSM which is bisimulation equivalent to M with respect to O'.*

Proposition 3.6 *Let M_1 and M_2 be FSMs and let $H \subseteq S_1 \times S_2$ be a bisimulation relation over M_1 and M_2 with respect to $O \subseteq O_1 \cap O_2$. Then H is a bisimulation relation with respect to every $O' \subseteq O$.*

Lemma 3.7 *Let M_1 and M_2 be minimized FSMs. If $O_1 \cap I_2 = \emptyset$ and $O_2 \cap I_1 = \emptyset$, then $M = M_1||M_2$ is minimized.*

4 The improved algorithm

In this section we present the improved algorithm. Like the naive algorithm, the improved algorithm receives a design, given as a set of n components. The improved algorithm works in iterations. In each iteration two minimized components, M_1 and M_2, are selected and a new minimized component, which is equivalent to $M_1 \| M_2$, is constructed. The algorithm terminates when an iteration results in a single component. In this case, the final component is the smallest, in terms of states and transitions, that is equivalent to the composition of the n original components.

In this section we focus on a single iteration of the improved algorithm. Unlike the naive algorithm, where the two components are first composed and then the result is minimized, the improved algorithm constructs a minimized FSM that is equivalent to $M_1 \| M_2$ without constructing the full model. Thus, the improved algorithm require less time and space.

By Lemma 3.7, if M is the result of a composition of two different FSMs that do not interact with each other, then M can be minimized by minimizing M_1 and M_2 separately. However, this does not hold in the general case: given two minimized components M_1 and M_2, their composition $M_1 \| M_2$ is not necessarily minimized. This is demonstrated in Figure 1.

Figure 1 shows two FSMs, M_1 and M_2, for which $O_1 \cap I_2 \neq \emptyset$. M_1 and M_2 are minimized but their composition M is not. Figure 1 also contains M_Q, which is the result of minimizing M. The FSMs in Figure 1 are Moore machines, and we use the following convention in their description. The labels in the states represent the outputs of the Moore machines. The inputs are represented by a boolean formula on the edges. For states $s, s' \in S$ and $i \subseteq I$, (s, i, s') is an element in R iff i satisfies the formula on the edge from s to s'.

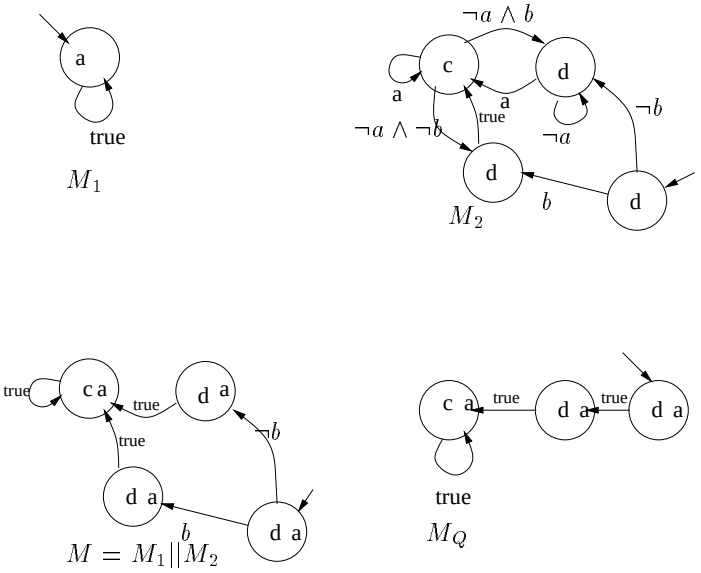

Figure 1: The composition of two minimized FSMs is not always minimized

The observation demonstrated in Figure 1 implies that a more sophisticated algorithm is needed for interacting components. We will present two

versions of the improved algorithm, one for deterministic FSMs and another for nondeterministic FSMs. While the former is less general, it has a better complexity. Since hardware designs are often modeled by a deterministic FSM, it worth developing a special algorithm for deterministic designs.

4.1 Deterministic FSMs

We now describe a single iteration of the improved algorithm. The version for deterministic FSMs and the version for nondeterministic FSMs differ only in the last stage of each iteration. We first present the version for deterministic systems, which is simpler, and then we present the change in the last stage for nondeterministic FSMs. In each iteration, the algorithm is given two minimized FSMs, M_1 and M_2, such that $O_1 \cap O_2 = \emptyset$. We use the notation $M = M_1||M_2$, $O_1' = O_1 \cap I_2$, and $O_2' = O_2 \cap I_1$. The algorithm performs the following steps:

1. Reduce M_1 with respect to O_1', resulting in M_1^r.
2. Reduce M_2 with respect to O_2', resulting in M_2^r.
3. Compose $M_1^e = M_1||M_2^r$.
4. Compose $M_2^e = M_1^r||M_2$.
5. Reduce M_1^e with respect to O_1, resulting in M_1^d.
6. Reduce M_2^e with respect to O_2, resulting in M_2^d.
7. Compose $M_d = M_1^d||M_2^d$.

Table 1 presents the inputs and outputs of the FSMs constructed by the improved algorithm.

An example for the *improved algorithm* is presented in Figure 2. The intuition behind the improved algorithm is as follows. When two FSMs are composed, each restricts the behavior of the other by providing a real environment, rather than an open one. States that behaved differently from one another are now indistinguishable and can be collapsed into the same equivalence class.

Our goal is to minimize M_1 and M_2 in separation, while taking into account the environment each runs in. While minimizing M_2, it is sufficient to consider only the part of M_1 which influences M_2. M_1^r is exactly that part. Therefore, states in M_2 that become indistinguishable in $M = M_1 \| M_2$ are

FSM	Input	Output
M_1	I_1	O_1
M_2	I_2	O_2
M_1^r	I_1	O_1'
M_2^r	I_2	O_2'
M	$(I_1 \setminus O_2) \cup (I_2 \setminus O_1)$	$O_1 \cup O_2$
M_1^e	$(I_1 \setminus O_2') \cup (I_2 \setminus O_1) = (I_1 \setminus O_2) \cup (I_2 \setminus O_1)$	$O_1 \cup O_2'$
M_2^e	$(I_1 \setminus O_2) \cup (I_2 \setminus O_1') = (I_1 \setminus O_2) \cup (I_2 \setminus O_1)$	$O_2 \cup O_1'$
M_1^d	$(I_1 \setminus O_2) \cup (I_2 \setminus O_1)$	O_1
M_2^d	$(I_1 \setminus O_2) \cup (I_2 \setminus O_1)$	O_2
M_d	$(I_1 \setminus O_2) \cup (I_2 \setminus O_1)$	$O_1 \cup O_2$

Table 1: The inputs and outputs of the intermediate FSMs in the improved algorithm

also indistinguishable in $M_2^e = M_1^r \| M_2$. These states are collapsed, resulting in M_2^d. Similarly, in M_1^e, states of M_1 that are indistinguishable in M are collapsed (resulting in M_1^d). When M_1^d and M_2^d are finally composed, M_d is the result of a composition of two minimized FSMs which do not interact, and therefore M_d is minimized.

The skeleton of the correctness proof for the algorithm is given in the lemma below. In the rest of the section we prove each of the claims, thus proving the correctness of our algorithm.

Lemma 4.1

- *M_1^e and M are bisimulation equivalent with respect to $O_1 \cup O_2'$.*
- *M_2^e and M are bisimulation equivalent with respect to $O_2 \cup O_1'$.*
- *M_1^d and M are bisimulation equivalent with respect to O_1.*
- *M_2^d and M are bisimulation equivalent with respect to O_2.*
- *M_d and M are bisimulation equivalent with respect to $O_1 \cup O_2$*
- *M_d is minimized with respect to $O_1 \cup O_2$.*

Lemma 4.2 *M_1^e and M are bisimulation equivalent with respect to $O_1 \cup O_2'$.*

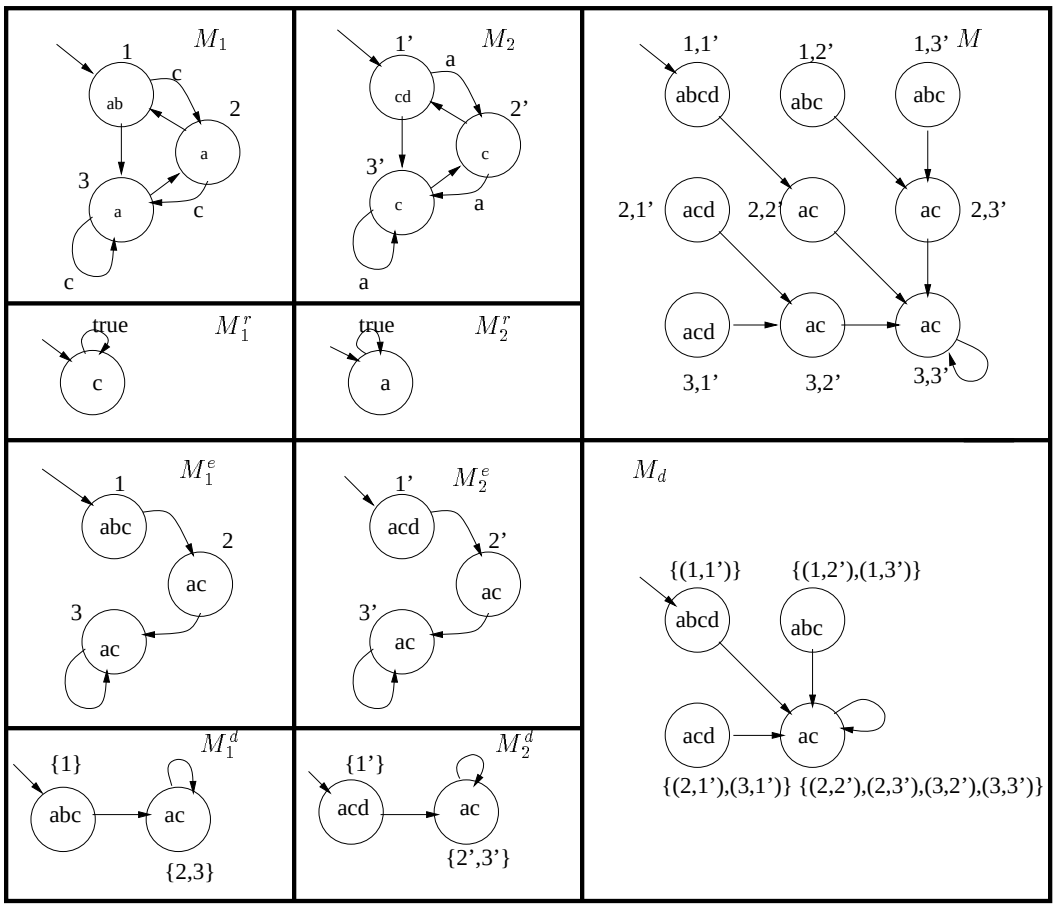

Figure 2: An example of the deterministic version of the improved algorithm: M_1 has input set $I_1 = \{c\}$ and output set $O_1 = \{a, b\}$. M_2 has input set $I_2 = \{a\}$ and output set $O_2 = \{c, d\}$. Note that even though M_1 and M_2 are minimized, M is not. M_d is the quotient model of M. It can also be obtained by composing M_1^d and M_2^d. The states of M_1^d, M_2^d and M_d are given as the sets of states in the equivalence classes the states represent.

Proof: Let $H_1^e \subseteq S \times S_1^e$ be $H_1^e = \{((s_1, s_2), (s_1, s_2^r)) | s_2^r \text{ is the equivalence class of } s_2\}$. We prove that H_1^e is a bisimulation relation.

- For every $(s_{10}, s_{20}) \in S_0$, we have $((s_{10}, s_{20}), (s_{10}, [s_{20}])) \in H_1^e$. Similarly, for every $(s_{10}, \alpha_0) \in S_{10}^r$, let s_{20} be the initial state in α. Then $((s_{10}, s_{20}), (s_{10}, [s_{20}])) \in H_1^e$.

Let $((s_1, s_2), (s_1, s_2^r)) \in H_1^e$:

- Since the labeling of an equivalence class is equal to the labeling of the states it contains, $L_2(s_2) \cap O_2' = L_2^r(s_2^r)$. The definition of composition therefore implies that $L((s_1, s_2)) \cap (O_1 \cup O_2') = L_1^e((s_1, s_2^r))$.

- Let $((s_1, s_2), i, (s_1', s_2'))$ be an element in R. This implies that for $i_1 = (i \cup L_2(s_2)) \cap I_1$, $(s_1, i_1, s_1') \in R_1$ and for $i_2 = (i \cup L_1(s_1)) \cap I_2$, $(s_2, i_2, s_2') \in R_2$. Let ${s_2}'^r$ be the equivalence class of s_2'. Then $(s_2^r, i_2, {s_2}'^r) \in R_2^r$. Since $L_2(s_2) \cap I_1 = L_2(s_2) \cap O_2' = L_2^r(s_2^r)$, $i_1 = (i \cup L_2^r(s_2^r)) \cap I_1$. The definition of composition implies that $((s_1, s_2^r), i, ({s_1}', {s_2}'^r)) \in R_1^e$. By the definition of H_1^e, $((s_1', s_2'), ({s_1}', {s_2}'^r)) \in H_1^e$.

- Let $((s_1, s_2^r), i, ({s_1}', {s_2}'^r))$ be an element in R_1^e. This implies that for $i_1 = (i \cup L_2^r(s_2^r)) \cap I_1$, $(s_1, i_1, s_1') \in R_1$ and for $i_2 = (i \cup L_1(s_1)) \cap I_2$, $(s_2^r, i_2, {s_2}'^r) \in R_2^r$. By Lemma 3.1, there exists a state s_2' such that $(s_2, i_2, s_2') \in R_2$ and ${s_2}'^r$ is the equivalence class of s_2'. Since $L_2(s_2) \cap I_1 = L_2(s_2) \cap O_2' = L_2^r(s_2^r)$, $i_1 = (i \cup L_2(s_2)) \cap I_1$. By the definition of composition, $((s_1, s_2), i, (s_1', s_2')) \in R$, and by the definition of H_1^e, $((s_1', s_2'), ({s_1}'^r, {s_2}'^r)) \in H_1^e$. $\square$

Lemma 4.3 *M_2^e and M are bisimulation equivalent with respect to $O_1' \cup O_2$.*

The proof is similar to the proof of Lemma 4.2.

Lemma 4.4 *M_1^d and M are bisimulation equivalent with respect to O_1.*

Proof : Proposition 3.6 together with Lemma 4.2 implies that M_1^e and M are bisimulation equivalent with respect to O_1. Lemma 3.4 implies that M_1^e and M_1^d are bisimulation equivalent with respect to O_1. Since bisimulation equivalence is transitive, then M and M_1^d are bisimulation equivalent with respect to O_1. $\square$

Lemma 4.5 *M_2^d and M are bisimulation equivalent with respect to O_2.*

The proof is similar to the proof of Lemma 4.4.

Lemma 4.6 *If M_1 and M_2 are deterministic, then M_d and M are bisimulation equivalent with respect to $O_1 \cup O_2$.*

Note that both M_1^d and M_2^d have the same input (I) and $I \cap O_1 = I \cap O_2 = \emptyset$.

Proof: Let $H_1^d \subseteq S \times S_1^d$ and $H_d^2 \subseteq S \times S_2^d$ be bisimulation relations over $M \times M_1^d$ and $M \times M_2^d$ respectively. Let $H_d \subseteq S \times S_d$ be the following relation: $H_d = \{((s_1, s_2), (s_1^d, s_2^d)) | ((s_1, s_2), s_1^d) \in H_d^1$ and $((s_1, s_2), s_2^d) \in H_d^2\}$. We prove that H_d is a bisimulation relation.

- $((s_{01}, s_{02}), s_{01}^d) \in H_d^1$ and $((s_{01}, s_{02}), s_{02}^d) \in H_d^2$ imply that $((s_{01}, s_{02}), (s_{01}^d, s_{02}^d)) \in H_d$.

Let $((s_1, s_2), (s_1^d, s_2^d))$ be a pair in H_d.

- $((s_1, s_2), s_1^d) \in H_d^1$ implies that $L((s_1, s_2)) \cap O_1 = L_1^d(s_1^d)$. $((s_1, s_2), s_2^d) \in H_d^2$ implies that $L((s_1, s_2)) \cap O_2 = L_2^d(s_2^d)$. Thus, $L((s_1, s_2)) = L_d((s_1^d, s_2^d))$.

- Let $((s_1, s_2), i, (s_1', s_2'))$ be an element in R. Since $((s_1, s_2), s_1^d) \in H_d^1$, there exists a state ${s_1}'^d$ such that $(s_1^d, i, {s_1}'^d) \in R_1^d$ and $((s_1', s_2'), {s_1}'^d) \in H_d^1$. Since $((s_1, s_2), s_2^d) \in H_d^2$, there exists a state ${s_2}'^d$ such that $(s_2^d, i, {s_2}'^d) \in R_2^d$ and $((s_1', s_2'), {s_2}'^d) \in H_d^2$. The definition of composition implies that $((s_1^d, s_2^d), i, ({s_1}'^d, {s_2}'^d)) \in R_d$ and by the definition of H_d, $((s_1', s_2'), ({s_1}'^d, {s_2}'^d)) \in H_d$.

- Let $((s_1^d, s_2^d), i, ({s_1}'^d, {s_2}'^d))$ be an element in R_d. Then $(s_1^d, i, {s_1}'^d) \in R_1^d$ and $(s_2^d, i, {s_2}'^d) \in R_2^d$. Since $((s_1, s_2), s_1^d) \in H_d^1$, there exists a state (s_1', s_2') such that $((s_1, s_2), i, (s_1', s_2')) \in R$ and $((s_1', s_2'), {s_1}'^d) \in H_d^1$. Since $((s_1, s_2), s_2^d) \in H_d^2$, there exists a state (s_1'', s_2'') such that $((s_1, s_2), i, (s_1'', s_2'')) \in R$ and $((s_1'', s_2''), {s_2}'^d) \in H_d^2$. Since M is deterministic, $(s_1', s_2') = (s_1'', s_2'')$. By the definition of H_d, $((s_1', s_2'), ({s_1}'^d, {s_2}'^d)) \in H_d$. □

M_1^d and M_2^d are minimized with respect to O_1 and O_2 respectively. Furthermore, $I \cap O_1 = I \cap O_2 = \emptyset$, and thus Lemma 3.7 induces the following corollary.

Corollary 4.7 *M_d is minimized with respect to $O_1 \cup O_2$.*

4.2 Time and space complexity

In this section we compare between the complexity of the naive algorithm and the complexity of the improved algorithm.

The algorithms include two basic operations:

1. Composing two FSMs, $M'' = M \| M'$. The most costly part of this operation is the computation of the transition relation R''. This can be done in time and space complexity of $O(|R''|)$.

2. Minimizing an FSM M into its quotient FSM, M_Q. The algorithms have the same complexity as the one in [11, 17]. Their space complexity is $O(|R|)$ and their time complexity is $O(|R| \cdot log(|S|))$.

Thus, the minimization is the dominant part of the algorithm. In the naive algorithm there is only one minimization of $M = M_1 \| M_2$. In the improved algorithm, however, there are 4 minimizations: The minimization of M_1 that results in M_1^r, the minimization of M_2 that results in M_2^r, the minimization of M_1^e that results in M_1^d, and the minimization M_2^e that results in M_2^d.

Since the complexity of a minimization depends on the size of the minimized FSM, we need to compare the sizes of M_1, M_2, M_1^e, M_2^e, to the size of M. We assume that the size of M_1 is equal to the size of M_2.

The differences in the sizes of M_1 and M_2 and the that of M depend on the interactions between M_1 and M_2. The interaction between M_1 and M_2 is measured by the number of inputs of one that are outputs of the other. The size of the state spaces of M_1 and M_2, is the square root of the size of the state space of M. However, the size of the transition relation depends on the interactions. When the interaction between M_1 and M_2 is high, many inputs of M_1 and M_2 are connected to the corresponding outputs of M_2 and M_1. These inputs are not part of the inputs of M[1]. In this case, every component in M_2 is an input of M_1 and vice versa. Thus, $|S_1| \cdot |2^{I_1}| \approx |S_2| \cdot |2^{I_2}| \approx |S| \cdot |2^I|$. Since $|R_1| \approx |S_1| \cdot |2^{I_1}|$ and $|R_2| \approx |S_2| \cdot |2^{I_2}|$, $|R_1| \approx |R_2| \approx |R|$, and $|M_1| \approx |M_2| \approx |M|$.

Next we compare the sizes of M_1^e and M_2^e with the size of M. Note that $M = M_1 \| M_2$, $M_1^e = M_1 \| M_2^r$ and $M_2^e = M_1^r \| M_2$. This implies that the difference in the sizes of M and M_1^e depends on the difference in the sizes of M_2 and M_2^r. Similarly, the difference between the sizes of M and M_2^e depends on the difference between M_1 and M_1^r. When there is no redundancy, $|M_1| = |M_1^r|$ and $|M_2| = |M_2^r|$. In this case, $|M_1^e| = |M_2^e| = |M|$.

The worst-case scenario is when the interaction between M_1 and M_2 is high and there is no redundancy in M_1 and M_2, $|M_1| = |M_1^r|$ and $|M_2| = |M_2^r|$. In this case the improved algorithm performs four minimizations, each requiring the same time as the single minimization of the naive algorithm.

[1] Recall that $I = (I_1 \setminus O_2) \cup (I_2 \setminus O_1)$.

Since we need to keep at most three different models at the same time, the space requirement of the improved algorithm is three times that of the naive algorithm.

In the best scenario, however, $|M_1| = |M_2| = |M_1^e| = |M_2^e| = \sqrt{|M|}$. In this scenario, the time complexity of $|R| \cdot log(|S|)$ in the naive algorithm, becomes $4 \cdot \sqrt{|R|} \cdot log(\sqrt{|S|})$ in the improved algorithm. This time complexity is significantly better.

4.3 Nondeterministic FSMs

In this section we extend the modular method to nondeterministic FSMs, for which, Lemma 4.6 does not hold. The result M_d of composing M_1^d and M_2^d might be inequivalent to M due to "illegal states" in M_d which are not equivalent to any state in M.

In order to understand this inequality, we inspect the role of the states of M_1 in M (the role of the states of M_2 is similar). Since M is a composition of M_1 and M_2, every state s_1 has two functions: the first is to determine the outputs and the next state of M_1, and the second is to determine the inputs of M_2.

In M_d these two functions are fulfilled by two states of S_1. Let $([(s_1, [s_2])], [([t_1], t_2)])$ be a state in M_d. Then s_1 determines the outputs and the next states of M_1, and t_1 determines the inputs of M_2. A state $([(s_1, [s_2])], [([t_1], t_2)])$ of M_d might be illegal when $s_1 \notin [t_1]$. In this case, the combination of the next state in M_1 and the input of M_2 does not occur in any state of S_1.

The problem of illegal states is demonstrated in Figure 3. In this figure, all the states of M_1 and M_2 are initial states. Therefore, M_1 and M_2 are nondeterministic. M_1^r and M_2^r cannot be further reduced, and the same holds for M_1^e and M_2^e. Since the result M_d of composing M_1^d and M_2^d is minimized and contains 16 initial states, it cannot be bisimulation equivalent to $M = M_1||M_2$. The error in the algorithm is due to illegal states such as $((0,2),(1,2))$ in M_d. This illegal state is related to both s_0 and s_1 in M_1 and is not equivalent to any state in M.

We now present the nondeterministic systems version of the improved algorithm. This algorithm restricts the states of M_d to legal states only. As before, the minimized FSM is constructed without constructing the non-minimized FSM $M_1 \| M_2$ itself. First we define two functions.

Definition 4.8 *The function* $f_1 : S_1 \times S_2 \rightarrow S_1^d$ *is defined as follows:* $f_1(s_1, s_2) = [(s_1, [s_2])]$.

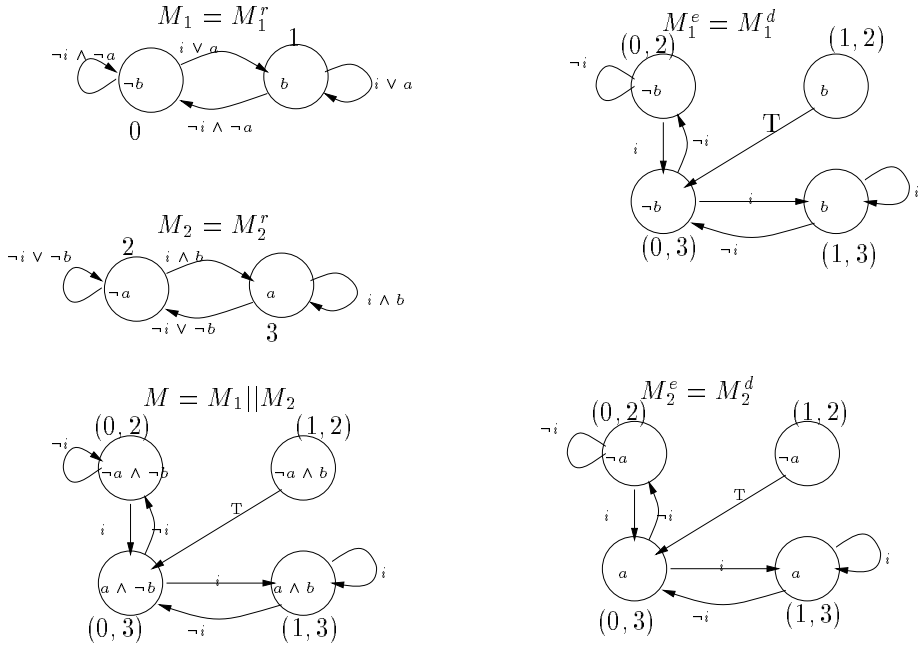

Figure 3: An example of an inequivalent result of the deterministic systems version of the improved algorithm, where M_1 and M_2 are not deterministic.

Definition 4.9 *The function* $f_2 : S_1 \times S_2 \rightarrow S_2^d$ *is defined as follows:* $f_2(s_1, s_2) = [([s_1], s_2)]$.

Next, we define a new FSM M_d', which is similar to M_d except that the set of states is restricted:
$S_d' = \{(s_1^d, s_2^d) | \exists s_1, s_2,\ s_1^d = f_1(s_1, s_2) \wedge s_2^d = f_2(s_1, s_2)\}$. The definitions for the other components of M_d' are straightforward. $S_0^{d\prime} = S_0{}^d \cap S_d'$, the inputs, outputs, and labeling function remain the same, and $R_d' = R_d \cap (S_d' \times S_d')$. We now prove that M_d' is bisimulation equivalent to M.

Lemma 4.10 *M_d' and M are bisimulation equivalent with respect to $O_1 \cup O_2$.*

Proof : Let $H \subseteq S \times S_d'$ be defined as follows: $H = \{((s_1, s_2), (s_1^d, s_2^d)) |\ s_1^d = f_1(s_1, s_2)\ \wedge\ s_2^d = f_2(s_1, s_2)\}$. We prove that H is a bisimulation relation.

- The definition of S_{d0}' implies that for every state $(s_{10}, s_{20}) \in S_0$ there exists a state $(s_{10}^d, s_{20}^d) = ([(s_{10}, [s_{20}])], [([s_{10}], s_{20})]) \in S_{d0}'$ such that

$((s_{10}, s_{20}), (s^d_{10}, s^d_{20})) \in H$. For the other direction, assume that $(s^d_{10}, s^d_{20}) = ([(s_{10}, [s_{20}])], [([s_{10}], s_{20})])$ is a state in S'_{d0}. Then, the state (s_{10}, s_{20}) is in S_0 and $((s_{10}, s_{20}), (s^d_{10}, s^d_{20})) \in H$.

- Let $((s_1, s_2), (s^d_1, s^d_2))$ be an element in H. Since $L^d_1([(s_1, [s_2])]) = L^e_1(s_1, [s_2]) \cap O_1 = L_1(s_1)$ and $L^d_2([([s_1], s_2)]) = L^e_2([s_1], s_2) \cap O_2 = L_2(s_2)$, $L((s_1, s_2)) = L_d(([(s_1, [s_2])], [([s_1], s_2)]))$.

- Let $((s_1, s_2), (s^d_1, s^d_2))$ be in H and let i be an element in I. Let $i_1 = (i \cup L_2(s_2)) \cap I_1 = (i \cup L^r_2([s_2])) \cap I_1$ and

  - $i_2 = (i \cup L_1(s_1)) \cap I_2 = (i \cup L^r_1([s_1])) \cap I_2$. Then, $((s_1, s_2), i, (s'_1, s'_2)) \in R$ iff
  - $(s_1, i_1, s'_1) \in R_1$ and $(s_2, i_2, s'_2) \in R_2$ iff (Definition 2.6 and Lemma 3.1)
  - $([s_1], i_1, [s'_1]) \in R^r_1$ and $([s_2], i_2, [s'_2]) \in R^r_2$. $(s_1, i_1, s'_1) \in R_1$ and $([s_2], i_2, [s'_2]) \in R^r_2$ iff $((s_1, [s_2]), i, (s'_1, [s'_2])) \in R^e_1$.
  - Similarly, $([s_1], i_1, [s'_1]) \in R^r_1$ and $(s_2, i_2, s'_2) \in R_2$ iff $(([s_1], s_2), i, ([s'_1], s'_2)) \in R^e_2$.
  - Therefore, $((s_1, [s_2]), i, (s'_1, [s'_2])) \in R^e_1$ and $(([s_1], s_2), i, ([s'_1], s'_2)) \in R^e_2$ iff
  - $(s^d_1, i, s^{d\prime}_1) = ([(s_1, [s_2])], i, [(s'_1, [s'_2])])$ is in R^d_1 and $(s^d_2, i, s^{d\prime}_2) = ([([s_1], s_2)], i, [([s'_1], s'_2)])$ is in R^d_2 iff
  - $((s^d_1, s^d_2), i, (s^{d\prime}_1, s^{d\prime}_2)) \in R_d$.

□

Next, we prove that M'_d is minimized. First, we show that the maximal bisimulation over M'_d includes a bisimulation over M^d_2.

Lemma 4.11 *Let H'_d be the maximal bisimulation relation over M'_d. We define a relation H^d_1 over $S^d_1 \times S^d_1$ as follows: $([(s_1, [s_2])], [(t_1, [t_2])]) \in H^d_1$ iff $(([(s_1, [s_2])], [([s_1], s_2)]), ([(t_1, [t_2])], [([t_1], t_2)])) \in H'_d$. Then H^d_1 is a bisimulation relation.*

Proof :

- Since H'_d contains all identity pairs, H^d_1 contains all identity pairs as well. This implies that for every initial state, the pair consisting of the initial state and itself is an element in H^d_1.

Let $([(s_1, [s_2])], [(t_1, [t_2])])$ be an element in H^d_1:

- $L_d(([[(s_1,[s_2])],[([s_1],s_2)])) = L_d(([[(t_1,[t_2])],[([t_1],t_2)]))$ implies that $L^1_d([[(s_1,[s_2])]]) = L^1_d([[(t_1,[t_2])]])$.
- Let $([(s_1,[s_2])],i,[(s'_1,[s'_2])])$ be an element in R^d_1. Let $i_1 = (i \cup L_2(s_2)) \cap I_1 = (i \cup L^r_2([s_2])) \cap I_1$ and $i_2 = (i \cup L_1(s_1)) \cap I_2 = (i \cup L^r_1([s_1])) \cap I_2$.
  1. By Lemma 3.1, $((s_1,[s_2]),i,(s'_1,[s'_2])) \in R^e_1$.
  2. Thus, $(s_1,i_1,s'_1) \in R_1$ and $([s_2],i_2,[s'_2]) \in R^r_2$.
  3. By Definition 2.6 and Lemma 3.1, $([s_1],i_1,[s'_1]) \in R^r_1)$ and $(s_2,i_2,s'_2) \in R_2$.
  4. This implies that $([([s_1],s_2)],i,[([s'_1],s'_2)]) \in R^d_2$.
  5. Thus, $(([[(s_1,[s_2])],[([s_1],s_2)]),i,([[(s'_1,[s'_2])],[([s'_1],s'_2)])) \in R'_d$.
  6. Since H'_d is a bisimulation relation, there exists a state $([(t'_1,[t'_2])],[([t'_1],t'_2)])$ such that $(([[(t_1,[t_2])],[([t_1],t_2)]),i,([[(t'_1,[t'_2])],[([t'_1],t'_2)])) \in R'_d$ and $(([[(s'_1,[s'_2])],[([s'_1],s'_2)]),([[(t'_1,[t'_2])],[([t'_1],t'_2)])) \in H'_d$.
  7. This implies that $([(t_1,[t_2])],i,[(t'_1,[t'_2])]) \in R^d_1$ and $([(s'_1,[s'_2])],[(t'_1,[t'_2])]) \in H^d_1$.
- Similarly, we can prove that for every state $[(t'_1,[t'_2])]$ such that $([(t_1,[t_2])],i,[(t'_1,[t'_2])]) \in R^d_1$ there exists a state $[(s'_1,[s'_2])]$ such that $([(s_1,[s_2])],i,[(s'_1,[s'_2])]) \in R^d_1$ and $([(s'_1,[s'_2])],[(t'_1,[t'_2])]) \in H^d_1$.

□

Lemma 4.12 *Let H'_d be the maximal bisimulation relation over M'_d. We define a relation H^d_2 over $S^d_2 \times S^d_2$ as follows: $([([s_1],s_2)],[([t_1],t_2)]) \in H^d_2$ iff $(([(s_1,[s_2])],[([s_1],s_2)]),([[(t_1,[t_2])],[([t_1],t_2)])) \in H'_d$. Then H^d_2 is a bisimulation relation.*

The proof of Lemma 4.12 is similar to the proof of Lemma 4.11.

Lemma 4.13 *M'_d is minimized.*

Proof Let H_d be the maximal bisimulation over $M'_d \times M'_d$. Assume to the contrary that the lemma does not hold. Then by Lemma 3.3, there are two different states $(s^d_1,s^d_2),(t^d_1,t^d_2)$ such that $((s^d_1,s^d_2),(t^d_1,t^d_2)) \in H_d$. Since $(s^d_1,s^d_2) \neq (t^d_1,t^d_2)$, either $s^d_1 \neq t^d_1$ or $s^d_2 \neq t^d_2$. Assume w.l.o.g. that $s^d_1 \neq t^d_1$. Let H^d_1 be the relation defined in Lemma 4.11. By Lemma 4.11, H^d_1 is a bisimulation. By the definition of H^d_1, $(s^d_1,t^d_1) \in H^d_1$. By Lemma 3.3, M^d_1 is not minimized, a contradiction. □

4.4 Additional complexity

The additional complexity is due to the computation of S'_d, which forces us to refer to the whole state space of M. Nevertheless, since we only compute the state space and do not use it in the reduction method, the nondeterministic systems version of the improved algorithm is still better than the naive algorithm. f_1 and f_2, can be computed during the construction of M_1^r and M_2^r and the construction of M_1^d and M_2^d without any additional time complexity. However, since the function operates on the states of $M_1||M_2$, the space complexity is $|S| = |S_1| \cdot |S_2|$. In a worst-case scenario, the complexity of the nondeterministic improved algorithm is identical to that of the deterministic improved algorithm. However, when $M_1^r \ll M_1$ and $M_2^r \ll M_2$, this complexity is worse than that of the deterministic version.

5 An implementation of the improved algorithm

In this section we describe an implementation of the improved algorithm. Our goal is to compare the improved algorithm, the naive algorithm, and the ordinary algorithm. The ordinary algorithm minimizes a given FSM directly and does not use modularity. The implementation was developed in the sequential equivalence verification CAD group of Intel design technologies in Haifa. The designs, which were tested in the equivalence department, have the following properties:

1. $S_0 = S$, i.e., every state in the model is an initial state.
2. The transition relation is a function, meaning that for every state s and input i there exists exactly one state t, such that (s, i, t) is a transition in R.

Note that the first property makes these designs nondeterministic. The above two properties prompted us to choose the nondeterministic systems version of the improved algorithm. However, we represent the transition relation as a function, which can be represented more concisely than a regular relation.

A general description of the implementation is given in Section 5.1. The improved algorithm uses the ordinary algorithm as a subroutine. The same ordinary algorithm is used for comparison with the improved algorithm. Since we deal with FSMs that have a transition relation that is a function, we use an algorithm that is similar to the algorithm presented in [11]. The experimental results are presented in Section 5.2.

```
typedef struct fsm {
    VarList     inputs;
    BddFunction outputs;
    BddFunction latches;
    BDD         domain;
    BddFunction equivFunc;
}FSM;
```

Figure 4: The data structure that models FSMs

5.1 The implementation framework

The minimization algorithms (improved, naive, or ordinary) receive an FSM from an Intel program, which compiles the RTL description of the design into an FSM. The given FSM contains three lists: A list of inputs, a list of latches, and a list of outputs. The list of *inputs* contains BDD variables only. The list of *latches*, which encodes the state space, is consists of pairs, with each pair containing a BDD variable and a BDD representing the next state function. The list of *outputs*, which encodes the labeling function, consists of pairs, with each pair containing a BDD variable and a BDD representing the output function.

We modeled an FSM by the *FSM* data structure shown in Figure 4. In addition to the inputs, latches, and outputs fields, the FSM data structure has the *domain* field, which is a BDD over the latches and represents the set of states, and the *equivFunc* field. When a minimization of an FSM is performed, a set of equivalence classes is constructed. These classes are the states of the resulting FSM. The equivFunc field of the resulting FSM contains a function that relates the states of the original FSM to their equivalence classes.

The information about the modular structure of the tested designs was lost during the development stage. Thus, instead of a set of components, the improved algorithm receives one FSM. In order to perform the minimization, it first partitions the FSM and then executes the improved algorithm. A basic description of the implementation of the improved algorithm is presented in Figure 5.

The algorithm receives an FSM om and partitions it into two FSMs, $m1$ and $m2$. Then it uses the improved algorithm to construct a minimized

model md, which is equivalent to om. The algorithm partitions the model by partitioning the set of latches and the set of outputs, (it is possible for $m1$ and $m2$ to share inputs). The goal of the partition is to minimize the interaction between the models. Since it is hard to find such a partition, the algorithm uses a heuristic to find a partition with low interaction.

The improved algorithm uses the subroutine *reduction*, which executes the ordinary algorithm. The algorithm is an adaptation of the algorithm given in [11] for constructing the quotient automaton for a given regular deterministic automaton. The algorithm is adapted for FSMs for which the transition relation is a function. Given an FSM, it constructs its quotient FSM. The main difference between the algorithm in [11] and the ordinary algorithm is in the initial partitioning. While for automata the initial partition forms two sets (accepting and rejecting), for FSM, the states are initially partitioned into $2^{|AP|}$ sets, one for each state labeling.

Both minimization algorithms (the improve and the ordinary) minimize the FSM with respect to its outputs. Thus, before they minimize M_1 into M_1^r (M_2 into M_2^r), they need to remove the outputs in $O_1 \setminus I_2$ ($O_2 \setminus I_1$). The algorithms use the *rmExternalOutputs* subroutine to remove these external outputs.

In order to construct the set rd of "legal states" of the form $([(s_1, [s_2])], [([s_1], s_2)])$, the algorithm constructs two functions, $f1d : S \to S_1^d$ and $f2d : S \to S_1^d$. In order to construct $f1d$, the algorithm composes the functions $M1d.equivFunc : S_1^e \to S_1^d$ and the function $m2r.equivFunc : S_2 \to S_2^r$. Since $S_1^e = S_1 \times S_2^r$, the resulting function relates the states of $S_1 \times S_2$ to the states of S_1^d. The function $f2d$ is constructed in a similar way. Then the algorithm calculates $rd = fd(om.domain)$, where $fd : S \to S_d$ is defined as follows: $fd(s) = (f1d(s), f2d(s))$.

The sets, functions and relations are represented by BDDs. We use Intel's BDD package for the implementation.

5.2 Experimental results

We compared the ordinary algorithm, the naive algorithm, and the improved algorithm. During testing of the improved algorithm, we discovered that the minimization of M_1^r and M_2^r does not improve performance. Thus, we also tested the algorithm without these minimizations. In this case M_1^e (M_2^e) are simply M with only some of the outputs. This test was performed with the design partitioned only once (this appears in the tables as improved2), and with the design partitioned recursively until it has only one output (this

```
FSM improvedAlgorithm(FSM om){
  FSM         m1, m2, m1r, m2r, m1e, m2e, m1d, m2d, md;
  BddFunction fd, f1d, f2d;
  BDD         re, rd;

  /* the recursion tail condition - based on the size of the model */
  if (!shouldSplit(om))
    return reduction(om);

  /* partition om to m1 and m2 */
  partModel(om, m1, m2);

  m1r = rmExternalOutputs(m1);
  m1r = improvedAlgorithm(m1r);
  m2e = modelComposition(m1r, m2);

  m2r = rmExternalOutputs(m2);
  m2r = improvedAlgorithm(m2r);
  m1e = modelComposition(m1, m2r);

  m1d = reduction(m1e);
  m2d = reduction(m2e);

  f1d = composeFunc(m1d.equivFunc, m2r.equivFunc);
  f2d = composeFunc(m2d.equivFunc, m1r.equivFunc);
  fd = joinBddFunc(f1d,f2d);

  rd = bdd_image(om.domain, fd);
  md = disjointComposition(m1d, m2d, rd);

  return md;
}
```

Figure 5: The improved algorithm

appears in the tables as improved3).

The results are presented in the following tables. In Table 2 we present general properties of the tested designs. Table 3 compares the minimization times of the algorithms. Table 4 compares the space requirements of the algorithms. The algorithms were tested on a machine with two CPUs of 550 MHZ each and 2GB memory.

The experimental results imply that in most designs, all versions of the improved algorithm perform better than the ordinary and naive algorithms in both time and space. The best time performance is for the improved algorithm without the minimization of M_1^r and M_2^r and with recursive partitioning of the outputs. The best space performance is for the improved algorithm without the minimization of M_1^r and M_2^r and with only one partition of the outputs.

The differences between these two versions of the improved algorithm demonstrate the tradeoff between the algorithm's efficiency and its overhead. While the algorithm's efficiency results in a better run time, the overhead results in larger space requirements. This tradeoff is taken into account in the subroutine `shouldSplit`. This subroutine that decides whether to reduce the sub-model by further partitioning it with the improved algorithm or to use the ordinary reduction algorithm. In general, if the sub-model is too small, then the overhead the improved algorithm become too large.

Note that, while the improved algorithm is up to 12 times faster than the ordinary minimization algorithm in some cases, the difference between the two algorithms is small in those cases when the ordinary algorithm perform better.

6 Conclusions

Modularity is used extensively in the development of systems. As a result, most systems have a modular structure. In this work we show how this structure can be used for a better minimization algorithm. Given an FSM M, we construct two disjoint FSMs, M_1^e and M_2^e, such that M is equivalent to the synchronized composition of M_1^e and M_2^e. Once we construct these FSMs, the problem of minimizing M is reduced to minimizing M_1^e and M_2^e separately and composing the result. Since the complexity of minimizing M might be quadratically greater than minimizing M_1^e and M_2^e separately, the potential of the algorithm is huge.

Name	No. of inputs	No. of latches	No of outputs
$s298$	5	14	7
$s298d2$	5	11	3
$s298d3$	5	13	5
$s400d1$	5	17	2
$s400d2$	5	17	2
$s400d3$	5	19	4
$s400$	5	21	6
$s349$	11	15	11
$s444.2$	5	20	5
$s444$	5	21	6

Table 2: General properties of the tested designs

Name	ordinary algorithm	naive algorithm	improved algorithm	improved2 algorithm	improved3 algorithm
s298	46	72	34	27	27
s298d2	21	26	22	22	23
s298d3	32	41	29	26	26
s400d1	190	469	385	206	206
s400d2	173	575	500	239	239
s400d3	1,336	2,048	1,209	590	601
s400	12,396	space overflow	2,302	1,129	999
s349	2,640	4,496	1,759	1,474	255
s444.2	3,512	3,379	1,380	828	799
s444	9,362	space overflow	2,891	1,055	1,038

Table 3: The minimization time in seconds for the different algorithms

Name	ordinary algorithm	naive algorithm	improved algorithm	improved2 algorithm	improved3 algorithm
s298	1,482,712	1,919,032	451,793	326359	467,705
s298d2	151,885	209,452	85,108	125577	144,624
s298d3	759,032	716,557	278,283	337834	362,691
s400d1	8,691,398	12,521,457	10,105,732	5,118214	5,118,214
s400d2	9,368,512	12,775,984	7,009,955	4,820907	4,820,907
s400d3	28,436,930	41,089,896	12,540,649	10,714419	25,165,669
s400	105,175,584	space overflow	32,491,632	17,740753	44,641,501
s349	27,567,754	36,747,754	12,442,018	2,552658	3,876,761
s444.2	49,964,703	66,788,414	19,376,000	17,451240	33,190,412
s444	97,687,526	space overflow	21,972,212	17,168679	43,223,054

Table 4: The maximal number of BDD nodes required by the different minimization algorithms

Acknowledgments: I want to thank to the sequential equivalence verification CAD group of Intel design technologies in Haifa, and Ziyad Hanna in particular, for the use of their system and for their helpful support.

References

[1] A. Aziz, T.R. Shiple, V. Singhal, and A.L. Sangiovanni-Vincetelly. Formula-dependent equivalence for compositional CTL model checking. In D. Dill, editor, *Proceedings of the Sixth Conference on Computer Aided Verification (CAV'94)*, volume 818 of *LNCS*, pages 324–337, 1994.

[2] A. Aziz, V. Singhal, G.M. Swamy, and R.K. Brayton. Minimizing interacting finite state machines: A compositional approach to language containment. In *Proceedings of the International Conference on Computer Design*, pages 255–261, 1994.

[3] M. Ben-Ari, Z. Manna, and A. Pnueli. The temporal logic of branching time. *Acta Informatica*, 20:207–226, 1983.

[4] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, August 1986.

[5] E.M. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. In *Formal Methods in System Design*, pages 77–104, 1996.

[6] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 1999.

[7] E.A. Emerson and E.M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *LNCS*, volume 85, pages 169–181, 1980.

[8] K. Fisler and M. Vardi. Bisimulation minimization in an automata-theoretic verification framework. In *Formal Methods in Computer-Aided Design (FMCAD)*, pages 115–132, 1998.

[9] K. Fisler and M. Vardi. Bisimulation and model checking. In *Conference on Correct Hardware Reasoning Methods (CHARME) 1999*, pages 338–341, 1999.

[10] Susanne Graf, Bernhard Steffen, and Gerlad Lüttgen. Compositional minimisation of finite state systems using interface specifications. *Formal Aspects of Computing*, 8(5):607–616, 1996.

[11] J. E. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In Z. Kohavi and A. Paz, editors, *Theory of Machines and Computations*. Academic Press, New York, 1971.

[12] D. Kozen. Results on the propositional μ-calculus. *TCS*, 27, 1983.

[13] J.P. Krimm and L. Mounier. Compositional state space generation from lotos programs. In *TACAS*, LNCS1217, pages 239–258, 1997.

[14] D. Lee and M. Yannakakis. Online minimization of transition systems. In *Proceedings of the 24th ACM Symp. on Theory of Computing*, 1992.

[15] R. Milner. *Communication and Concurrency*. Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[16] E. F. Moore. Gedanken–experiments on sequential machines. In C. E. Shannon and J. McCarthy, editors, *Annals of Mathematics Studies (34), Automata Studies*, pages 129–153. Princeton University Press, Princeton, NJ, 1956.

[17] R. Paige and R.E. Tarjan. Three partition refinement algorithms. In *SIAM Journal on COMPUTING*, volume 16, 1987.

[18] D. Park. Concurrency and automata on infinite sequences. In *5th GI-Conference on Theoretical Computer Science*, pages 167–183. Springer-Verlag, 1981. LNCS 104.

[19] A. Pnueli. A temporal logic of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.

[20] T.R. Shiple. *Formal analysis of synchronous circuits.* PhD thesis, University of California at Berkeley, 1996.

[21] A. Sistla, M. Vardi, and P. Wolper. The complementation problem for Buchi automata with applications to temporal logic. In *10th International Colloquium on Automata, Languages and Programming*, volume LNCS 194, pages 465–474, 1985.

A Properties of bisimulation

In this section we prove the claims presented in Section 3. Note that whenever two FSMs, M_1 and M_2, are composed, they must satisfy $O_1 \cap \O_2 = \emptyset$.

Lemma 3.1 *Let M be an FSM, and let M_Q be the quotient FSM of M. Let (α, i, α') be an element in R_Q. Then for every state s in α there exists a state s' in α' such that $(s, i, s') \in R$.*

Proof :Assume that $(\alpha, i, \alpha') \in R_Q$. Let $H \subseteq S \times S$ be the maximal bisimulation relation over $M \times M$. The definition of a quotient FSM implies that there are states t, t' in S such that $t \in \alpha$, $t' \in \alpha'$ and $(t, i, t') \in R$. Let s be a state in α. Since s and t are in the same equivalence class, $(t, s) \in H$. Thus, there exists a state s' such that $(s, i, s') \in R$ and $(t', s') \in H$. Since $(t', s') \in H$, t' and s' are in the same equivalence class, $s' \in \alpha'$. □

Proposition A.1 *If M is deterministic, then M_Q is deterministic.*

Lemma 3.3 *M is minimized iff the maximal bisimulation relation over $M \times M$ contains exactly the identity pairs.*

Proof : For the first direction, assume that H is the maximal bisimulation over $M \times M$ and that H contains exactly the identity pairs. Then every equivalence class contains exactly one state. Let M_Q be the quotient FSM of M. We define a function $f : S \to S_Q$ as follows: $f(s) = \alpha$ iff s is in

α. Obviously f is a total and onto function. Since every equivalence class contains exactly one state, f is also one to one. Furthermore, by Lemma 3.1 and the definition of quotient FSM, $(s, i, s') \in R$ iff $(f(s), i, f(s')) \in R_Q$. Thus, M and M_Q are isomorphic and M is minimized.

For the second direction, assume that there is a pair $(s_1, s_2) \in H$ such that $s_1 \neq s_2$. Then s_1, s_2 are in the same equivalence class. Since the equivalence classes partition the states set and at least one class contains more than one state, $|S_Q| < |S|$. Thus M and M_Q are not isomorphic. $\Box$

Lemma A.2 *Let M be an FSM. The identity relation $H_{ID} = \{(s, s) | s \in S\}$ is a bisimulation relation over $M \times M$.*

Proof :

- For every $s_0 \in S_0$, $(s_0, s_0) \in H_{ID}$.

Let (s, s) be a pair in H_{ID}:

- $L(s) = L(s)$.
- Let (s, i, s') be an element in R. Then (s, i, s') is an element in R, and $(s', s') \in H_{ID}$. $\Box$

Lemma A.3 *Let M_Q be the quotient FSM of M, and let H_{QQ} be the maximal bisimulation relation over $M_Q \times M_Q$. Let $H_q = \{(s_1, s_2) | ([s_1], [s_2]) \in H_{QQ}\}$. Then H_q is a bisimulation relation over $M \times M$.*

Proof :

- By the definition of the quotient FSM, for every $s_0 \in S_0$, $[s_0] \in S_{0Q}$. Since $([s_0], [s_0]) \in H_{QQ}$, $(s_0, s_0) \in H_q$.

Let (s_1, s_2) be a pair in H_q.

- $([s_1], [s_2]) \in H_{QQ}$ implies that $L_Q([s_1]) = L_Q([s_2])$ which, implies that $L(s_1) = L(s_2)$.
- Let (s_1, i, s_1') be an element in R. Then $([s_1], i, [s_1']) \in R_Q$. Since $([s_1], [s_2]) \in H_{QQ}$, there exists a class α_2' such that $([s_2], i, \alpha_2') \in R_Q$ and $([s_1'], \alpha_2') \in H_{QQ}$. $([s_2], i, \alpha_2') \in R_Q$, together with Lemma 3.1, implies that there exists a state s_2' such that $(s_2, i, s_2') \in R$. The definition of H_q implies $(s_1', s_2') \in H_q$.

- Similarly, we can prove that for every successor s_2' of s_2 there exists a successor s_1' of s_1 such that $(s_1', s_2') \in H_q$. □

Lemma A.4 *Let M_Q be the quotient FSM of M, and let H_{QQ} be the maximal bisimulation relation over $M_Q \times M_Q$. Then H_{QQ} is the identity relation.*

Proof : Lemma A.2 implies that the identity relation is a bisimulation relation over $M_Q \times M_Q$, and thus it is contained in H_{QQ}. Assume to the contrary that H_{QQ} contains a pair (α_1, α_2) such that $\alpha_1 \neq \alpha_2$. Let s_1 and s_2 be states in α_1 and α_2 respectively and let H_q be the relation defined in Lemma A.3. By the definition of H_q, $(s_1, s_2) \in H_q$. By Lemma A.3, H_q is a bisimulation over $M \times M$, and thus (s_1, s_2) is an element in the maximal bisimulation over $M \times M$. This implies that s_1 and s_2 are in the same equivalence class, a contradiction. □

Corollary A.5 *Every quotient FSM is minimized.*

For the rest of this paper, we will use the term "minimized FSM" for quotient FSM.

Lemma 3.4 *Let M be an FSM and M_Q be the quotient FSM of M with respect to O'. Then M and M_Q are bisimulation equivalent with respect to O'.*

Proof: Let $H_Q \subseteq S \times S_Q$ be the following relation: $H_Q = \{(s, \alpha) | s \text{ is in } \alpha\}$. We prove that H_Q is a bisimulation relation.

- By the definition of the quotient FSM, for every $s_0 \in S_0$, s_0 is in $\alpha_0 \in S_{0Q}$. Similarly, for every $\alpha_0 \in S_{0Q}$ there exists $s_0 \in S_0$ such that $s_0 \in \alpha_0$.

Let (s, α) be a pair in H_Q:

- By the definition of the quotient FSM, $L(s) \cap O' = L_Q(\alpha)$.
- Let (s, i, s') be an element in R. Let α' be the equivalence class of s'. Then by the definition of the quotient FSM, $(\alpha, i, \alpha') \in R_Q$, and by the definition of H_Q, $(s', \alpha') \in H_Q$.
- Let (α, i, α') be an element in R_Q. By Lemma 3.1, there exists a state s' such that $(s, i, s') \in R$ and s' is in α'. Thus $(s', \alpha') \in H_Q$. □

Lemma A.6 *Let M_1 and M_2 be two FSMs that are bisimulation equivalent. Let $H \subseteq S_1 \times S_2$ be a bisimulation relation over $M_1 \times M_2$. Then the relation $H' = \{(s_1, s_1') | \text{there exists } s_2 \in S_2 \text{ such that } (s_1, s_2) \in H \text{ and } (s_1', s_2) \in H\}$ is a bisimulation relation over M_1 with respect to $O_1 \cap O_2$.*

Proof : We prove that H' is a bisimulation relation.

- Since H is a bisimulation relation, for every initial state $s_{01} \in S_{01}$ there exists an initial state $s_{02} \in S_{02}$ such that $(s_{01}, s_{02}) \in H$. Thus, for every initial state $s_{01} \in S_{01}$, $(s_{01}, s_{01}) \in H'$.

For every pair $(s_1, s_1') \in H'$, the following holds:

- Since $(s_1, s_1') \in H'$, there exists a state $s_2 \in S_2$ such that $(s_1, s_2) \in H$ and $(s_1', s_2) \in H$. This implies that $L_1(s_1) \cap (O_1 \cap O_2) = L_2(s_2) \cap (O_1 \cap O_2) = L_1(s_1') \cap (O_1 \cap O_2)$.

- Let (s_1, i, t_1) be a transition in R_1. Since $(s_1, s_1') \in H'$, there exists a state $s_2 \in S_2$ such that $(s_1, s_2) \in H$ and $(s_1', s_2) \in H$. Since H is a bisimulation, there exists a state $t_2 \in S_2$ such that $(s_2, i, t_2) \in R_2$ and $(t_1, t_2) \in H$. This implies that there exists a state $t_1' \in S_1$ such that $(s_1', i, t_1') \in R_1$ and $(t_1', t_2) \in H$. Thus $(t_1, t_1') \in H'$.

- Similarly, for every transition $(s_1', i, t_1') \in R_1$ there exists a transition $(s_1, i, t_1) \in R_1$ such that $(t_1, t_1') \in H'$.

□

Lemma 3.5 *Let M be an FSM and M_Q be the quotient FSM of M with respect to O'. Then M_Q is the smallest (in number of states and transitions) FSM which is bisimulation equivalent to M with respect to O'.*

Proof : First we prove that M_Q is smallest with respect to the number of states. Assume to the contrary that there exists an FSM M' that is bisimulation equivalent to M and smaller than M_Q. Since bisimulation is transitive, M_Q and M' are bisimulation equivalent. Let H be a bisimulation relation over $M_Q \times M'$. Then, there exist two different states s_q and t_q in S_Q that are equivalent to the same state in M'. Let H_q be the relation $H_q = \{(s_q, t_q) | \text{there exists } s' \in S' \text{ such that } (s_q, s') \in H \text{ and } (t_q, s') \in H\}$. By Lemma A.6, H_q is a bisimulation relation. Thus s_q and t_q are bisimulation equivalent, contradicting Lemma A.4.

Next, we prove that M_Q is smallest with respect to number of transitions. Assume to the contrary that there exists an FSM M' that is bisimulation

equivalent to M and smaller than M_Q. Since bisimulation is transitive, M_Q and M' are bisimulation equivalent. Let H be a bisimulation relation over $M_Q \times M'$. Since the number of states in M_Q is not larger than the number of states in M', there exists a pair $(s_q, s') \in H$ such that the number of transitions from s_q is greater than the number of transitions from s'. Since for every transition $(s', i, t') \in R'$ there exists a matching transition from s_q, there exists a transition $(s', i, t') \in R'$ having two transitions (s_q, i, t_{q1}) and (s_q, i, t_{q2}) in R_q which match it. This implies that $(t_{q1}, t') \in H$ and $(t_{q2}, t') \in H$. Let H_q be the relation $H_q = \{(s_q, t_q) | \text{there exists } s' \in S' \text{ such that } (s_q, s') \in H \text{ and } (t_q, s') \in H\}$. By Lemma A.6, H_q is a bisimulation relation. Thus t_{q1} and t_{q2} are bisimulation equivalent, contradicting Lemma A.4. □

A.1 Composition and bisimulation

Next we present some properties of composition and bisimulation.

Lemma A.7 *Let $M = M_1 || M_2$ and let H_1 and H_2 be the maximal bisimulation relations over $M_1 \times M_1$ and $M_2 \times M_2$ with respect to O_1 and O_2 respectively. Let H be the relation $H = \{((s_1, s_2), (t_1, t_2)) | (s_1, t_1) \in H_1,\ (s_2, t_2) \in H_2\}$. Then H is a bisimulation over $M \times M$.*

Proof :

- Let $(s_{10}, s_{20}) \in S_0$. Since $(s_{10}, s_{10}) \in H_1$ and $(s_{20}, s_{20}) \in H_2$, $((s_{10}, s_{20}), (s_{10}, s_{20})) \in H$.

Let $((s_1, s_2), (t_1, t_2))$ be a pair in H.

- By the definition of H, $(s_1, t_1) \in H_1$ and $(s_2, t_2) \in H_2$. Thus $L_1(s_1) = L_1(t_1)$ and $L_2(s_2) = L_2(t_2)$. Since $O_1 \cap O_2 = \emptyset$, $L((s_1, s_2)) = L((t_1, t_2))$.

- Let $((s_1, s_2), i, (s_1', s_2'))$ be an element in R. By the definition of composition, $(s_1, (i \cup L_2(s_2)) \cap I_1, s_1') \in R_1$ and $(s_2, (i \cup L_1(s_1)) \cap I_2, s_2') \in R_2$. Since $(s_1, t_1) \in H_1$ and $L_2(s_2) = L_2(t_2)$, there exists a state t_1' such that $(t_1, (i \cup L_2(t_2)) \cap I_1, t_1') \in R_1$ and $(s_1', t_1') \in H_1$. Similarly, there exists a state t_2' such that $(t_2, (i \cup L_1(t_1)) \cap I_2, t_2') \in R_2$ and $(s_2', t_2') \in H_2$. The definition of composition implies that $((t_1, t_2), i, (t_1', t_2')) \in R$ and by the definition of H, $((s_1', s_2'), (t_1', t_2')) \in H$.

- In a similar way we can show that for every successor (t_1', t_2') of (t_1, t_2) there exists a successor (s_1', s_2') of (s_1, s_2) such that $((s_1', s_2'), (t_1', t_2')) \in H$. $\Box$

Lemma A.8 *If $M = M_1||M_2$ is minimized, then M_1 and M_2 are also minimized.*

Proof : Assume to the contrary that the lemma does not hold. W.l.o.g. assume that M_1 is not minimized. By Lemma 3.3, there are two different states s_1, t_1 such that $(s_1, t_1) \in H_1$. Since every bisimulation relation contains the identity pairs, there exists a state s_2 such that $(s_2, s_2) \in H_2$. Let H be the relation defined in Lemma A.7. Then $((s_1, s_2), (t_1, s_2)) \in H$. By Lemma A.7 , H is a bisimulation relation, and thus it is contained in the maximal bisimulation relation over $M \times M$. This implies that $((s_1, s_2), (t_1, s_2))$ is an element in the maximal bisimulation relation. By Lemma 3.3, M is not minimized, a contradiction. $\Box$

Lemma A.9 *Let $M = M_1||M_2$ and H be a bisimulation over $M \times M$. If $O_2 \cap I_1 = \emptyset$, then the relation $H_1 = \{(s_1, t_1) | s_1, t_1 \in S_1$ and $\exists\, s_2, t_2\ ((s_1, s_2), (t_1, t_2)) \in H\}$ is a bisimulation relation over $M_1 \times M_1$.*

Proof :

- Let $s_{10} \in S_{10}$ and $s_{20} \in S_{20}$. Since $((s_{10}, s_{20}), (s_{10}, s_{20})) \in H$, $(s_{10}, s_{10}) \in H_1$.

Let (s_1, t_1) be a pair of states such that $(s_1, t_1) \in H_1$ and let s_2, t_2 be states such that $((s_1, s_2), (t_1, t_2)) \in H$.

- $((s_1, s_2), (t_1, t_2)) \in H$ implies that $L((s_1, s_2)) = L((t_1, t_2))$. Since $O_1 \cap O_2 = \emptyset$, we conclude that $L_1(s_1) = L_1(t_1)$.

- Let (s_1, i_1, s_1') be an element in R_1. Since $O_2 \cap I_1 = \emptyset$, $I_1 \subseteq I$. Let $i \subseteq I$ be such that $i_1 = i \cap I_1$. Since $O_2 \cap I_1 = \emptyset$, $(i \cup L_2(s_2)) \cap I_1 = i \cap I_1 = i_1$. Let s_2' be a state such that $(s_2, (i \cup L_1(s_1)) \cap I_2, s_2') \in R_2$. Such an s_2' exists by the receptiveness of Moore machines. Then $((s_1, s_2), i, (s_1', s_2')) \in R$. Since $((s_1, s_2), (t_1, t_2)) \in H$, there exists a state (t_1', t_2') such that $((t_1, t_2), i, (t_1', t_2')) \in R$ and $((s_1', s_2'), (t_1', t_2')) \in H$. This implies that $(t_1, i_1, t_1') \in R_1$. By the definition of H_1, $(s_1', t_1') \in H_1$.

- In a similar way we can show that for every successor t_1' of t_1 there exists a successor s_1' of s_1 such that $(s_1', t_1') \in H_1$. $\Box$

Lemma 3.7 *Let M_1 and M_2 be minimized FSMs. If $O_1 \cap I_2 = \emptyset$ and $O_2 \cap I_1 = \emptyset$, then $M = M_1||M_2$ is minimized.*

Proof Let H be the maximal bisimulation over $M \times M$. Assume to the contrary that the lemma does not hold. Then, by Lemma 3.3, there are two different states $(s_1, s_2), (t_1, t_2)$ such that $((s_1, s_2), (t_1, t_2)) \in H$. Since $(s_1, s_2) \neq (t_1, t_2)$, either $s_1 \neq t_1$ or $s_2 \neq t_2$. We assume w.l.o.g. that $s_1 \neq t_1$. Let H_1 be the relation defined in Lemma A.9. By Lemma A.9, H_1 is a bisimulation. By the definition of H_1, $(s_1, t_1) \in H_1$. By Lemma 3.3, M_1 is not minimized, a contradiction. $\Box$