Classes of finite state automata for which compositional minimization is linear time

Piergiulio Katis, N. Sabadini, R.F.C. Walters
Università degli Studi dell' Insubria,
via Valleggio 11,
Como, Italy

25th March 2001

Abstract

We describe a compositional algorithm for the minimization of finite state automata relative to branching bisimulation. With respect to this algorithm we describe an infinite class of automata for which the algorithm is linear time. The class includes the classical dining philospher problem and variants. The proof involves showing that certain families of automata generate finite submonoids of a *bisimulation monoid*. In the case of the dining philosopher the critical fact is that $(F \cdot P)^3 = (F \cdot P)^2$ in the bisimulation monoid, where F is the fork and P the philosopher.

1 Introduction

There is a considerable body of work on finite state compositional model checking (e.g. [3], [12], [11]). One aspect of this work involves involves the use of minimization with respect to different notions of bisimulation ([6]). The method has been used in practical model checking programs (e.g. [5]), with success in a variety of problems. This paper is a contribution to the theoretical analysis of the problems which are tractable using the method.

To concentrate attention we have considered problems which consist of a line of parallel processes each communicating the its two adjacent processes. This is not an essential restriction of our approach – with a richer algebra we could also discuss systems with richer geometry. Our model of process is an automaton with interfaces [8]; the operation of communicating parallel is a restricted product and is denoted $X \cdot Y$; communication is anonymous. We describe what we require of the model in section 2, together with the notion of bisimilarity we will use – essentially branching bisimulation.We introduce also the critical notion of the bisimulation monoid **BisAut**, whose elements are automata modulo bisimulation. The main part of the paper is the analysis of certain finite submonoids of **BisAut**.

Section 3 begins with the description of a compositional algorithm for minimizing an automaton relative to bisimulation – roughly speaking one alternately composes and minimizes. We then introduce the notion of a tractable family of automata – a family such that minimizing $X_1 \cdot X_2 \cdot \dots \cdot X_n$ is linear time for any sequence $X_1, X_2, \dots, X_n, \dots$ of automata from the family. The relation with the bisumulation monoid is that if a family of automata generate a *finite* submonoid of **BisAut** then that family is tractable.

The notion of tractable would have little interest if examples were rare. The main technical content of the paper is to provide an infinite number of tractble families (in section 4), including the classical dining philosopher problem and variants of it. We obtained these examples by a detailed analysis of automata which communicate through locking and unlocking. We believe the analysis of finite submonoids of the bisimulation monoid is a promising direction of research.

2 The automaton model

In this section we define the notion of an automaton with two interfaces, and the composition of two automata. Composition amounts to a product of automata involving synchronization on a common interface which is then hidden. We also consider bisimulation classes of automata.

2.1 Automata

The definition of automaton we give is based on the notion of a reflexive graph. Recall that a reflexive graph is a directed graph in which parallel edges may exists, and which has the following additional structure: for every vertex there is a specified edge beginning and ending at that vertex; this edge is called the reflexive edge at that vertex.

We will consider automata whose edges have two labellings in a set A. We assume that the set A has a distinguished element which we denote '$-$', which is to be thought of as a silent action.

Definition 1 *An* automaton with two interfaces labelled in A*, or just an automaton, is a tuple* (G, g_0, l, r)*, where:*
(i) G is a finite reflexive graph. Its vertices and edges are sometimes referred to as the states *and* transitions *of the automaton. The reflexive edges are also called* idling *transitions.*
(ii) g_0 is a vertex of G such that every vertex of G is reachable from g_0. We call this the initial state *of the automaton.*
(iii) l and r are functions $l, r : \mathsf{Edge}(G) \to A$ such that if α is a reflexive edge then $l(\alpha) = r(\alpha) = -$. For each transition α of G, we call $l(\alpha)$ and $r(\alpha)$ the left *and* right labellings *of the transition respectively. If $l(\alpha) = -$ then we say the transition e is* invisible on the left interface*. If $r(\alpha) = -$ then we say the transition α is* invisible on the right interface*.*

If there is no cause for confusion, we denote the automaton (G, g_0, l, r) merely by G. A *behaviour* of the automaton G is a path (finite or infinite) in G which starts at the initial state.

Example 2 *We describe here two automata which can be used to model the example of the dining philosophers. When we list the transitions of an automaton, we do not bother to list the reflexive edges, and a transition $\alpha : v \to w$ is denoted by $l(\alpha)/r(\alpha) : v \to w$. For this and other examples in this paper, we assume A contains the symbols 'l' and 'u', which stand for 'lock' and 'unlock' respectively.*
The first automaton F models a fork. It has three states $0, 1, 2$ and the following transitions:
$\mathsf{l}/- : 0 \to 1 \quad \mathsf{u}/- : 1 \to 0 \quad -/\mathsf{l} : 0 \to 2 \quad -/\mathsf{u} : 2 \to 0$
The state 0 corresponds to the fork not being used, the state 1 to it being used, or locked, on the left, and the state 2 to it being used on the right. The initial state for F is 0. The second automaton P models a philosopher. It has four states $0, 1, 2, 3$ and the following transitions:
$-/\mathsf{l} : 0 \to 1 \quad \mathsf{l}/- : 1 \to 2 \quad -/\mathsf{u} : 2 \to 3 \quad \mathsf{u}/- : 3 \to 0$
The state 0 corresponds to the philosopher not holding either fork, the state 1 to him holding, or having locked, the fork on his right, the state 2 to him holding both forks, the state 3 to him only holding the fork to his left. The initial state for P is 0.

2.2 Composition of automata

The composite of two automata G and H is a type of synchronizing product. It may be thought of as a parallel operation applied to G and H, where the right interface of G has been glued to the left interface of H.

Definition 3 *Given two automata (G, g_0, l_G, r_G) and (H, h_0, l_H, r_H) we define their* composite *$G \cdot H$ as follows. First we define a reflexive graph $G \times_A H$ called the restricted product of G and H. A vertex of $G \times_A H$ is an arbitrary pair of vertices $(g, h) \in G \times H$. An edge of $G \times_A H$ is a pair of edges $(\alpha, \beta) \in G \times H$ such that $r_G(\alpha) = l_H(\beta)$. The reflexive graph $G \cdot H$ is the subgraph of $G \times_A H$ reachable from the vertex (g_0, h_0), this vertex being the initial state of the composite. The left labelling of an edge $(\alpha, \beta) \in G \cdot H$ is defined to be $l_G(\alpha)$, and its right labelling is $r_H(\beta)$.*

We define a *straight-line system*, or just a system, to be a word $X_1 X_2 ... X_n$ of automata. The *value* of the system $X_1 X_2 ... X_n$ is the automaton $(...(((X_1 \cdot X_2) \cdot X_3) \cdot ...) \cdot X_n)$, which we also denote by $X_1 X_2 ... X_n$.

Example 4 *The system $(FP)^n F$ models a system of n dining philosophers sitting in a row.*

Remark 5 *There are other operations on automata which permit a more general notion of system. For example, in Section 5 a feedback operation is described* *which allows us to describe dining philosophers sitting in a circle. For a more general account of the automaton model described here, where automata with several (rather than just two) interfaces are considered, the reader is referred to [8].*

Composition of automata has a simple interpretation in terms of Arnold and Nivat's synchronized product of finite transition systems ([2]). Consider automata G and H as transition systems $\mathbf{G}$ and $\mathbf{H}$ labelled in the set $A \times A$. The restricted product $G \times_A H$ can be calculated as the synchronized product $(\mathbf{G} \times \mathbf{H})_S$ of transition systems, where the synchronizing set of vectors $S \subseteq (A \times A) \times (A \times A)$ is the set

$$\{((a,b),(b,c)) \mid a,b,c \in A\}.$$

The example of three dining philosophers in a row can be calculated as the synchronized product $(\mathbf{F} \times \mathbf{P} \times \mathbf{F} \times \mathbf{P} \times \mathbf{F} \times \mathbf{P} \times \mathbf{F})_S$, where the synchronizing set of vectors $S \subseteq A^{14}$ is the set

$$\{(a_1,...,a_{14}) \mid a_i = a_{i+1},\ i = 2,4,6,8,10,12\}.$$

The set of behaviours of this transition system starting in the state $(0,0,...,0)$ is precisely the set of behaviours of the value of the system $(FP)^3F$.

In order to compare the automaton model here presented with process algebras, below we write a series of six CCS expressions which model three dining philosophers in a row.

$$\begin{array}{lcl}
Phil & = & a.b.\overline{a}.\overline{b}.Phil \\
Fork & = & \overline{c}.c.Fork + \overline{d}.d.Fork \\
X & = & \mathsf{new}\, m\ (\{m/d\}\, Fork \mid \{m/b\}\, Phil) \\
Y & = & \mathsf{new}\, m\ (\{m/a\}\, X \mid \{m/c\}\, X) \\
Z & = & \mathsf{new}\, m\ (\{m/a\}\, Y \mid \{m/c\}\, X) \\
Dinner & = & \mathsf{new}\, m\ (\{m/a\}\, Z \mid \{m/c\}\, Fork)
\end{array}$$

The CCS expression *Dinner* does not have the same set of behaviours as those behaviours of $(FP)^3F$. The reason for this is that process algebras are an interleaving model of concurrency, while automata models, such as ours, are not. For example, there is no transition out of the CCS expression *Dinner* which corresponds to all three philosophers picking up their right forks simultaneously; but there is such a transition in $(FP)^3F$, namely the one which goes from the state $(0,0,0,0,0,0,0)$ to the state $(0,1,1,1,1,1,1)$.

There is, however, a sub-automaton $\mathsf{Interleave}((FP)^3F)$ of $(FP)^3F$ such that each transition of $\mathsf{Interleave}((FP)^3F)$ corresponds to only one act of synchronization between a philosopher and a fork; and such that every state reachable from $(0,0,0,0,0,0,0)$ in $(FP)^3F$ is also reachable in $\mathsf{Interleave}((FP)^3F)$. The behaviours of $\mathsf{Interleave}((FP)^3F)$ which start from the state $(0,0,0,0,0,0)$ correspond to the behaviours of the CCS expression *Dinner*.

2.3 Bisimulation

In this section we define a notion of bisimulation between automata. It is a slight variation of the notion of branching bisimulation. By the Stuttering Lemma of [7], the bisimulation notion defined here gives the same bisimilarity equivalence relation on automata as does branching bisimulation.

Definition 6 *A* bisimulation *between G and H is a relation $R \subseteq \mathsf{Vertex}(G) \times \mathsf{Vertex}(H)$ such that $(g_0, h_0) \in R$, and if $(g, h) \in R$ then:*
(a) if there is an edge $(a, b) : g \to g'$ (where a and b may equal $-$), then there is a path $h = h_0 \to h_1 \to ... \to h_n$ in H such that (i) the first $n-1$ edges are invisible on both interfaces, (ii) the last edge is labelled (a, b), (iii) for $0 \leq i < n$, $(g, h_i) \in R$ and (iv) (g', h_n).
(b) The symmetric property for h in terms of g.

We denote such a bisimulation by $R : G \Longrightarrow H$. We say two automata G and $H$are *bisimilar* if there exists a bisimulation $R : G \Longrightarrow H$. If G and H are bisimilar, we write $G \sim H$.

The following theorem regards the compositionality of bisimulations. It is straightforward to prove.

Theorem 7 *(i) If $R : G \Longrightarrow H$ and $S : H \Longrightarrow K$ are bisimulations, then the composite relation*

$$R \circ S \subseteq \mathsf{Vertex}(G) \times \mathsf{Vertex}(K)$$

defines a bisimulation $R \circ S : G \Longrightarrow K$.
(ii) The identity relation on $\mathsf{Vertex}(G)$ is a bisimulation $1 : G \Longrightarrow G$.
(iii) If $R : G \Longrightarrow H$ is a bisimulation, then the dual relation $R^\circ : G \Longrightarrow H$ is a bisimulation.
(iv) If $R : F \Longrightarrow G$ and $S : H \Longrightarrow K$ are bisimulations, then the relation

$$R \times S \subseteq (\mathsf{Vertex}(F) \times \mathsf{Vertex}(H)) \times (\mathsf{Vertex}(G) \times \mathsf{Vertex}(K))$$

is a bisimulation $R \cdot S : F \cdot H \Longrightarrow G \cdot K$.

Corollary 8 *Bisimilarity is an equivalence relation on automata. Composition of automata induces a monoid structure on the set* **BisAut** *of equivalences classes of automata under the bisimilarity equivalence relation.*

Proof. The identity for the monoid **BisAut** is the equivalence class of the automaton $(S(A), *, 1_A, 1_A)$, where $S(A)$ is the suspension of the set A: it has one vertex $*$, its set of edges is A, and the reflexive edge is $-$. It is straightforward to check that for any automata G, H, K we have $S(A) \cdot G \sim G \sim G \cdot S(A)$ and $G \cdot (H \cdot K) \sim (G \cdot H) \cdot K$. ■

3 A compositional minimization algorithm

In this section we give an algorithm for calculating the minimization of a system.

First we give the algorithm for constructing the maximal auto-bisimulation of any automaton G. The construction follows the standard method for constructing maximal bisimulations of transition systems (for example, see [2]).

Given a relation R on the states of G define a new relation $E(R)$ as follows: $(x, y) \in E(R)$ if
(i) for any transition $a/b : x \to x'$ in G there is a path $y = y_0 \to y_1 \to y_2 \to \ldots \to y_n$ with the first $n-1$ transitions invisible on the left and the right, with $(x, y_i) \in R$ for $i = 0, 1, .., n-1$, and with $(x', y_n) \in R$, *and*
(ii) the symmetric property for y in terms of x.
Commence with the total relation T - any x is related to any y.Then $E^k(T)$ $(k = 1, 2, 3, \ldots)$ eventually stabilizes as an equivalence relation on the states of G, which is a bisimulation from G to itself. We call this equivalence relation the maximal auto-bisimulation of G.

For any automaton G we define the minimization of G to be the following automaton. Its set of states consist of the equivalence classes of states of G under the maximal auto-bisimulation relation. There is a transition $a/b : [g] \to [h]$ in $\min(G)$ if there is such a labelled transition in G for any representatives of the classes. We denote this automaton by $\min(G)$.

Note that the construction of the minimization of G not only produces an automaton $\min(G)$ but also a bisimulation $\varepsilon_G : G \Longrightarrow \min(G)$ which is actually a function: $\varepsilon_G : g \mapsto [g]$. In Section 5, the functions ε will be used for locating deadlocks in a system.

The crucial property of the construction min is that it has the following compositional properties.

Proposition 9 *For any automata G and H,* $\min(G \cdot H) = \min(\min(G) \cdot \min(H))$. *Furthermore,* $\varepsilon_{G \cdot H} = (\varepsilon_G \cdot \varepsilon_H) \circ \varepsilon_{\min(G) \cdot \min(H)}$.

The compositional minimization algorithm applied to a system $X_1 \ldots X_n$ is simply to calculate min of the value of the system by alternately calculating min and evaluating composition from the left. (We assume the X_i's are already minimal.) Explicitly:

```
initially set $M = X_1$;
for $i = 2$ to $n$
        calculate $M \cdot X_i$
        calculate $\min(M \cdot X_i)$
        set $M$ equal to $\min(M \cdot X_i)$;
$\min(X_1 \ldots X_n) = M$.
```

Proposition 9 guarantees the correctness of this algorithm.

4 Tractable families of automata

In this section we define classes of systems for which the time of the minimization algorithm presented in the previous section grows linearly with respect to the number of automata in the system.

Definition 10 *A family $\mathcal{F}$ of automata is* bounded *by a function $b : \mathbf{N} \to \mathbf{N}$ if, for any $X_1, ..., X_n \in \mathcal{F}$, the number of states* $\min(X_1...X_n)$ *is less than $b(n)$.*

Any finite family $\mathcal{F}$ is bounded by a function: namely the exponential function $b(n) = c^n$, where c is the maximum number of states that a member of $\mathcal{F}$ may have. With regards to model checking, we are interested in families that are bounded by polynomial functions.

As an example bounded by a linear function, consider the automaton B with two states $0, 1$ and two (non-reflexive) transitions $m/- : 0 \to 1$ and $-/m : 1 \to 0$. This models a buffer, or message passer, of capacity one: the state 0 corresponds to the buffer being empty, and the state 1 to the buffer being full. It can be shown that $\min(B^n)$ has $n + 1$ states: the intuition is that B^n is a model of a buffer of capacity n. So the family $\{B, B^2, ...\}$ is bounded by $n + 1$.

Proposition 11 *Suppose the submonoid $\langle\mathcal{F}\rangle$ of* **BisAut** *generated by the bisimulation equivalence classes of the members of the family $\mathcal{F}$ is finite. Then $\mathcal{F}$ is bounded by a constant function.*

Proof. Take the constant to be the maximum number of states that a minimal automaton in any equivalence class of $\langle\mathcal{F}\rangle$ may have. ■

Definition 12 *A family $\mathcal{F}$ of automata is tractable if there exist a linear function $b : \mathbf{N} \to \mathbf{N}$ such that, for any $X_1, ..., X_n \in \mathcal{F}$, the time taken to apply the minimization algorithm to the system $X_1...X_n$ is less than $b(n)$.*

Proposition 13 *If $\mathcal{F}$ is bounded by a constant function then it is tractable.*

Proof. Suppose $\mathcal{F}$ is bounded by a constant. Then, for any G in $\langle\mathcal{F}\rangle$ and any H in $\mathcal{F}$, the number of states of $\min(G)\cdot H$ is bounded by a constant. Let t be the maximum time taken for the algorithm to compute the composite $\min(G) \cdot H$, for any G in $\langle\mathcal{F}\rangle$ and any H in $\mathcal{F}$; and let s be the maximum time taken for the algorithm to compute $\min(\min(G) \cdot H)$. Clearly there exist constants c and d such that such that, for any $X_1, ..., X_n \in \mathcal{F}$, the time taken to apply the minimization algorithm to the system $X_1...X_n$ is less than $(t+s+c)\times(n-1)+d$. ■

Corollary 14 *If the submonoid $\langle\mathcal{F}\rangle$ of* **BisAut** *generated by the bisimulation equivalence classes of the members of the family $\mathcal{F}$ is finite, then $\mathcal{F}$ is tractable.*

4.1 Examples

In this section we consider some examples of finite submonoids of **BisAut**, and thus of tractable families of automata. We study classes of, what we call, 'lock-unlock' automata. One of these classes includes the example of the dining philosopher.

1. Let Z be the automaton with one state, and whose only transition is the reflexive edge. This automaton has the property that $ZGZ \sim Z$, for any G. Immediate consequences of this are that $(ZG)(ZH) \sim ZH$ and $(GZ)(HZ) \sim GZ$, for any G and H. So any finite family comprising automata of the form ZG or HZ generate a finite submonoid of **BisAut** and is thus tractable. The automaton Z may be thought of as one which blocks an interface, and thus prohibits the exponential growth of state in model checking. For example, if we wanted to check a property such as deadlock of the system $ZG_1ZG_2Z...ZG_nZ$, it is intuitively clear we only need check the automata $ZG_1Z, ..., ZG_nZ$ separately for this property; and the time taken for such a process will increase linearly with respect to n.

2. Let N be the automaton with two states $0, 1$ and transitions $m/- : 0 \to 1$, $m/- : 1 \to 1$, $-/m : 1 \to 1$, $m/m : 1 \to 1$ and $-/m : 1 \to 0$. We call this automaton a non-deterministic buffer or a buffer of an undetermined capacity. It has the property that $N^2 \sim N$; that is, its equivalence class is idempotent in **BisAut**. So the family containing only N is tractable.

3. Let F be the fork automaton of Example 2. We call X a lock-unlock automaton if it has the following property: from every state of FXF there is a path to the initial state. The idea is that X is a process which wants to complete a 'cycle', and to do so it may require to lock and unlock a resource on its left and one on its right, perhaps many times and perhaps non-deterministically. A simple example of a lock-unlock automaton is the philosopher P, described in Example 2. We will define four infinite classes $\mathcal{F}_1$, $\mathcal{F}_2$, $\mathcal{F}_3$ and $\mathcal{F}_4$ of lock-unlock automata (one of which will include P) such that for X in any of these classes $(FX)^2 \sim (FX)^3$. To do so we require the following definitions.

We say a path in G is invisible if every transition in the path is invisible on the left and on the right.

A state $(2, x, 2) \in FX\ F$ is said to be *bound* if there does not exist a $y \in X$ and an invisible path in FX from $(2, x)$ to $(0, y)$. In other words, $(2, x, 1)$ is bound if in order for X to unlock its left fork X must gain access to its right fork.

Similarly, a state $(1, x, 1) \in FXF$ is said to be *bound* if there does not exist a $y \in X$ and an invisible path in XF from $(x, 1)$ to $(y, 0)$.

A state $(f, x, 2) \in FXF$ is said to be *free* if there is a $y \in X$ and an invisible path in FX from (f, x) to $(0, y)$, and if there is no invisible path in FX from (f, x) to $(2, z)$, where $(2, z, 2)$ is a bound state. Note that this means $f \neq 1$.

Similarly, we say a state $(1, x, f) \in FXF$ is *free* if there is a $y \in X$ and an invisible path in XF from (x, f) to $(y, 0)$, and if there is no invisible path in XF from (x, f) to $(z, 1)$, where $(1, z, 1)$ is a bound state. Note that this means $f \neq 2$.

A state $(f, x, 2) \in FXF$ is said to be *free-bound* if one of the following two conditions hold: (i) $(f, x, 2)$ is bound; or (ii) there is a $y \in X$ and an invisible path in FX from (f, x) to $(0, y)$, and there is no invisible path in FX from (f, x) to (g, z), where $(g, z, 2)$ is a free state. So a free-bound state is bound or can become bound.

Similarly, we say a state $(1, x, f) \in FXF$ is *free-bound* if one of the following two conditions hold: (i) $(1, x, f)$ is bound; or (ii) there is a $y \in X$ and an invisible path in XF from (x, f) to $(y, 0)$, and there is no invisible path in XF from (x, f) to (z, g), where $(1, z, g)$ is a free state.

We now define the four classes of lock-unlock automata.

1. $X \in \mathcal{F}_1$ if and only if all states of the forms $(1, x, 1), (1, x, 0), (0, x, 2), (2, x, 2) \in FXF$ are free. So if X performs a lock on the left (resp. right), it is able to follow that action with an unlock on the left (resp. right).
2. $X \in \mathcal{F}_2$ if and only if all states of the forms $(1, x, 1), (1, x, 0), (0, x, 2), (2, x, 2) \in FXF$ are free-bound.
3. $X \in \mathcal{F}_3$ if and only if all states of the forms $(1, x, 1), (1, x, 0) \in FXF$ are free and all states of the form $(0, x, 2), (2, x, 2) \in FXF$ are free-bound.
4. $X \in \mathcal{F}_4$ if and only if all states of the forms $(1, x, 1), (1, x, 0) \in FXF$ are free-bound and all states of the form $(0, x, 2), (2, x, 2) \in FXF$ are free.

An example of an automaton in $\mathcal{F}_1$ is F; Z is also an example. An example of an automaton in $\mathcal{F}_4$ is P. An example of an automaton in $\mathcal{F}_3$ is P°, the automaton obtained by swapping the left and right labellings of P; P° may be thought of as a philosopher who wants to pick up his left fork first. An example of an automaton in $\mathcal{F}_2$ is $P + P^\circ$, the automaton formed by taking the disjoint union of P and P° and then smashing their initial states together (it has 7 states and 8 non-reflexive edges).

The following is an example of a lock-unlock automaton P' which is not in one of the above four classes. P' has six states 0,1,2,3,4,5 and the following transition:

$$\begin{array}{lll} l/- : 0 \to 1 & -/l : 1 \to 2 & -/u : 2 \to 3 \\ u/- : 3 \to 4 & -/l : 4 \to 5 & -/u : 5 \to 0 \end{array}$$

The reason why it does not fit into one of these classes is that the state $(1, 1, 2) \in FP'F$ is bound and the state $(0, 4, 2) \in FP'F$ is free. As an aside we note that $(FP')^2 \quad (FP')^3$ but $(FP')^3 \sim (FP')^4$.

Let $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2 \cup \mathcal{F}_3 \cup \mathcal{F}_4$.

Claim 15 *Any finite subset of $F\mathcal{F} = \{FX \mid X \in \mathcal{F}\}$ is tractable.*

We prove that if $S \subset F\mathcal{F}$ is finite then the submonoid $\langle S \rangle$ of **BisAut** is finite.

Lemma 16 *If X and Y are lock-unlock automata then XFY is a lock-unlock automaton.*

Proof. Consider a state $(f, x, g, y, h) \in FXFYF$. Either X or Y has, or has access to, the middle fork. Assume X does. Then there is a path from (f, x, g, y, h) to $(0, 0, 0, y, h)$. There is a path from $(0, 0, 0, y, h)$ to $(0, 0, 0, 0, 0)$. ■

We define a monoid structure on the four element set $\{1, 2, 3, 4\}$ as follows:

$*$	1	2	3	4
1	1	1	1	1
2	1	2	3	4
3	1	3	3	1
4	1	4	1	4

Lemma 17 *If $X \in \mathcal{F}_i$ and $Y \in \mathcal{F}_j$ then $XFY \in \mathcal{F}_{i*j}$.*

Proof. First consider $i = 1$ and any $j = 1, 2, 3, 4$. Consider a state $(2, x, f, y, 2) \in FXFYF$. If $f \neq 2$ then, since X is an lock-unlock automaton, there is an invisible path in FXF from $(2, x, f)$ to $(0, 0, 0)$, and thus an invisible path in $FXFYF$ from $(2, x, f, y, 2)$ to $(0, 0, 0, y, 2)$. Suppose $f = 2$. Since $(2, x, f) \in FXF$ is free, there is an invisible path in FX from $(2, x)$ to $(0, x)$, and thus an invisible path in $FXFYF$ from $(2, x, f, y, 2)$ to $(0, x, f, y, 2)$. Since we have not placed any restrictions on $(x, f, y) \in XFY$, the states $(2, x, f, y, 2)$ are free as they can never become bound. Consider a state of the form $(1, x, f, y, 1)$. If $f \neq 1$ then there is an invisible path in FYF from $(f, y, 1)$ to $(0, 0, 0)$, since Y is a lock-unlock automaton; and thus an invisible path from $(1, x, f, y, 1)$ to $(1, x, 0, 0, 0)$. If $f = 1$ then, since $(1, x, 1) \in FXF$ is free, there is an invisible path in XF from $(x, 1)$ to $(x', 0)$, and thus an invisible path in $FXFYF$ from $(2, x, f, y, 2)$ to $(0, z, 0, y, 2)$, which brings us to the previous case $f \neq 1$. Since we have not placed any restrictions on $(x, f, y) \in XFY$, the states $(2, x, f, y, 2)$ are free, as they can never become bound. This proves that if $X \in \mathcal{F}_1$ and $Y \in \mathcal{F}_j$ then $XFY \in \mathcal{F}_1$. A symmetric argument shows that if $X \in \mathcal{F}_i$ and $Y \in \mathcal{F}_1$ then $XFY \in \mathcal{F}_1$.

We now consider the case $X \in \mathcal{F}_2$ and $Y \in \mathcal{F}_3$. Arguments similar to those given in the previous paragraph can be used to show any state $(1, x, f, y, 1)$ is free. We need to show that there is an invisible path from any state $(2, x, f, y, 2)$ to a bound state. If $f \neq 2$ then there is an invisible path to a state $(2, x', 2, y, 2)$, since X is a lock-unlock automaton. So assume $f = 2$. Since $X \in \mathcal{F}_2$ there is an invisible path in FX from $(2, x)$ to $(2, x')$, where $(2, x', 2) \in FXF$ is bound.

Since $Y \in \mathcal{F}_3$ there is an invisible path in FY from $(2,y)$ to a state $(2,y',2) \in FYF$ which is bound. Thus there is an invisible path in $FXFYF$ from any $(2,x,f,y,2)$ to a bound state $(2,x',2,y',2)$. So $XY \in \mathcal{F}_3$. The remaining cases can be argued in similar ways. ■

Lemma 18 *For $i = 1,2,3,4$, if $X, Y \in \mathcal{F}_i$ then $FXF \sim FYF$.*

Proof. For any $X \in \mathcal{F}$, we partition the states of FXF into eight sets $U_1, U_2, U_3, U_4, U_5, U_6, U_7, U_8$ (some of which may be empty).
1. For all x, $(f,x,g) \in U_1$ if $f \in \{0,2\}$ and $g \in \{0,1\}$.
2. For all x, $(1,x,2) \in U_2$.
3. $(f,x,2) \in U_3$ if it is free.
4. $(f,x,2) \in U_4$ if it is free-bound and not bound.
5. $(f,x,2) \in U_4$ if it is bound.
6. $(1,x,f) \in U_5$ if it is free.
7. $(1,x,f) \in U_6$ if it is free-bound and not bound.
8. $(1,x,f) \in U_6$ if it is bound.

For $X \in \mathcal{F}$, this partition is well defined. Notice that, by definition of the $\mathcal{F}_i$'s, for each X either U_3 or $(U_4 \cup U_5)$ will be empty, and either U_6 or $(U_7 \cup U_8)$ will be empty. With this last observation in mind, it is easy to check that the equivalence relation induced by this partition is an auto-bisimulation (in fact, maximal). Furthermore, it is straightforward to show that if $X, Y \in \mathcal{F}_i$ then the minimal automata of X and Y are the same. ■

We can use the previous results to deduce that $(FX)^2 \sim (FX)^3$ for $X \in \mathcal{F}_i$. Lemma 17, together with the four element multiplication table above it, implies that $XFX \in \mathcal{F}_i$. Lemma 18 now implies that $FXFXF \sim FXF$. Hence, the desired result.

Lemma 19 *If $S \subset F\mathcal{F}$ has s elements, then the number of elements of the submonoid $\langle S \rangle$ of* **BisAut** *is less than or equal to $4 \times s + 1$.*

Proof. Let F_1, F_2, F_3, F_4 be the four minimal automata such that for any $X \in \mathcal{F}_i$, $FXF \sim F_i$. By the previous two Lemmas, for any $X_1, ..., X_n \in \mathcal{F}$, $FX_1F...FX_nF \sim F_i$, for some i. So any element of $\langle S \rangle$ equals the equivalence class of F_iX, for some $i \in \{1,2,3,4\}$ and X such that $FX \in S$. (The additional 1 in the sum comes from counting the identity of the monoid.) ■

This completes the proof of the claim.

We give explicit descriptions of the four minimal automata F_1, F_2, F_3, F_4. From the proof of Lemma 18, it is clear that F_1 has 4 states (corresponding to the cases 4,5,7 and 8 being empty), F_2 has 6 states (cases 3 and 6 empty), and F_3 (cases 3,7 and 8 empty) and F_4 (cases 4,5 and 6 empty) have 5 states. We describe the minimal automata as sub-automata of the automaton Q which has

8 states (each state corresponds to one of the cases of the partition described in the proof of Lemma 18) and the following transitions:

$l/l : 1 \to 2$ $\quad -/l : 1 \to 3$ $\quad -/l : 1 \to 4$ $\quad -/l : 1 \to 5$ $\quad l/- : 1 \to 6$ $\quad l/- : 1 \to 7$
$l/- : 1 \to 8$ $\quad u/u : 2 \to 1$ $\quad u/- : 2 \to 3$ $\quad u/- : 2 \to 4$ $\quad -/u : 2 \to 6$ $\quad -/u : 2 \to 7$
$-/u : 3 \to 1$ $\quad l/- : 3 \to 2$ $\quad l/u : 3 \to 6$ $\quad l/u : 3 \to 7$ $\quad -/u : 4 \to 1$ $\quad l/- : 4 \to 2$
$-/- : 4 \to 5$ $\quad l/u : 4 \to 6$ $\quad l/u : 4 \to 7$ $\quad -/u : 5 \to 1$ $\quad u/- : 6 \to 1$ $\quad -/l : 6 \to 2$
$u/l : 6 \to 3$ $\quad u/l : 6 \to 4$ $\quad u/- : 7 \to 1$ $\quad -/l : 7 \to 2$ $\quad u/l : 7 \to 3$ $\quad u/l : 7 \to 4$
$-/- : 7 \to 8$ $\quad u/- : 8 \to 1$

F_1 is the sub-automaton of Q with states 1,2,3 and 6. F_2 is the sub-automaton of Q with states 1,2,4,5,7 and 8. F_3 is the sub-automaton of Q with states 1,2,4,5 and 6. F_4 is the sub-automaton of Q with states 1,2,3,7 and 8.

Remark 20 *The above calculations may be interpreted as follows. Lock-unlock automata form a submonoid $\mathcal{LU}$ of* **BisAut** *comprising bisimulation classes of automata of the form FX where X is a lock-unlock automaton. Though the submonoid $\langle F\mathcal{F}_1 \cup F\mathcal{F}_2 \cup F\mathcal{F}_3 \cup F\mathcal{F}_4 \rangle$ of $\mathcal{LU}$ is infinite, any finite subset S of it generates a finite submonoid.*

5 Detecting deadlocks

In this section, we indicate how the minimization algorithm can be used to locate all the deadlock states of a system; note that this asks more than just determining whether or not a system has a deadlock. We consider the example of the dining philosophers in a circle.

Definition 21 *A deadlock of an automaton G is a state g such that the only transitions with source g are of the form $-/- : g \to g$.*

Proposition 22 *Suppose $q : G \Longrightarrow H$ is a bisimulation which is a function. If $g \in G$ is a deadlock then $q(g)$ is a deadlock.*

Proof. Suppose g is a deadlock. The only transitions out of $q(g)$ can be of the form $-/- : q(g) \to h$. In this case, there exists a path $g = g_0 \to g_1 \to ... \to g_n$ in G such that (i) all the edges are invisible, (iii) for $0 \leq i < n$, $f(g_i) = f(g)$ and (iv) $f(g_n) = h$. Since G is a deadlock all the g_i's equal g. Thus $h = q(g)$. And so $q(g)$ is a deadlock. ∎

The above proposition does not hold for arbitrary bisimulations. Also, it is not the case that $q(g)$ is a deadlock implies g is a deadlock.

Recall that the bisimulations $\varepsilon_G : G \Rightarrow \min(G)$ defined in Section 3 are all functions. To find all the deadlocks in a system $X_1 ... X_n$, we apply a modification of the compositional minimization algorithm of Section 3 made by recording all the bisimulations ε made along the way, search $\min(X_1...X_n)$ for deadlocks, and then, for each deadlock $g \in \min(X_1...X_n)$, search for deadlocks in the inverse images $\varepsilon^{-1}_{X_1...X_n}(g)$. Explicitly:

```
initially set $M = X_1$ and $\varepsilon = 1_{X_1}$;
for $i = 2$ to $n$
        calculate $M \cdot X_i$
        calculate $\min(M \cdot X_i)$
        calculate the function $\varepsilon_{M \cdot X_i}$
        set $M$ equal to $\min(M \cdot X_i)$
        set $\varepsilon$ equal to the word $(\varepsilon \cdot 1_{X_i}) \circ \varepsilon_{M \cdot X_i}$;
search $M$ for deadlocks;
for each deadlock $g \in M$
        calculate $\varepsilon^{-1}(g)$
        search $\varepsilon^{-1}(g)$ for deadlocks.
```

The role of ε in this algorithm, and in particular line 7, needs explanation. In this first place, the algorithm iteratively constructs a word ε built out of the operations $\cdot$ and $\circ$, where each term in the word is an identity function 1_{X_i} or a bisimulation $\varepsilon_{M \cdot X_i}$. If we were to evaluate this word, we would obtain the bisimulation $\varepsilon_{X_1...X_n} : X_1...X_n \Rightarrow \min(X_1...X_n)$. We do not want to evaluate the function $\varepsilon_{X_1...X_n}$ in general, since we are only interested in inverse images of deadlocks under this function. This is what is calculated in the penultimate line of the algorithm.

We call this algorithm the ε-minimization algorithm. The correctness of this algorithm is guaranteed by Proposition 9.

The notion of a system in a straight-line is too simple to even model the classical problem of the dining philosophers. As stated earlier, there are other operations on automata, and the general theory and constructions described in this paper – including the minimization algorithm – pass to this more general setting. To give the example of the dining philosopher, we introduce another operation:

Definition 23 *If G is an automaton then the feedback of G, denoted* $\mathsf{Fb}(G)$, *is the reflexive graph defined as follows. Define $G^\dagger$ to be the reflexive graph with the same vertices as G, but only those edges for which the left labelling equals the right labelling.* $\mathsf{Fb}(G)$ *is the subgraph of $G^\dagger$ reachable from the initial state of G.*

The classical problem of n dining philosophers sitting around a table is modelled by the system $\mathsf{Fb}((FP)^n)$.

The notion of bisimulation becomes trivial at the level of (unlabelled) reflexive graphs. To see this, notice that any automaton G for which every transition is invisible is bisimilar to Z, the automaton with one state and no non-reflexive edges. Thus general bisimulations between unlabelled automata are not helpful from the point of view of detecting deadlock. However we have the following result. (The proof is the same as that for Proposition 22.)

Proposition 24 *Suppose $q : G \Rightarrow H$ is a bisimulation which is a function. If $g \in \mathsf{Fb}(G)$ is a deadlock in $\mathsf{Fb}(G)$ then $q(g)$ is a deadlock in $\mathsf{Fb}(H)$.*

The relevance of this proposition for model checking is that $\mathsf{Fb}(H)$ may not be as trivial as Z, and, hopefully, the inverse image of deadlocks in $\mathsf{Fb}(H)$ will be small. For example, to find the deadlocks in $\mathsf{Fb}((FP)^n)$, we begin by applying the first 8 lines of the ε-minimization algorithm to $(FP)^n$ (that is, up to the point of looking for deadlocks). Notice that $M = \min((FP)^n) = F_4P$ (the 5 state automaton F_4 is calculated in the previous section). We then calculate $\mathsf{Fb}(M)$, search it for deadlocks, and finally, for each deadlock in $\mathsf{Fb}(M)$, search $\varepsilon^{-1}(g)$ for deadlocks. The automaton $\mathsf{Fb}(\min((FP)^n))$ has 3 states, one of which is a deadlock. The inverse image under ε of this deadlock is a single state, namely $(1, 1, ..., 1, 1)$.

6 Final Remarks

We have presented a compositional automata model, in which a system is represented as an expression of automata. Algorithms for computing the minimization of a system and for locating all the deadlocks of a system were described. Some classes of systems for which the algorithm is linear were identified by considering part of the algebraic structure of the monoid **BisAut** of bisimulation equivalence classes of automata. In particular, we considered certain classes of, what we call, lock-unlock automata.

In Remark 20, it was noted that lock-unlock automata form a submonoid $\mathcal{LU}$ of **BisAut**. There is another algebraic structure related to these automata.

Let $[G]$ denote the bisimulation equivalence class of an automaton G. Consider the semigroup $\mathcal{S}$ whose elements are equivalence classes $[FXF]$ where X is a lock-unlock automaton. Composition for the semigroup is $[FXF] * [FYF] = [FXFYF]$. The four classes $\mathcal{F}_1$, $\mathcal{F}_2$, $\mathcal{F}_3$ and $\mathcal{F}_4$ of lock-unlock automata defined in Section 4.1 represent a four element sub-semigroup of $\mathcal{S}$ (which happens to be a monoid).

One way of viewing this semigroup is to notice that any FXF may be viewed as a span of reflexive graphs $F \longleftarrow FXF \rightarrow F$, where the left (resp. right) leg of the span is a projection onto the left (resp. right) fork. Spans form a compact closed bicategory. (For more on spans of reflexive graphs, see [8].) The composition of the spans $F \longleftarrow FXF \rightarrow F$ and $F \longleftarrow FYF \rightarrow F$ gives the span $F \longleftarrow FXFYF \rightarrow F$. There is a notion of bisimulation of spans which respect composition of spans. Quotienting out the bicategory of spans by bisimilarity gives a compact closed category **Span**. The semigroup $\mathcal{S}$ lives in the category **Span** as a family of endomorphisms on the object F.

A similar construction on bicategories of spans of transition systems was used in [4] to give a categorical description of Abramsky's categories of synchronous and asynchronous processes [1].

Two direct generalizations of the results in this paper could be pursued. The first is to attempt to give a more detailed analysis of lock-unlock automata with respect to the minimization algorithm, which would involve a classification of all the finite sub-semigroups of $\mathcal{S}$. Another would be to extend the results of the paper to accommodate a more general notion of system - that is, systems that are not confined to a straight line. This would require dealing with algebraic structures richer than monoids, namely compact closed categories.

References

[1] S. Abramsky. Interaction categories. In Theory and Formal Methods Workshop. Springer Verlag, 1993.

[2] A. Arnold, Finite transition systems, Prentice Hall, 1994.

[3] E Clarke, D Long, K. McMillan. Compositional model checking. In Proceedings of the Fourth Annual IEEE Symposium on Logic in Computer Science, pp. 353-362, 1989.

[4] J. Cockett, D. Spooner. Categories for synchrony and asynchrony. Electronic Notes in Theoretical Computer Science. 1:25 (electronic), 1995.

[5] J Fernandez. Aldébaran: Manuel de l'utilisateur. Technical Report, LGI-IMAG Grenoble, 1988.

[6] J Fernandez, L. Mounier. A tool set for deciding behavioural equivalences. Preprint.

[7] R. van Glabbeek, P. Weijland, Branching time and asbtraction in bisimulation semantics, in: JACM 43(3), 1996, pp. 555-600.

[8] P. Katis, N. Sabadini, R.F.C. Walters, Span(Graph): A categorical algebra of transition systems, Proceedings Algebraic Methodology and Software Technology, volume 1349 of Lecture Notes in Computer Science, 307–321, Springer Verlag, 1997.

[9] P. Katis, N. Sabadini, R.F.C. Walters, On the algebra of systems with feedback and boundary, Rendiconti del Circolo Matematico di Palermo Serie II, Suppl. 63: 123-156, 2000.

[10] P. Katis, N. Sabadini, R.F.C. Walters, A formalization of the IWIM model, Coordination Languages and Models, volume 1906 Lecture Notes in Computer Science, 267–283, Springer Verlag, 2000.

[11] B. Steffen S. Graf, G Lüttgen. Compositional minimization of finite state systems. International Journal of Formal Aspects of Computing, Vol 8, pp 607-616, 1996.

[12] W Yeh, M Young. Compositional reachability analysis using process algebra. In Proc. Symposium on Testing, Analysis, and Verification (TAV4), pp 178-187, New York, Oct 1991. ACM SIGSOFT.