Acta Cybernetica **16** (2004) 367–384.

A Uniform Approach to Test Computational Complementarity

Elena Calude, Bruce Mills, and Lan Mills*

Abstract

Studies of computational complementarity properties in finite state interactive automata may shed light on the nature of both quantum and classical computation. But, complementarity is *difficult* to test even for small-size automata. This paper introduces the concept of an observation graph of an automaton which is used as the main tool for the design of an algorithm which tests, in a uniform manner, two types of complementarity properties. Implementations have been run on a standard desktop computer examining all 5-state binary automata.

1 Two Computational Complementarity Principles

Building on Moore's "Gedanken" experiments, in [15, 14] complementarity was modeled by means of finite automata. Two new computational complementarity principles have been introduced and studied in [3, 6, 5, 4, 2] using Moore's automata.

To understand Moore's approach it is enough, at this stage, to say that the machines we are going to consider are *finite* in the sense that they have a finite number of states, a finite number of input symbols, and a finite number of output symbols. Such a machine has a strictly deterministic behaviour: the current state of the machine depends only on its previous state and previous input; the current output depends only on the present state. A (simple) Moore experiment can be described as follows: a copy of the machine will be experimentally observed, i.e. the experimenter will input a finite sequence of input symbols to the machine and will observe the sequence of output symbols. The correspondence between input and output symbols depends on the particular chosen machine and on its initial state. The experimenter will study the sequences of input and output symbols and will try to conclude that "the machine being experimented on was in state q at the beginning of the experiment".[1] Moore's experiments have been studied

*Institute for Information and Mathematical Sciences, Massey University at Albany, Private Bag 102904, NSMC, Auckland, New Zealand.
Email: {E.Calude,B.I.Mills,L.Mills}@massey.ac.nz

[1]This is often referred to as a *state identification experiment*.

from a mathematical point of view by various researchers, notably by Ginsburg [9], Chaitin [7], Conway [8], and Brauer [1]. A comprehensive survey on testing finite state machines is presented in [11].

In what follows we are going to use two non-equivalent concepts of computational complementarity based upon modeling finite automata (see [3]). Informally, they can be expressed as follows. Consider the class of all elements of reality[2] and consider the following properties.

A Any two distinct elements of reality can be mutually distinguished by a suitably chosen measurement procedure.

B For any element of reality, there exists a measurement which distinguishes between this element and all the others. That is, a distinction between any one of them and all the others is operational.

C There exists a measurement which distinguishes between any two elements of reality. That is, a single pre-defined experiment exists to distinguish between an arbitrary pair of elements of reality. (Classical case.)

Complementarity corresponds to the following cases:

CI *Property* **A** *but not property* **B** *(and therefore not* **C***):* The elements of reality can be mutually distinguished by experiments, but one of these elements cannot be distinguished from all the other ones by any single experiment.

CII *Property* **B** *but not property* **C***:* Any element of reality can be distinguished from all the other ones by a single experiment, but there does not exist a single experiment which distinguishes between any pair of distinct elements.

2 Moore Automata

A finite deterministic automaton consists of a finite set of states and a set of transitions from state to state that occur on input symbols chosen from some fixed alphabet. For each symbol there is exactly one transition out of each state, possible back to the state itself. So, formally, a *finite automaton* consists of a finite set Q of states, an input alphabet Σ, and a transition function $\delta : Q \times \Sigma \to Q$. Sometimes a fixed state, say 1, is considered to be the *initial state*, and a subset F of Q denotes the *final states*. A *Moore automaton* is a finite deterministic automaton having an *output function* $f : Q \to O$, where O is a finite set of output symbols. At each time the automaton is in a given state q and is continuously emitting the output $f(q)$. The automaton remains in state q until it receives an input signal σ, when it assumes the state $\delta(q, \sigma)$ and starts emitting $f(\delta(q, \sigma))$. In this paper we are going to concentrate on the case of automata on a binary alphabet $\Sigma = \{0, 1\}$ having $O = \Sigma$. So, from now on, a Moore automaton will be just a triple $M = (Q, \delta, f)$. Let Σ^* be the set of all finite sequences (words) over the alphabet Σ, including the empty word

[2]The terms "elements of reality", "properties", and "observables" will be used as synonyms.

ϵ (the neutral element in the semigroup of string concatenation); by Σ^+ we denote $\Sigma^* \setminus \{\epsilon\}$. The transition function δ can be extended to a function $\overline{\delta} : Q \times \Sigma^* \to Q$, as follows: $\overline{\delta}(q,\epsilon) = q, \overline{\delta}(q,\sigma w) = \overline{\delta}(\delta(q,\sigma),w), \forall q \in Q, \sigma \in \Sigma, w \in \Sigma^*$. The output produced by an experiment started in state q with input sequence $w \in \Sigma^*$ is described by $E(q,w)$, where E is the function $E : Q \times \Sigma^* \longrightarrow \Sigma^*$ defined as follows: $E(q,\varepsilon) = f(q)$, $E(q,\sigma w) = f(q)E(\delta(q,\sigma),w), q \in Q, \sigma \in \Sigma, w \in \Sigma^*$, and $f : Q \longrightarrow O(= \Sigma)$ is the output function. Consider, for example, Moore's automaton, in which $Q = \{1,2,3,4\}$, $\Sigma = \{0,1\}$. The transition is given by the following tables

q	σ	$\delta(q,\sigma)$
1	0	4
1	1	3
2	0	1
2	1	3

q	σ	$\delta(q,\sigma)$
3	0	4
3	1	4
4	0	2
4	1	2

Table 1.

and the output function is defined by $f(1) = f(2) = f(3) = 0$, $f(4) = 1$. The following graphical representation will be consistently used in what follows:

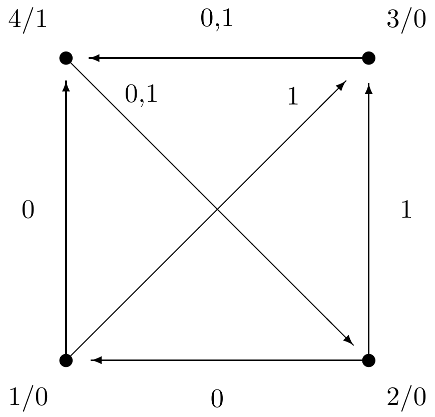

Figure 1.

The experiment starting in state 1 with input sequence 000100010 leads to the output 0100010001. Indeed, $E(1,000100010) = f(1)f(4)f(2)f(1)f(3)f(4)f(2)f(1)f(3)f(4) = 0100010001$.

From a mathematical point of view properties **A, B, C** can be expressed as follows. Let $M = (Q, \delta, f)$ be a Moore automaton. Following Moore [13] we shall say that a state q is "indistinguishable" from a state q' (with respect to M) if every experiment performed on M starting in state q produces the same outcome as it would starting in state q'. Formally, $E(q, x) = E(q', x)$, for all words $x \in \Sigma^+$. A pair of states will be said to be "distinguishable" if they are not "indistinguishable".

- The automaton M has property **A** if every pair of different states of M are distinguishable, i.e. for every distinct states q, q' there exists a word $w \in \Sigma^+$ (depending upon q, q') such that $E(q, w) \neq E(q', w)$. This is simply the assertion that the automaton is minimal.

- The automaton M has property **B** if every state of M is distinguishable from any other distinct state, i.e. for every state q there exists a word $w \in \Sigma^+$ (depending upon q) such that $E(q, w) \neq E(q', w)$, for every state q' distinct from q.

- The automaton M has property **C** if there exists an experiment distinguishing between each different states of M, i.e. there exists a word $w \in \Sigma^+$ such that $E(q, w) \neq E(q', w)$, for every distinct states q, q'.

Of course, **C** implies **B**, which, in turn, implies **A**; none of the converse implications is true, hence we get *CI*, *CII*.

We continue with some examples of Moore automata having **C**, *CI*, and *CII*.

First, the automaton in Figure 2 has **C** as experiment 10 distinguishes between any pair of distinct states.

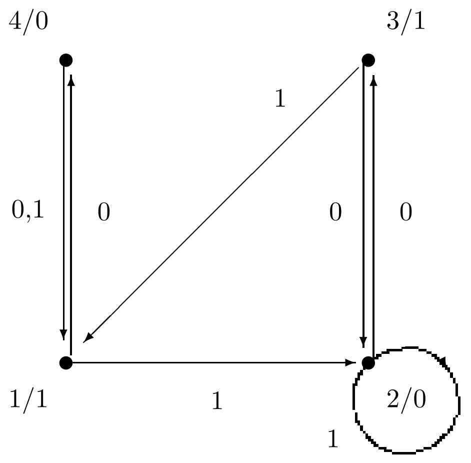

Figure 2.

Moore's automaton in Figure 1 has **A** but non-**B**, hence *CI* (cf. [13]). Every pair of distinct states can be distinguished by an experiment: states $1,2$ by $x = 0$, states $1,3$ by $x = 1$, states $1,4$ by $x = 0$, states $2,3$ by $x = 0$, states $2,4$ by $x = 0$, and states $3,4$ by $x = 0$. However, there is no (unique) experiment capable to distinguish between every pair of arbitrary distinct states. If the experiment starts with 1, i.e. $x = 1u$, then $E(1,x) = E(2,x)$, that is x cannot distinguish between the states $1,2$ as $E(1,x) = E(1,1u) = f(1)f(\delta(1,1))E(\delta(1,1),u) = f(1)f(3)E(3,u) = 00E(3,u)$ and $E(2,x) = E(2,1u) = f(2)f(\delta(2,1))E(\delta(2,1),u) = f(2)f(3)E(3,u) = 00E(3,u)$. If the experiment starts with 0, i.e. $x = 0v$, $v \in \Sigma^*$, then x cannot distinguish between the states $1,3$ as $E(1,x) = E(1,0v) = f(1)f(\delta(1,0))E(\delta(1,0),v) = f(1)f(4)E(4,v) = 01E(4,v)$ and $E(3,x) = E(3,0v) = f(3)f(\delta(3,0))E(\delta(3,0),v) = f(3)f(4)E(4,v) = 01E(4,v)$.

The automaton in Figure 3 has **B** but not **C**, hence *CII*. Indeed, the following pairs of states are distinguishable by every experiment: $(1,2)$, $(1,4)$, $(2,3)$, $(3,4)$. Accordingly, 1 is distinguishable from the other states by $w = 0$, 2 is distinguishable by $w = 1$, 3 is distinguishable by $w = 0$ and 4 is distinguishable by $w = 1$, so the automaton has property **B**. It does not have property **C** because any experiment w which starts with 1, i.e. $w = 1x$, $x \in \Sigma^*$, does not distinguish between 1 and 3, and any experiment w which starts with 0, i.e. $w = 0y$, $y \in \Sigma^*$, does not distinguish between 2 and 4.

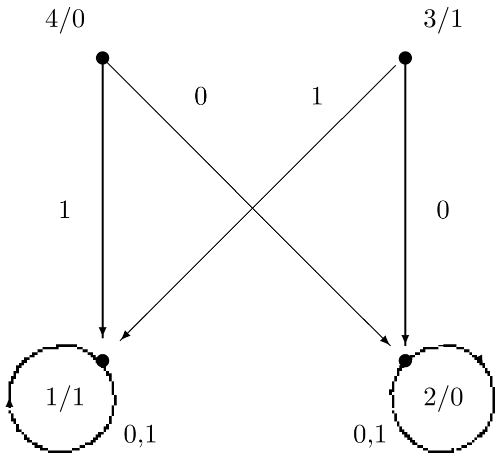

Figure 3.

3 An Algorithm for Testing Simultaneously *CI* and *CII*

In this section we briefly review a few facts on partitions and present the algorithm that is used to test properties **A**, **B** and **C**, which uses partitions defined on sets of states of an automaton. An elegant algebraic theory for machine decomposition based on the closed partition lattice of a machine is presented in [10] and efficient algorithms for constructing the lattice are presented in [12]; here we construct a different partition lattice testing the properties *CI* and *CII*, a different problem.

A partition P of a set Q is a set of non-empty disjoint sets whose union is Q. Partitions are in an one-to-one onto correspondence with equivalence relations. In particular, we will use the partition induced by the level sets of a map $f : Q \to Q$, that is, the sets $[q]_f = \{x : f(x) = f(q)\}, q \in Q$.

Given two partitions P_1 and P_2 of Q, we say that P_1 is no coarser than (or at least as coarse as) P_2, written $P_1 \leq P_2$ if for every $p_1 \in P_1$, there exists $p_2 \in P_2$ such that $p_1 \subseteq p_2$. We say that P_2 is coarser than P_1 if $P_1 \leq P_2$ and $P_1 \neq P_2$, in symbols $P_1 < P_2$. The term *finer* means the inverse relation of coarser. When $P_1 \leq P_2$ we say that P_1 is a refinement of P_2, or that P_2 is a coarsening of P_1.

Treating the above refinement relation as a partial order $\leq$, we see that the greatest lower bound $P_1 \wedge P_2$ is the coarsest partition of Q that is a refinement of both P_1 and P_2. This operation, which we will call *CCR* (the coarsest common refinment), can be conducted in principle by taking the intersection of all classes in P_1 with all classes in P_2, and then throwing out the empty sets.

Let P_1 and P_2 be two partitions of Q, and $\equiv_1$ and $\equiv_2$ be the corresponding equivalences. Then the equivalence relation $p \equiv q$ defined by $p \equiv_1 q$ and $p \equiv_2 q$, corresponds to $P_1 \wedge P_2$.

The level sets of the composition $f \circ g$ are coarser than those of g; if g is invertible then the level sets are the same. Let $f \times g : a \to (f(a), g(a))$. Then, the level set partition of $f \times g$ is the coarsest common refinement of f and g.

For an automaton M, we construct a graph, called an observation graph, which describes how information about the state of machine changes with observations. Each vertex R is a record $\begin{bmatrix} t \\ P \end{bmatrix}$, where the two fields t and P are the configuration of states under the transition function and the partition induced by the output function respectively. The partition induced by t, $\Pi(t)$, is given by the following equivalence relation: i is equivalent to j modulo $\Pi(t)$ if $f(t[i]) = f(t[j])$.

An edge $\left(\begin{bmatrix} t_1 \\ P_1 \end{bmatrix}, \begin{bmatrix} t_2 \\ P_2 \end{bmatrix}\right)$ belongs to the graph exactly when there exists $s \in \{0, 1\}$ such that $t_2 = \delta(t_1, s)$ and $P_2 = \Pi(f \circ t_2) \wedge P_1$. Since the CCR of two partitions is no coarser than either it is apparent that along any path through the graph the partitions may become finer. The function $\left(\begin{bmatrix} t \\ P \end{bmatrix}, s\right) \to P$, mapping vertices into lattice of partitions, is monotonic.

If at the start of a path the condition $P_1 \leq t_1$ occurs, then, $P_2 = \Pi(\delta(t_1, s)) \wedge$

$P_1 \le \Pi(\delta(t_1, s)) = \Pi(t_2)$.

For any path $\left(\begin{bmatrix} t_1 \\ P_1 \end{bmatrix} \cdots \begin{bmatrix} t_n \\ P_n \end{bmatrix}\right)$ in the observation graph, $P_n \le P_1$. For any path $\left(\begin{bmatrix} t_1 \\ P_1 \end{bmatrix} \cdots \begin{bmatrix} t_n \\ P_n \end{bmatrix}\right)$ in the observation graph if $P_1 \le \Pi(t_1)$, then $P_n = P_1$. Finally, suppose that $\left(\begin{bmatrix} t \\ P \end{bmatrix}, B\right)$ is a rooted sub-tree of the observation graph, and $P \le \Pi(t)$. The partition P is constant throughout the entire tree since each node on the rooted sub-tree is the end point of a path starting at the root $\begin{bmatrix} t \\ P \end{bmatrix}$. Consequently, the node $\begin{bmatrix} t \\ P \end{bmatrix}$ will be pruned (ignored).

The algorithm for testing properties **A**, **B**, and **C** for an automaton M generates records that are nodes of an observation graph and checks whether the partitions components of the nodes verify the conditions associated with the properties **A,** **B**, and **C**. The algorithm has the following steps.

Step 1. Initialization of

- a vector *Counter* recording all non-repeating nodes generated so far
- a trimmed binary tree *OG* recording the nodes in the observational graph
- a vector *TB* recording those states for which the condition in property **B** is currently verified, and a Boolean variable *hasB* that is true if M has property **B** and false otherwise
- a table *TA* recording those states for which the condition in property **A** is currently verified, and a Boolean variable *hasA* that is true if M has property **A** and false otherwise.

Step 2. Generate and test the first record

- *Step 2.1.* Generate the first record. The first record, say R, will be the root of OG. Its two components are given by
  - the vector of states $(1, 2, \ldots, n)$ and
  - the partition P_R, generated by the output function f,

  so $R = \begin{bmatrix} (1, 2, \ldots, n) \\ P_R \end{bmatrix}$
- *Step 2.2.* If M has **C** stop. Else:
- *Step 2.3.* Add the record to *Counter*
- *Step 2.4.* Update *TA* and *hasA*

- *Step 2.5.* Update *TB* and *hasB*
- *Step 2.6.* Add the record to *OG*

[Comment: We generate (from left to right) all children of non-pruned nodes. If no child can be generated, we check the values of *hasB* and *hasA* to determine whether the automaton has **B** or **A**, then stop.]

Step 3. While there are children to be generated do

- *Step 3.1.* Generate the next record obtained from the left/right child, say $N = \begin{bmatrix} t_N \\ P_N \end{bmatrix}$. Its two components are given by
  - t_N the vector of states $(i_1, i_2, \ldots, i_n)$, obtained by applying the transition function on each element of sequence of states in the parent node with input letter 0 for the left child and 1 for the right child and
  - the partition P_N obtained by taking the CCR of the parent's partition component and the partion of states induced by the output function on t_N
- *Step 3.2.* If N is in *Counter*, then go to *Step 3.1.* Otherwise, add the current record to *Counter*
- *Step 3.3.* If M has **C**, stop. Else:
- *Step 3.4.* Update *TA* and *hasA*
- *Step 3.5.* Update *TB* and *hasB*
- *Step 3.6.* If the record can be pruned, then go to *Step 3.1*
- *Step 3.6.* Add the node N to *OG* and go to *Step 3.1*

End of while loop.

If TA is false, then return non-**A**, stop; else, if TB is false, then return *CI*, stop; else, retun *CII*, stop.

End of algorithm.

4 The Algorithm in Action

In this section we present some examples illustrating the algorithm presented in the previous section.

Example 1. Let us run the algorithm on the automaton in Figure 1.

Step 1. Initialize OG to $\emptyset$, $Counter$ to the $\emptyset$ vector, TA and TB, respectively, the $n \times n$ matrix with all elements 0 and the n-element all 0 vector, and the Boolean variables $hasA$ and $hasB$ to false

Step 2 Generate the first record

Step 2.1 Generate the record $R = \begin{bmatrix} 1,\, 2,\, 3,\, 4 \\ \{1,2,3\},\{4\} \end{bmatrix}$

Step 2.2 The record R does not satisfy $\mathbf{C}$ as $P_R \neq \{1\},\{2\},\{3\},\{4\}$

Step 2.3 Update $Counter$ to $[R]$

Step 2.4 Update TA to $\begin{bmatrix} 0,\, 0,\, 0,\, 1 \\ 0,\, 0,\, 0,\, 1 \\ 0,\, 0,\, 0,\, 1 \\ 1,\, 1,\, 1,\, 0 \end{bmatrix}$ and $hasA$ to false

Step 2.5 Update TB to $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ and $hasB$ to false

Step 2.6 Update OG to $(R, \emptyset)$

Step 3 Generate the next child

First iteration:

Step 3.1 The next record, $N0$, is

$$\begin{bmatrix} 4,\, 1,\, 4,\, 2 \\ \{1,3\},\{2\},\{4\} \end{bmatrix},$$

where $t_{N0} = (4,1,4,2)$ is obtained by applying the transition function on $1,2,3,4$ to the input letter 0. The partition induced by the output function on $4,1,4,2$ is $\{1,3\},\{2,4\}$. Taking the CCR between this partition and the partition component of its parent, i.e. $P_R = \{1,2,3\},\{4\}$, we obtain the partition component of $N0$. Therefore

$$P_{N0} = \{1,3\},\{2,4\} \wedge \{1,2,3\},\{4\} = \{1,3\},\{2\},\{4\}$$

Step 3.2 As the current node is not in *Counter* we add it:

$$Counter = [R, N0]$$

Step 3.3 The current automaton has not **C** as

$$P_{N0} \neq \{1\}, \{2\}, \{3\}, \{4\}$$

Step 3.4 Update *TA* to $\begin{bmatrix} 0, 1, 0, 1 \\ 1, 0, 1, 1 \\ 0, 1, 0, 1 \\ 1, 1, 1, 0 \end{bmatrix}$ and *hasA* to false

Step 3.5 Update *TB* to $\begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$ and *hasB* to false

Step 3.6 As

$$\{1,3\}, \{2\}, \{4\} \wedge \{1,3\}, \{2\}, \{4\} = \{1,3\}, \{2\}, \{4\}$$

it follows that

$$\{1,3\}, \{2,4\} \wedge P_{N0} = P_{N0}$$

and therefore this record has to be pruned (as none of its children can bring any new information)

Step 3 Generate next child

Second iteration:

Step 3.1 The next record, $N1$ is

$$\begin{bmatrix} 3, 3, 4, 2 \\ \{1,2\}, \{3\}, \{4\} \end{bmatrix},$$

where $t_{N1} = (3, 3, 4, 2)$ is obtained by applying the transition function on $1, 2, 3, 4$ and the input letter 1. The partition induced by the output function on $3, 3, 4, 2$ is $\{1, 2, 4\}, \{3\}$. Taking the CCR between this partition and the partition component of the node's parent, i.e. $P_R = \{1, 2, 3\}, \{4\}$, we obtain the partition component of $N1$. Therefore

$$P_{N1} = \{1, 2, 4\}, \{3\} \wedge \{1, 2, 3\}, \{4\} = \{1, 2\}, \{3\}, \{4\}$$

Step 3.2 As the current node is not in *Counter* we add it:

$$Counter = [R, N0, N1]$$

Step 3.3 The current automaton has not **C** as

$$P_{N1} \neq \{1\}, \{2\}, \{3\}, \{4\}$$

Step 3.4 Update *TA* to $\begin{bmatrix} 0, 1, 1, 1 \\ 1, 0, 1, 1 \\ 1, 1, 0, 1 \\ 1, 1, 1, 0 \end{bmatrix}$ and *hasA* to true

Step 3.5 Update *TB* to $\begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$ and *hasB* to false

Step 3.6 As $\{1,2\},\{3\},\{4\} \wedge P_N = \{1,2\},\{3\},\{4\} \wedge \{1,2\},\{3\},\{4\} = \{1,2\},\{3\},\{4\}$ this record has to be pruned

Step 3 Generate the next child

Third iteration:

As $OG = (R, \emptyset)$ and $Counter = (R, N0, N1)$, there is no child to generate, stop. As *hasA* is true and *hasB* is false, the output is "the automaton has *CI*".

Example 2. Let us run the algorithm on the automaton in Figure 2.

Step 1. Initialize *OG* to $\emptyset$, *Counter* to the $\emptyset$ vector, *TA* and *TB*, respectively, the $n \times n$ matrix with all elements 0 and the n-element all 0 vector, and the Boolean variables *hasA* and *hasB* to false

Step 2 Generate the first record

Step 2.1 Generate the record $R = \begin{bmatrix} 1, 2, 3, 4 \\ \{1,3\}, \{2,4\} \end{bmatrix}$

Step 2.2 As $P_R \neq \{1\}, \{2\}, \{3\}, \{4\}$ the automaton has not **C**

Step 2.3 Update $Counter = [R]$

Step 2.4 Update *TA* to $\begin{bmatrix} 0, 1, 0, 1 \\ 1, 0, 1, 0 \\ 0, 1, 0, 1 \\ 1, 0, 1, 0 \end{bmatrix}$ and *hasA* to false

Step 2.5 Update *TB* to $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ and *hasB* to false

Step 2.6 Update *OG* to $(R, \emptyset)$

Step 3 Generate the next child

First iteration:

Step 3.1 The next record, $N0$, is

$$\begin{bmatrix} 4,3,2,1 \\ \{1,3\},\{2,\,4\} \end{bmatrix}$$

where $t_{N0} = (4,3,2,1)$ is obtained by applying the transition function on $1,2,3,4$ to the input letter 0. The partition induced by the output function on $4,3,2,1$ is $\{1,3\},\{2,\,4\}$. Taking the CCR between this partition and the partition component of its parent $PR = \{1,3\},\{2,4\}$, we obtain the partition component of $N0$:

$$P_{N0} = \{1,3\},\{2,\,4\} \wedge P_R = \{1,3\},\{2,\,4\}$$

Step 3.2 As the current node is not in *Counter* we add it:

$$Counter = [R, N0]$$

Step 3.3 The current automaton has not **C** as

$$P_{N0} = \{1\},\{2\},\{3\},\{4\}$$

Step 3.4 Update *TA* to $\begin{bmatrix} 0,\,1,\,0,\,1 \\ 1,\,0,\,1,\,0 \\ 0,\,1,\,0,\,1 \\ 1,\,0,\,1,\,0 \end{bmatrix}$ and *hasA* to false

Step 3.5 Update *TB* to $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ and *hasB* to false

Step 3.6 As

$$\{1\},\{2\},\{3\},\{4\} \wedge P_{N0} \neq P_{N0}$$

this record should not be pruned. Update $OG = (R, N0, [R, N0])$

Step 3 Generate the next child

Second iteration:

Step 3.1 The next record, $N1$, is

$$\begin{bmatrix} 2, 2, 1, 1 \\ \{1\}, \{2\}, \{3\}, \{4\} \end{bmatrix}$$

where $t_{N1} = (2, 2, 1, 1)$ is obtained by applying the transition function on states $1, 2, 3, 4$ and the input letter 1. The partition induced by the output function on $2, 2, 1, 1$ is $\{1, 2\}, \{3, 4\}$. Taking the CCR between this partition and the partition component of the node's parent, i.e. $P_R = \{1, 3\}, \{2, 4\}$, we obtain the partition component of $N1$:

$$P_{N1} = \{1, 3\}, \{2, 4\} \wedge \{1, 2\}, \{3, 4\} = \{1\}, \{2\}, \{3\}, \{4\}$$

Step 3.2 The current node is not in *Counter*, so we add it:

$$Counter = [R, N0, N1]$$

Step 3.3 The current automaton has **C** as

$$P_{N1} = \{1\}, \{2\}, \{3\}, \{4\}$$

so the output is "the automaton has **C**".

Example 3. Let us run the algorithm on the automaton in Figure 3.

Step 1. Initialize OG to $\emptyset$, *Counter* to the $\emptyset$ vector, TA and TB, respectively, the $n \times n$ matrix with all elements 0 and the n-element all 0 vector, and the Boolean variables *hasA* and *hasB* to false

Step 2 Generate the first record

Step 2.1 Generate the record $R = \begin{bmatrix} 1, 2, 3, 4 \\ \{1, 3\}, \{2, 4\} \end{bmatrix}$

Step 2.2 The record R has not **C** as $P_R \neq \{1\}, \{2\}, \{3\}, \{4\}$

Step 2.3 Update $Counter = [R]$

Step 2.4 Update TA to $\begin{bmatrix} 0, 1, 0, 1 \\ 1, 0, 1, 0 \\ 0, 1, 0, 1 \\ 1, 0, 1, 0 \end{bmatrix}$ and *hasA* to false

Step 2.5 Update TB to $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ and $hasB$ to false

Step 2.6 Update OG to $(R, \emptyset)$

Step 3 Generate the next child

First iteration:

Step 3.1 The next record, $N0$, is

$$\begin{bmatrix} 1,2,2,2 \\ \{1\},\{2,4\},\{3\} \end{bmatrix},$$

where $t_{N0} = (1,2,2,2)$ is obtained by applying the transition function on states $1,2,3,4$ to the input letter 0. The partition induced by the output function on $1,2,2,2$ is $\{1\},\{2,3,4\}$. Taking the CCR between this partition and the partition component of its parent $P_R = \{1,3\},\{2,4\}$ we obtain the partition component of $N0$:

$$P_{N0} = \{1\},\{2,3,4\} \wedge P_R = \{1\},\{2,4\},\{3\}$$

Step 3.2 The current node is not in *Counter*, so we add it:

$$Counter = [R, N0]$$

Step 3.3 The current automaton has not $\mathbf{C}$ as

$$P_{N0} \neq \{1\},\{2\},\{3\},\{4\}$$

Step 3.4 Update TA to $\begin{bmatrix} 0,\ 1,\ 1,\ 1 \\ 1,\ 0,\ 1,\ 0 \\ 1,\ 1,\ 0,\ 1 \\ 1,\ 0,\ 1,\ 0 \end{bmatrix}$ and hasA to false

Step 3.5 Update TB to $\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ and $hasB$ to false

Step 3.6 As

$$\{1\},\{2,3,4\} \wedge P_{N0} = P_{N0}$$

this record has to be pruned

Step 3 Generate the next child

Second iteration:

Step 3.1 The next record, $N1$, is

$$\begin{bmatrix} 1,\ 2,\ 1,\ 1 \\ \{1,3\},\{2\},\{4\} \end{bmatrix},$$

where $t_{N1} = (1,2,1,1)$ is obtained from applying the transition function on states $1,2,3,4$ and the input letter 1. The partition induced by the output function on $2,1,1,1$ is $\{1,3,4\},\{2\}$. Taking the CCR between this partition and the partition component of the node's parent $P_R = \{1,3\},\{2,4\}$ we obtain the partition component of $N1$:

$$P_{N1} = \{1,3,4\},\{2\} \wedge \{1,3\},\{2,4\} = \{1,3\},\{2\},\{4\}.$$

Step 3.2 As the current node is not in *Counter* we add it:

$$Counter = [R, N0, N1]$$

Step 3.3 The current automaton has not **C** as

$$P_{N1} \neq \{1\},\{2\},\{3\},\{4\}$$

Step 3.4 Update *TA* to $\begin{bmatrix} 0,\ 1,\ 1,\ 1 \\ 1,\ 0,\ 1,\ 1 \\ 1,\ 1,\ 0,\ 1 \\ 1,\ 1,\ 1,\ 0 \end{bmatrix}$ and *hasA* to true

Step 3.5 Update *TB* to $\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$ and *hasB* to true

Step 3.6 As $\{1,3,4\},\{2\} \wedge P_N = P_N$ this record has to be pruned

Step 3 Generate the next child

Third iteration:

As $OG = (R, \emptyset)$ and $Counter = [R, N0, N1]$, there is no child to generate, stop. As *hasA* is true and *hasB* is true, the automaton has property **B**, but not **C**, so the output is "the automaton has *CII*".

5 Experimental Results

The proposed algorithm was implemented in C and the program[3] was run on a Pentium III, i686 processor using Redhat 8.0 Linux, 250 Mb of RAM. The aim was to study the distributions of *CI* and *CII* over the set of all possible automata with a given number of states and input/output symbols. Table 2 presents the results of the main tests that have been done so far.

$n \times s$	# automata	time (sec)	*CI*	*CII*	*CII*/*CI* %	*CI* %	*CII* %
2×2	32	< 1	0	0	0	0	0
3×2	2916	< 2	0	0	0	0	0
4×2	524288	< 11	73728	30720	41.67	14.06	5.86
5×2	156250000	< 8435	46862400	19436160	41.47	29.99	12.44
4×3	452984832	< 14018	54577152	46227456	84.70	12.05	10.12

Table 2.

In the first column $n \times s$ stands for the class of automata with n states and s input letters. Because of symmetries (the automaton (Q, δ, f) "is equivalent" to the automaton $(Q, \delta, 1 - f)$), the program actually tests only half of automata of type $n \times s$; the numbers of tested automata are shown in the second column. The third column contains the time for processing all automata mentioned in the second column. The numbers of automata verifying *CI* and *CII* are given in the next two columns. The last three columns present the percentage of *CII* over *CI* and the percentage of *CI*, respectively, *CII*, over the total number of automata processed.

We also tested automata with more than five states. For example, the 10-state automaton in Table 3

q	$\delta(q,0)$	$\delta(q,1)$	$f(q)$
1	2	1	0
2	3	2	0
3	4	3	0
4	5	4	0
5	6	5	0
6	7	6	0
7	8	7	0
8	9	8	0
9	9	10	0
10	10	10	1

Table 3.

[3]See `http://www.massey.ac.nz/~bimills/obgraph.c` for the program.

has **A** (00000001 distinguishes the pairs (i,j) for $i=1,2,3, j=1,2,\ldots,10$ and $i \neq j$, 00001 distinguishes the pairs $(4,5),(4,6),(4,7),(4,8),(4,9)$, 0001 distinguishes the pairs $(5,6),(5,7),(5,8),(5,9)$, 001 distinguishes the pairs $(6,7),(6,8),(6,9)$, 01 distinguishes the pairs $(7,8),(7,9)$, 1 distinguishes the pair $(8,9)$ and ε distinguishes the pairs $(i,10)$ for $i=1,2,\ldots,9$), has **B** (as state 1 is distinguished from all other states by the word 00000001, state 2 by 000000101, state 3 by 00000101, state 4 by 0000101, state 5 by 000101, state 6 by 00101, state 7 by 0101, state 8 by 101, state 9 by 1, and state 10 by ε) and has **C** (the word 101010101010101 distinguishes every pair of distinct states). The algorithm has scanned 766 nodes in less than a second.

6 Final Remarks

Based on the concept of observation graph of an automaton, new equivalent definitions have been given for two types of computational complementarity studied in [3]. As a result, we proposed an algorithm for simultaneously determining these properties. The algorithm has been shown in practice to be fast enough (on a standard desktop machine) for testing all binary Moore machines up to five states. Some other experiments reported in the paper illustrate the power of the algorithm.

Many problems remain open; for example, what is the complexity of the decision problems *CI*, *CII*.

Acknowledgement

We are grateful to the anonymous referee for suggestions which improved the paper.

References

[1] Brauer, W. *Automaten theorie.* Teubner, Stuttgart, 1984.

[2] Calude, C. S., Calude, E. and Ştefănescu, C. Computational complementarity for Mealy automata, *EATCS Bull.* 66 (1998), 139–149.

[3] Calude, C., Calude, E., Svozil, K. and Yu, S. Physical versus computational complementarity I. *International Journal of Theoretical Physics 36* (1997), 1495–1523.

[4] Calude, C. S. and Lipponen, M. Computational complementarity and sofic shifts, in X. Lin (ed.). *Theory of Computing 98, Proceedings of the 4th Australasian Theory Symposium, CATS'98*, Springer-Verlag, Singapore, 1998, 277–290.

[5] Calude, E. and Lipponen, M. Minimal deterministic incomplete automata. *Journal of Universal Computer Science 11* (1997), 1180–1193.

[6] Calude, E. and Lipponen, M. Deterministic incomplete automata: Simulation, universality and complementarity, in C. S. Calude, J. Casti, and M. J. Dinneen (eds.). *Unconventional Models of Computation*, Springer-Verlag, Singapore, 1998, 131–149.

[7] Chaitin, G. J. An improvement on a theorem by E. F. Moore. *IEEE Transactions on Electronic Computers EC-14* (1965), 466–467.

[8] Conway, J. H. *Regular Algebra and Finite Machines.* Chapman and Hall Ltd., London, 1971.

[9] Ginsburg, S. On the length of the smallest uniform experiment which distinguishes the terminal states of the machine. *Journal of the Association for Computing Machinery 5* (1958), 266–280.

[10] Hartmanis J. and Stearns R. E. *Algebraic Structure Theory of Sequential Machines*, Prentice-Hall, Englewood Cliffs, NJ, 1966.

[11] Lee, D. and Yannakakis, M. Principles and methods of testing finite state machines—A survey, *Proc. IEEE* 84, 8, (1996),1089–1123.

[12] Lee, D. and Yannakakis, M. Closed partition lattice and machine decomposition, *IEEE Transactions on Computers*, 51, 2 (2002), 216–228.

[13] Moore, E. F. Gedanken-experiments on sequential machines, in C. E. Shannon and J. McCarthy (eds.). *Automata Studies*, Princeton University Press, Princeton, 1956.

[14] Schaller, M. and Svozil, K. Automaton partition logic versus quantum logic. *International Journal of Theoretical Physics 34*, 8 (1995), 1741–1750.

[15] Svozil, K. *Randomness &Undecidability in Physics.* World Scientific, Singapore, 1993.

Received September, 2003