Simulation of Gate Circuits in the Algebra of Transients

Janusz Brzozowski and Mihaela Gheorghiu

School of Computer Science, University of Waterloo,
Waterloo, ON, Canada N2L 3G1
{brzozo,mgheorgh}@uwaterloo.ca

Abstract. We study simulation of gate circuits in algebra C recently introduced by Brzozowski and Ésik. A transient is a word consisting of alternating 0s and 1s; it represents a changing signal. In C, gates process transients instead of 0s and 1s. Simulation in C is capable of counting signal changes, and detecting hazards. We study two simulation algorithms: a general one, A, that works with any state, and Ã, that applies if the initial state is stable. We show that the two algorithms agree in the stable case. We prove the sufficiency of the simulation: all signal changes occurring in binary analysis are also predicted by Algorithm A.

1 Introduction

Asynchronous circuits, in contrast to synchronous ones, operate without a clock. Interest in asynchronous circuits has grown in recent years [4, 6, 9], because they offer the potential for higher speed and lower energy consumption, avoid clock distribution problems, handle metastability safely, and are amenable to modular design.

Despite its advantages, asynchronous design has some problems, among them, hazards. A hazard is an unwanted signal change, caused by stray delays. A hazard may affect the correctness of a computation. Because hazards are important, much research has been done on their detection. Multiple-valued algebras play an important role here [2]. Recently, Brzozowski and Ésik introduced an infinite-valued algebra C, which subsumes all the previously used algebras [1], and a polynomial-time simulation algorithm based on C. The algorithm is capable not only of detecting hazards, but also of counting the number of signal changes in the worst case; this provides an estimate of the energy consumption.

The purpose of this paper is to compare the Brzozowski-Ésik simulation of a circuit with the binary analysis of the circuit. We prove that all the changes that occur in the binary analysis, are also predicted by simulation.

2 The Network Model

The material here is based on [3]. For an integer $n > 0$, $[n]$ denotes $\{1, \ldots, n\}$. Boolean operations AND, OR, and NOT are denoted $\wedge$, $\vee$, and $\overline{}$, respectively. Given a gate circuit with n inputs and m gates, we associate an *input variable* X_i with each input, $i \in [n]$, and a *state variable* s_j with the output of each gate, $j \in [m]$. Input and state variables take values in the binary domain $\mathcal{D} = \{0, 1\}$. Each state variable s_i has an *excitation* S_i, which is the Boolean function of the corresponding gate.

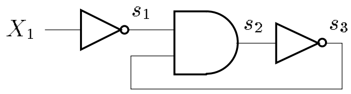

Fig. 1. Sample gate circuit

Definition 1. *A* network *is a tuple* $N = \langle \mathcal{D}, \mathcal{X}, \mathcal{S}, \mathcal{E} \rangle$*, where* $\mathcal{D}$ *is the domain of values,* $\mathcal{X} = \{X_1, \ldots, X_n\}$*, the set of inputs,* $\mathcal{S} = \{s_1, \ldots, s_m\}$*, the set of state variables with associated excitations* $S_1, \ldots, S_m$*, and* $\mathcal{E} \subseteq (\mathcal{X} \times \mathcal{S}) \cup (\mathcal{S} \times \mathcal{S})$*, a set of directed edges. There is an edge between* x *and* y *if and only if the excitation of* y *depends on* x*. The* network graph *is the digraph* $(\mathcal{X} \cup \mathcal{S}, \mathcal{E})$*. Note that* $\mathcal{D}$ *need not be* $\{0, 1\}$.

Example 1. The circuit of Fig. 1 has input X_1, state variables s_1, s_2, s_3, and excitations $S_1 = \overline{X_1}, \quad S_2 = s_1 \wedge s_3, \quad S_3 = \overline{s_2}$ in domain $\mathcal{D} = \{0, 1\}$. Its network graph is shown in Fig. 2.

Fig. 2. Network graph for circuit of Fig. 1

A *state* of N is an m-tuple b of values from $\mathcal{D}$ assigned to state variables $s_1, \ldots, s_m$. A *total state* is an $(n + m)$-tuple $c = a \cdot b$ of values from $\mathcal{D}$, the n-tuple a being the values of the inputs, and the m-tuple b, the values of state variables. The dot " $\cdot$ " separates inputs from state variables.

Each excitation S_i is a function of some inputs $X_{j_1}, \ldots, X_{j_l} \in \mathcal{X}$, and some state variables $s_{i_1}, \ldots, s_{i_k} \in \mathcal{S}$, *i.e.*, $S_i = f(X_{j_1}, \ldots, X_{j_l}, s_{i_1}, \ldots, s_{i_k})$, where $f : \mathcal{D}^{l+k} \to \mathcal{D}$. We also treat S_i as a function from $\mathcal{D}^{n+m}$ into $\mathcal{D}$. Thus, let $\tilde{S}_i : \mathcal{D}^{n+m} \to \mathcal{D}$ be $\tilde{S}_i(a \cdot b) = f(a_{j_1}, \ldots, a_{j_l}, b_{i_1}, \ldots, b_{i_k})$, for any $a \cdot b$. From now on we write S_i for $\tilde{S}_i$; the meaning is clear from the context.

For any $i \in [m]$, the value of S_i in total state $a \cdot b$ is denoted $S_i(a \cdot b)$. The tuple $S_1(a \cdot b), \ldots, S_m(a \cdot b)$ is denoted by $S(a \cdot b)$. For any $a \cdot b$, we define the set of unstable state variables as $U(a \cdot b) = \{s_i \mid b_i \neq S_i(a \cdot b)\}$. Thus, $a \cdot b$ is *stable* if and only if $U(a \cdot b) = \emptyset$, *i.e.*, $S(a \cdot b) = b$.

3 Binary Analysis of Networks

In response to changes of its inputs, a circuit passes through a sequence of states as its internal signals change. By analyzing a circuit we mean exploring all possible sequences of states. This section describes a formal analysis model introduced by Muller [10], and later called the *General Multiple Winner* (GMW) model. Our presentation follows that of [3], but here we refer to the GMW model as *binary analysis*.

In this section we use the binary domain, $\mathcal{D} = \{0, 1\}$. We describe the behavior of a network started in a given state with the input kept constant at value $a \in \{0, 1\}^n$, by defining a binary relation R_a on the set $\{0, 1\}^m$ of states of N. For any $b \in \{0, 1\}^m$,

bR_ab, if $U(a \cdot b) = \emptyset$, *i.e.*, total state $a \cdot b$ is stable, and bR_ab^K, if $U(a \cdot b) \neq \emptyset$, and K is any nonempty subset of $U(a \cdot b)$, where by b^K we mean b with all the variables in K complemented. No other pairs of states are related by R_a. As usual, we associate a digraph with the R_a relation, and denote it G_a. In examples, we represent tuples without commas or parentheses, for convenience. Thus $(0, 0, 0)$ is written as 000, *etc.*

For given $a \in \{0,1\}^n$, and $b \in \{0,1\}^m$ we define the set of all states reachable from b in relation R_a as *reach*$(R_a(b)) = \{c \mid bR_a^*c\}$, where R_a^* is the reflexive and transitive closure of R_a. We denote by $G_a(b)$ the subgraph of G_a corresponding to *reach*$(R_a(b))$.

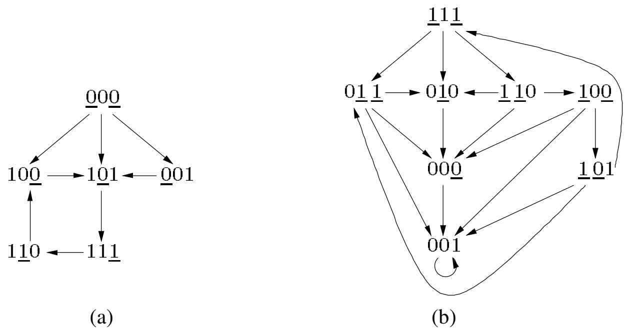

Fig. 3. Sample $G_a(b)$ graphs for circuit of Fig. 1

Example 2. For the circuit in Fig. 1, graph $G_0(000)$ is shown in Fig. 3(a), where unstable variables are underlined. Note that the graph contains no stable states. Graph $G_1(111)$ is shown in Fig. 3(b). Here there is one stable state. To illustrate hazardous behavior, consider path $\pi_1 = 111, 011, 001$. Here s_2 changes once from 1 to 0, and s_3 does not change. However, along path $\pi_2 = 111, 110, 100, 101, 011, 001$, s_2 changes from 1 to 0 to 1 to 0, and s_3 changes from 1 to 0 to 1. If the behavior of π_1 is the intended one, then π_2 violates it. Along π_2 there are unwanted signal pulses: a 1-pulse in s_2, and a 0-pulse in s_3. The first pulse is an example of a *dynamic hazard*, and the second, of a *static hazard*. These pulses can introduce errors in the circuit operation.

4 Transients

While binary analysis is an exhaustive analysis of a circuit, it is inefficient, since the state space is exponential. Simulation using a multi-valued domain is an efficient alternative, if not all the information from binary analysis is needed.

The material here is based on [1]. A *transient* is a nonempty word over $\{0, 1\}$ in which no two consecutive symbols are the same. Thus the set of all transients is $\mathbf{T} = 0(10)^* \cup 1(01)^* \cup 0(10)^*1 \cup 1(01)^*0$. Transients represent waveforms in a natural way, as shown in Fig. 4. We use boldface symbols to denote transients, tuples of transients, and functions of transients. For any transient $\mathbf{t}$ we denote by $\alpha(\mathbf{t})$ and $\omega(\mathbf{t})$ its first and last characters, respectively. A transient can be obtained from any nonempty binary word by *contraction*, *i.e.*, the elimination of all duplicates immediately following a symbol (*e.g.*,

Fig. 4. Transients as words for waveforms

the contraction of 00100011 is 0101). For a binary word s we denote by $\hat{s}$ the result of its contraction. For any $\mathbf{t}, \mathbf{t}' \in \mathbf{T}$, we denote by $\mathbf{t}\mathbf{t}'$ the concatenation of $\mathbf{t}$ and $\mathbf{t}'$.

The prefix order on $\mathbf{T}$ is denoted $\leq$, and is extended to tuples. For $\mathbf{u} = (\mathbf{u}_1, \ldots, \mathbf{u}_m)$ and $\mathbf{v} = (\mathbf{v}_1, \ldots, \mathbf{v}_m)$ in $\mathbf{T}^m$, we say that $\mathbf{u}$ is a prefix of $\mathbf{v}$ and write $\mathbf{u} \leq \mathbf{v}$, if $\mathbf{u}_i \leq \mathbf{v}_i$, for all $i \in [m]$.

Extensions of Boolean functions to functions of transients are defined in [1]. Any Boolean function $f : B^n \to B$ is extended to a function $\mathbf{f} : \mathbf{T}^n \to \mathbf{T}$ so that, for any tuple $(\mathbf{t}_1, \ldots, \mathbf{t}_n)$ of transients, $\mathbf{f}$ produces the longest transient when $\mathbf{t}_1, \ldots, \mathbf{t}_n$ are applied to the inputs of a gate performing the Boolean function f. We give an example of extended Boolean function next. For more details see [1].

Example 3. Let f to be the two-input OR function and $\mathbf{f}$, its extension. Suppose we want to compute $\mathbf{f}(01, 010)$. We construct a digraph $D(01, 010)$ in which the nodes consist of all the pairs $(\mathbf{t}, \mathbf{t}')$ of transients such that $(\mathbf{t}, \mathbf{t}') \leq (01, 010)$, and there is an edge between any two pairs $\mathbf{p}, \mathbf{p}'$ only if $\mathbf{p} \leq \mathbf{p}'$, and $\mathbf{p}$ differs from $\mathbf{p}'$ in exactly one coordinate by exactly one letter. The resulting graph is shown in Fig. 5(a). Also, for each node $(\mathbf{t}, \mathbf{t}')$ in the graph we consider as its label the value $f(\omega(\mathbf{t}), \omega(\mathbf{t}'))$. This results in a graph of labels, shown in Fig. 5(b). The value of $\mathbf{f}(01, 010)$ is the contraction of the label sequence of those paths in the graph of labels that have the largest number of alternations between 0 and 1. Therefore, $\mathbf{f}(01, 010) = 0101$.

Let $z(\mathbf{t})$ and $u(\mathbf{t})$ denote the number of 0s and the number of 1s in a transient $\mathbf{t}$, respectively. We denote by $\otimes$ and $\oplus$ the extensions of the Boolean AND and OR operations, respectively. It is shown in [1] that for any $\mathbf{w}, \mathbf{w}' \in \mathbf{T}$ of length > 1, $\mathbf{w} \otimes \mathbf{w}' = \mathbf{t}$, where $\mathbf{t} \in \mathbf{T}$ is such that $\alpha(\mathbf{t}) = \alpha(\mathbf{w}) \wedge \alpha(\mathbf{w}')$, $\omega(\mathbf{t}) = \omega(\mathbf{w}) \wedge \omega(\mathbf{w}')$, and $u(\mathbf{t}) = u(\mathbf{w}) + u(\mathbf{w}') - 1$. Similarly, $\mathbf{w} \oplus \mathbf{w}' = \mathbf{t}$, where $\mathbf{t} \in \mathbf{T}$ is such that $\alpha(\mathbf{t}) = \alpha(\mathbf{w}) \vee \alpha(\mathbf{w}')$, $\omega(\mathbf{t}) = \omega(\mathbf{w}) \vee \omega(\mathbf{w}')$, and $z(\mathbf{t}) = z(\mathbf{w}) + z(\mathbf{w}') - 1$. If one of the arguments is 0 or 1 the following rules apply:

$$\mathbf{t} \oplus 0 = 0 \oplus \mathbf{t} = \mathbf{t}, \quad \mathbf{t} \oplus 1 = 1 \oplus \mathbf{t} = 1,$$
$$\mathbf{t} \otimes 1 = 1 \otimes \mathbf{t} = \mathbf{t}, \quad \mathbf{t} \otimes 0 = 0 \otimes \mathbf{t} = 0.$$

The complement $\overline{\mathbf{t}}$ of $\mathbf{t} \in \mathbf{T}$ is obtained by complementing each character of $\mathbf{t}$. For example, $\overline{1010} = 0101$.

The algebra $C = (\mathbf{T}, \oplus, \otimes, ^{-}, 0, 1)$, is called the *change-counting algebra*, and is a commutative de Morgan bisemigroup [1]. We also refer to C as the *algebra of transients*.

We denote by $\mathbf{t} \circ \mathbf{t}'$ concatenation followed by contraction, *i.e.*, $\mathbf{t} \circ \mathbf{t}' = \widehat{\mathbf{t}\mathbf{t}'}$. The $\circ$ operation is associative, and also satisfies for $\mathbf{t}, \mathbf{t}', \mathbf{t}_1, \ldots, \mathbf{t}_n \in \mathbf{T}$ and $b \in \{0, 1\}$: 1. if $\mathbf{t} \leq \mathbf{t}'$ then $b \circ \mathbf{t} \leq b \circ \mathbf{t}'$; and 2. $\mathbf{t}_1 \circ \ldots \circ \mathbf{t}_n = \widehat{\mathbf{t}_1 \ldots \mathbf{t}_n}$.

$$\begin{tikzcd}
(0,0) \arrow[r] \arrow[d] & (01,0) \arrow[d] \\
(0,01) \arrow[r] \arrow[d] & (01,01) \arrow[d] \\
(0,010) \arrow[r] & (01,010)
\end{tikzcd}
\qquad
\begin{tikzcd}
0 \arrow[r] \arrow[d] & 1 \arrow[d] \\
1 \arrow[r] \arrow[d] & 1 \arrow[d] \\
0 \arrow[r] & 1
\end{tikzcd}$$

(a) (b)

Fig. 5. Graph $D(01, 010)$ with labels

5 Simulation with Algebra C

A simulation algorithm using algebra C has been proposed in [1]; it generalizes ternary simulation [3, 5]. We now give a more general version of the simulation algorithm, and show how it relates to the original version. This parallels the extension of ternary simulation from stable initial state to any initial state [3].

Given any circuit, we use two networks: a binary network $N = \langle\{0, 1\}, \mathcal{X}, \mathcal{S}, \mathcal{E}\rangle$ and the *transient network* $\mathbf{N} = \langle\mathbf{T}, \mathcal{X}, \mathcal{S}, \mathcal{E}\rangle$ having set $\mathbf{T}$ of transients as the domain. The two networks have the same input and state variables, but these variables take values from different domains. A state of network $\mathbf{N}$ is a tuple of transients; the value of the excitation of a variable is also a transient. Excitations in $\mathbf{N}$ are the extensions to C of the Boolean excitations in N. It is shown in [1] that an extended Boolean function depends on one of its arguments if and only if the corresponding Boolean function depends on that argument. Therefore N and $\mathbf{N}$ have the same set of edges.

Binary variables, words, tuples and excitations in N are denoted by italic characters (*e.g.*, s, S). Transients, tuples of transients, and excitations in $\mathbf{N}$ are denoted by boldface characters (*e.g.*, $\mathbf{s}$, $\mathbf{S}$). We refer to components of a tuple by subscripts (*e.g.*, $\mathbf{s}_i$, s_i).

5.1 General Simulation: Algorithm A

We want to record in the value of a variable all the changes in that variable since the start of the simulation, as dictated by its excitation. For variables that are stable initially, since the initial state agrees with the initial excitation, the state transient and the excitation transient will be the same, so at each step we just copy the excitation into the variable. For example, with initial state 0 and excitation 0, if the excitation becomes 01, we set the variable to 01, and so on. For variables that are initially unstable, we first record the initial state, and then the excitation. The operator that gives us the desired result in both cases is $\circ$; thus we have *new_value* $=$ *initial_value* $\circ$ *excitation.*

Let $a \cdot b$ be a (binary) total state of $\mathbf{N}$. Algorithm A is defined as follows:

Algorithm A
$\mathbf{s}^0 := b$;
$h := 1$;
$\mathbf{s}^h := b \circ \mathbf{S}(a \cdot \mathbf{s}^0)$;
<u>while</u> $(\mathbf{s}^h <> \mathbf{s}^{h-1})$ **<u>do</u>**
    $h := h + 1$;
    $\mathbf{s}^h := b \circ \mathbf{S}(a \cdot \mathbf{s}^{h-1})$;

where $\circ$ is applied to tuples component-wise, i.e., for all m-tuples $\mathbf{u}, \mathbf{v}$ of transients, $\mathbf{u} \circ \mathbf{v} = \mathbf{w}$, where $\mathbf{w}$ is such that $\mathbf{w}_i = \mathbf{u}_i \circ \mathbf{v}_i$, for all $i \in [m]$.

Algorithm A produces a sequence $\mathbf{s}^0, \mathbf{s}^1, \ldots, \mathbf{s}^h, \ldots$, where $\mathbf{s}^h = (\mathbf{s}_1^h, \mathbf{s}_2^h, \ldots, \mathbf{s}_m^h) \in \mathbf{T}^m$, for all $h \geq 0$. This sequence can be finite, if we reach $\mathbf{s}^{h_0} = \mathbf{s}^{h_0 - 1}$ for some $h_0 > 0$, or infinite otherwise. For convenience, we sometimes consider the finite sequences as being infinite, with $\mathbf{s}^h = \mathbf{s}^{h_0}$, for all $h > h_0$.

It is shown in [1] that any extended Boolean function $\mathbf{f} : \mathbf{T}^m \to \mathbf{T}$ is *monotonic* with respect to the prefix order, *i.e.*, for any $\mathbf{x}, \mathbf{y} \in \mathbf{T}^m$, if $\mathbf{x} \leq \mathbf{y}$, then $\mathbf{f}(\mathbf{x}) \leq \mathbf{f}(\mathbf{y})$.

Proposition 1. *The sequence resulting from Algorithm A is nondecreasing or monotonic with respect to the prefix order, that is, for all $h \geq 0$, $\mathbf{s}^h \leq \mathbf{s}^{h+1}$.*

Proof: Since extended Boolean functions are monotonic with respect to the prefix order, so are excitations. We proceed by induction on h.
Basis, $h = 0$: $\mathbf{s}^0 = b \leq b \circ \mathbf{S}(a \cdot \mathbf{s}^0) = \mathbf{s}^1$.
Induction step: $\mathbf{s}^h = b \circ \mathbf{S}(a \cdot \mathbf{s}^{h-1}) \leq b \circ \mathbf{S}(a \cdot \mathbf{s}^h) = \mathbf{s}^{h+1}$. ■

For feedback-free circuits, the sequence resulting from Algorithm A is finite. We can see this if we order the state variables by levels as follows. Level 1 consists of all state variables which depend only on external inputs. Level l consists of all state variables which depend only on variables of level $< l$, and on at least one variable of level $l - 1$. Since the inputs do not change during simulation, level-1 variables change at most once, in the first step of Algorithm A. In general, level-i variables change at most i times. Since the number of levels is finite, our claim follows. Thus the running time of A for feedback-free circuits is polynomial in the number of state variables.

For display reasons, in examples of simulation we write binary states as words, but during computations they are regarded as tuples.

Example 4. Consider the feedback-free circuit in Fig. 6. The excitations are: $\mathbf{S}_1 = \overline{\mathbf{X}_2}$, $\quad \mathbf{S}_2 = \mathbf{X}_1 \otimes \mathbf{s}_1$, $\quad \mathbf{S}_3 = \overline{\mathbf{s}_2}$, $\quad \mathbf{S}_4 = \mathbf{s}_2 \oplus \mathbf{s}_3$. For the initial state $a \cdot b = 11 \cdot 1011$, Algorithm A results in Table 1 (left).

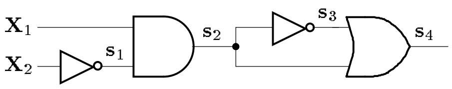

Fig. 6. Circuit with finite simulation

Example 5. For circuits with feedback the simulation sequence may be infinite. Consider the circuit with feedback in Fig. 1. The excitation functions are: $\mathbf{S}_1 = \overline{\mathbf{X}_1}$, $\quad \mathbf{S}_2 = \mathbf{s}_1 \otimes \mathbf{s}_3$, $\quad \mathbf{S}_3 = \overline{\mathbf{s}_2}$. We run Algorithm A for this network started in state $a \cdot b = 0 \cdot 000$; the resulting sequence of states, which is infinite, is illustrated in Table 2.

Table 1. Results of Algorithms A and Ã

$\mathbf{X}_1$	$\mathbf{X}_2$	$\mathbf{s}_1$	$\mathbf{s}_2$	$\mathbf{s}_3$	$\mathbf{s}_4$	$state$
1	1	1	0	1	1	$\mathbf{s}^0$
1	1	10	01	1	1	$\mathbf{s}^1$
1	1	10	010	10	1	$\mathbf{s}^2$
1	1	10	010	101	1010	$\mathbf{s}^3$
1	1	10	010	101	10101	$\mathbf{s}^4$

$\mathbf{X}_1$	$\mathbf{X}_2$	$\tilde{\mathbf{s}}_1$	$\tilde{\mathbf{s}}_2$	$\tilde{\mathbf{s}}_3$	$\tilde{\mathbf{s}}_4$	$state$
1	10	0	0	1	1	$\tilde{\mathbf{s}}^0$
1	10	01	0	1	1	$\tilde{\mathbf{s}}^1$
1	10	01	01	1	1	$\tilde{\mathbf{s}}^2$
1	10	01	01	10	1	$\tilde{\mathbf{s}}^3$
1	10	01	01	10	101	$\tilde{\mathbf{s}}^4$

Table 2. Infinite simulation

$\mathbf{X}_1$	$\mathbf{s}_1$	$\mathbf{s}_2$	$\mathbf{s}_3$	state
0	0	0	0	$\mathbf{s}^0$
0	01	0	01	$\mathbf{s}^1$
0	01	01	01	$\mathbf{s}^2$
0	01	01	010	$\mathbf{s}^3$
...	...	...	...	...

5.2 Simulation with Stable Initial State: Algorithm Ã

Algorithm A above makes no assumptions about the starting state $a \cdot b$. If the network starts in a stable total state and the inputs change, then we have a slightly simpler formulation which we call Algorithm Ã; this is the version used in [1]. Assume $\mathbf{N}$ is started in stable total state $\tilde{a} \cdot b$ and the input tuple changes to a.

Algorithm Ã
$\mathbf{a} = \tilde{a} \circ a$;
$\tilde{\mathbf{s}}^0 := b$;
$h := 1$;
$\tilde{\mathbf{s}}^h := \mathbf{S}(\mathbf{a} \cdot \tilde{\mathbf{s}}^0)$;
while $(\tilde{\mathbf{s}}^h <> \tilde{\mathbf{s}}^{h-1})$ **do**
  $h := h + 1$;
  $\tilde{\mathbf{s}}^h := \mathbf{S}(\mathbf{a} \cdot \tilde{\mathbf{s}}^{h-1})$;

Example 6. We illustrate Algorithm Ã with the network in Fig. 6, started in stable state $\tilde{a} \cdot b = 11 \cdot 0011$, with the input changing to $a = 10$. The result is shown in Table 1 (right).

It is shown in [1] that the sequence of states resulting from Algorithm Ã is nondecreasing with respect to the prefix order, *i.e.*, Algorithm Ã is monotonic.

For our next result, we modify the circuit model slightly. For each input X_i we add a delay, called *input gate*, with output s_i and excitation $S_i = X_i$. This follows the model of [3]. The following shows that Algorithms A and Ã are equivalent for any network $\mathbf{N}$ started in a stable state, provided that $\mathbf{N}$ contains input-gate variables.

Theorem 1. *Let* $\mathbf{N}$ *be a network containing input-gate variables. Let* $\tilde{\mathbf{s}}^0, \tilde{\mathbf{s}}^1, \ldots, \tilde{\mathbf{s}}^h, \ldots$ *be the sequence of states produced by Algorithm Ã for* N *started in the stable (binary)* *total state $\tilde{a} \cdot b$ with the input tuple changing to a. Then, for all $h \geq 0$, $\tilde{\mathbf{s}}^h = \mathbf{s}^h$, where $\mathbf{s}^0, \mathbf{s}^1, \ldots, \mathbf{s}^h, \ldots$ is the sequence of states produced by Algorithm A for N started in total state $a \cdot b$.*

Proof: We prove the theorem by induction on h.
Basis, $h = 0$. Since $\mathbf{s}^0 = b = \tilde{\mathbf{s}}^0$, the basis holds.
First step, $h = 1$. In states $\tilde{\mathbf{s}}^0$ and $\mathbf{s}^0$ only input-gate variables can be unstable; therefore only they can change in the first step of $\tilde{\text{A}}$, and of A. One easily verifies that $\tilde{\mathbf{s}}^1 = \mathbf{s}^1$.
Induction step. For any $i \in [m]$, if $\mathbf{s}_i$ is an input-gate variable then $\mathbf{s}_i^h = \mathbf{s}_i^{h-1}$ and $\tilde{\mathbf{s}}_i^h = \tilde{\mathbf{s}}_i^{h-1}$, because in both algorithms the input-gate variables do not change after the first step. By the induction hypothesis, we have $\mathbf{s}_i^h = \tilde{\mathbf{s}}_i^h$. If $\mathbf{s}_i$ is not an input-gate variable, then it is initially stable in both algorithms, and its excitation does not depend on the input tuple, *i.e.*, $\mathbf{S}_i(a \cdot \mathbf{x}) = \mathbf{S}_i(\mathbf{a} \cdot \mathbf{x})$, for any (internal) state tuple $\mathbf{x}$. Then $\mathbf{s}_i^h = \mathbf{S}_i(a \cdot \mathbf{s}^{h-1}) = \mathbf{S}_i(\mathbf{a} \cdot \tilde{\mathbf{s}}^{h-1}) = \tilde{\mathbf{s}}_i^h$. Hence $\tilde{\mathbf{s}}_i^h = \mathbf{s}_i^h$, for all $i \in [m]$. ■

6 Covering of Binary Analysis by Simulation

Given the two networks N and $\mathbf{N}$ modeling a gate circuit, we perform the binary analysis for N and Algorithm A for $\mathbf{N}$, both with the same starting total state $a \cdot b$. The binary analysis results in graph $G_a(b)$. Let the state sequence resulting from Algorithm A be $\mathbf{s}^0, \mathbf{s}^1, \ldots, \mathbf{s}^h, \ldots$, where $\mathbf{s}^h = (\mathbf{s}_1^h, \mathbf{s}_2^h, \ldots, \mathbf{s}_m^h) \in \mathbf{T}^m$, for all $h \geq 0$.

We now show that binary analysis is covered by Algorithm A. Take any path from the initial state b in graph $G_a(b)$. Suppose the length of the path is h. For each state variable s_i we consider the transient that shows the changes of that variable along the path. We show that this transient is a prefix of the value $\mathbf{s}_i^h$ that variable $\mathbf{s}_i$ takes in the h-th iteration of Algorithm A.

Example 7. Consider the binary counterpart of the transient network in Fig. 6 with $S_1 = \overline{X_2}$, $S_2 = X_1 \wedge s_1$, $S_3 = \overline{s_2}$, $S_4 = s_2 \vee s_3$. In $G_{11}(1011)$, with the same initial total state as in Example 4, we find a path $\pi = 1011, 1111, 0111, 0001$ of length $h = 3$. If we follow state variable s_3, for example, it changes from 1 to 0 along this path, so the corresponding transient is 10. The value of $\mathbf{s}_3$ in the third step of Algorithm A is $\mathbf{s}_3^3 = 101$, which has 10 as a prefix. In fact, this holds for all variables, since $(10, 010, 10, 1) \leq (10, 010, 101, 1010)$.

Definition 2. *Let $\pi = s^0, \ldots, s^h$ be a path of length $h \geq 0$ in $G_a(b)$. Recall that each s^j is a tuple $(s_1^j, \ldots, s_m^j)$. For any $i \in [m]$, we denote by $\boldsymbol{\sigma}_i^\pi$ the transient $\widehat{s_i^0 \ldots s_i^h}$, which shows the changes of the i-th state variable along path π. We refer to it as the* history *of variable s_i along the path. We define $\boldsymbol{\Sigma}_i^\pi$ to be $\widehat{E_i}$, where $E_i = S_i(a \cdot s^0) S_i(a \cdot s^1) \ldots S_i(a \cdot s^h)$, and we call it the* excitation history *of variable s_i along path π. The histories of all variables along π constitute tuple $\boldsymbol{\sigma}^\pi = (\boldsymbol{\sigma}_1^\pi, \ldots, \boldsymbol{\sigma}_m^\pi)$. The histories of all excitations along π form tuple $\boldsymbol{\Sigma}^\pi = (\boldsymbol{\Sigma}_1^\pi, \ldots, \boldsymbol{\Sigma}_m^\pi)$.*

Note that $\boldsymbol{\sigma}_i^\pi$ and $\boldsymbol{\Sigma}_i^\pi$ are not always the same. If s_i is unstable initially, they are obviously different, since their first characters are different, that is $s_i^0 \neq S_i(a \cdot s_i^0)$. Even if the variable is stable initially, $\boldsymbol{\sigma}_i^\pi$ and $\boldsymbol{\Sigma}_i^\pi$ can still be different.

Example 8. An example of a path in graph $G_{11}(1011)$ of the previous example, on which a variable changes fewer times than its excitation is path $\pi = 1011, 0111, 0011$, where $\boldsymbol{\sigma}_3^\pi = 1$, whereas $\boldsymbol{\Sigma}_3^\pi = 101$.

Let $\mathbf{s}^h$ be the state produced by Algorithm A after h steps, and let $\pi = s^0, \ldots, s^h$ be a path of length $h \geq 0$ in $G_a(b)$, with $s^0 = b$. We prove that $\boldsymbol{\sigma}^\pi \leq \mathbf{s}^h$.

Proposition 2. *Let* $\pi = s^0, \ldots, s^h, s^{h+1}$ *be a path in* $G_a(b)$*, and let* $\pi' = s^0, \ldots, s^h$*. Then,* $\boldsymbol{\sigma}^\pi \leq \boldsymbol{\sigma}^{\pi'} \circ S(a \cdot s^h)$*.*

Proof: For any variable s_i we have one of the following cases.
Case I, s_i changes during the transition from s^h to s^{h+1}. Then s_i must be unstable in state s^h, *i.e.*, $S_i(a \cdot s^h) \neq s_i^h$, and $s_i^{h+1} = S_i(a \cdot s^h)$, by the definition of binary analysis. Hence $\boldsymbol{\sigma}_i^\pi = s_i^0 \widehat{\ldots s_i^h s_i^{h+1}} = \widehat{s_i^0 \ldots s_i^h} \circ s_i^{h+1} = \boldsymbol{\sigma}_i^{\pi'} \circ S_i(a \cdot s^h)$.
Case II, s_i does not change during the transition from s^h to s^{h+1}. Then $s_i^{h+1} = s_i^h$, by the definition of binary analysis. Then $\boldsymbol{\sigma}_i^\pi = s_i^0 \widehat{\ldots s_i^h s_i^{h+1}} = \widehat{s_i^0 \ldots s_i^h} = \boldsymbol{\sigma}_i^{\pi'} \leq \boldsymbol{\sigma}_i^{\pi'} \circ S_i(a \cdot s^h)$. Thus, our claim holds. ■

Corollary 1. *For any path* $\pi = s^0, \ldots, s^h, s^{h+1}$ *in* $G_a(b)$*, with* $\pi' = s^0, \ldots, s^h$ *we have* $\boldsymbol{\sigma}^\pi \leq s^0 \circ \boldsymbol{\Sigma}^{\pi'}$*.*

Proof: $\boldsymbol{\sigma}^\pi \leq \boldsymbol{\sigma}^{\pi'} \circ S(a \cdot s^h) \leq (\ldots((s^0 \circ S(a \cdot s^0)) \circ S(a \cdot s^1)) \circ \ldots) \circ S(a \cdot s^h) = s^0 \circ (S(a \cdot s^0) \circ S(a \cdot s^1) \circ \ldots \circ S(a \cdot s^h)) = s^0 \circ \boldsymbol{\Sigma}^{\pi'}$. ■

Proposition 3. *For any path* $\pi = s^0, \ldots, s^h$ *in* $G_a(b)$*,* $\boldsymbol{\Sigma}^\pi \leq \mathbf{S}(a \cdot \boldsymbol{\sigma}^\pi)$*.*

Proof: Let $\pi_j = s^0, \ldots, s^j$, for all j such that $0 \leq j \leq h$. Then $\boldsymbol{\sigma}^{\pi_0} \leq \boldsymbol{\sigma}^{\pi_1} \leq \ldots \leq \boldsymbol{\sigma}^\pi$. Thus $a \cdot \boldsymbol{\sigma}^{\pi_0} \leq a \cdot \boldsymbol{\sigma}^{\pi_1} \leq \ldots \leq a \cdot \boldsymbol{\sigma}^\pi$, which means that $a \cdot \boldsymbol{\sigma}^{\pi_0}, a \cdot \boldsymbol{\sigma}^{\pi_1}, \ldots, a \cdot \boldsymbol{\sigma}^\pi$ is a subsequence q of nodes on a path p from $a \cdot \alpha(\boldsymbol{\sigma}_1^\pi) \ldots \alpha(\boldsymbol{\sigma}_m^\pi) = a \cdot s^0 = a \cdot \boldsymbol{\sigma}^{\pi_0}$ to $a \cdot \boldsymbol{\sigma}^\pi$ in the graph $D(a \cdot \boldsymbol{\sigma}^\pi)$. For any $i \in [m]$, we consider the labeling of graph $D(a \cdot \boldsymbol{\sigma}^\pi)$ with Boolean excitation S_i. Let λ be the sequence of labels of p. The sequence of labels on q is $E_i = S_i(a \cdot s^0), S_i(a \cdot s^1), \ldots, S_i(a \cdot s^h)$. Since q is a subsequence of p, $\widehat{E_i} \leq \hat{\lambda}$. By the definition of extended Boolean functions, $\mathbf{S}_i(a \cdot \boldsymbol{\sigma}^\pi)$ is the longest transient obtained by the contraction of the label sequences of paths from $a \cdot \boldsymbol{\sigma}^{\pi_0}$ to $a \cdot \boldsymbol{\sigma}^\pi$ in graph $D(a \cdot \boldsymbol{\sigma}^\pi)$. Hence $\hat{\lambda} \leq \mathbf{S}_i(a \cdot \boldsymbol{\sigma}^\pi)$. By the definition of the excitation history, $\boldsymbol{\Sigma}_i^\pi = \widehat{E_i}$. It follows that $\boldsymbol{\Sigma}_i^\pi \leq \mathbf{S}_i(a \cdot \boldsymbol{\sigma}^\pi)$. ■

Theorem 2. *For all paths* $\pi = s^0, \ldots, s^h$ *in* $G_a(b)$*, with* $s^0 = b$*,* $\boldsymbol{\sigma}^\pi \leq \mathbf{s}^h$*, where* $\mathbf{s}^h$ *is the* $(h+1)$*st state in the sequence resulting from Algorithm A.*

Proof: We prove the theorem by induction on $h \geq 0$.
Basis, $h = 0$. We have $\pi = s^0 = b = \mathbf{s}^0$; hence $\boldsymbol{\sigma}^\pi = s^0 = \mathbf{s}^0$, so the claim holds.
Induction hypothesis. The claim holds for some $h \geq 0$, *i.e.*, for all paths π of length h from b in $G_a(b)$, we have $\boldsymbol{\sigma}^\pi \leq \mathbf{s}^h$.
Induction step. Let $\gamma = s^0, \ldots, s^h, s^{h+1}$ be a path of length $h+1$ from b in $G_a(b)$. Then $\pi = s^0, \ldots, s^h$ is a path of length h, and we have

$$\begin{aligned}
\boldsymbol{\sigma}^{\gamma} &\le s^0 \circ \boldsymbol{\Sigma}^{\pi} && \{\text{ Cor. 1 }\}\\
&\le b \circ \mathbf{S}(a \cdot \boldsymbol{\sigma}^{\pi}) && \{\ s^0 = b \text{ and Prop. 3 }\}\\
&\le b \circ \mathbf{S}(a \cdot \mathbf{s}^{h}) && \{\text{ induction hypothesis, monotonicity of excitations,}\\
& && \quad \text{and property of } \circ \ \}\\
&= \mathbf{s}^{h+1} && \{\text{ definition of Algorithm A }\}.
\end{aligned}$$

■

Corollary 2. *If Algorithm A terminates with state* $\mathbf{s}^H$*, then for any path* π *from* b *in* $G_a(b)$*,* $\boldsymbol{\sigma}^{\pi} \le \mathbf{s}^H$.

Proof: Suppose there exists a path π from b in $G_a(b)$ that satisfies $\boldsymbol{\sigma}_i^{\pi} > \mathbf{s}_i^H$, for some $i \in [m]$. Let h be the length of π. If $h \le H$, Theorem 2 shows that $\boldsymbol{\sigma}^{\pi} \le \mathbf{s}^h$. We also have $\mathbf{s}^h \le \mathbf{s}^H$, by Prop. 1. So $\boldsymbol{\sigma}^{\pi} \le \mathbf{s}^H$, and in particular $\boldsymbol{\sigma}_i^{\pi} \le \mathbf{s}_i^H$, which contradicts our supposition. If $h > H$, then Theorem 2 states that $\boldsymbol{\sigma}^{\pi} \le \mathbf{s}^h$. By our convention, $\mathbf{s}^h = \mathbf{s}^H$. So, again we have $\boldsymbol{\sigma}_i^{\pi} \le \mathbf{s}_i^H$, which is a contradiction. ■

7 Conclusions

We have proved that all the changes that occur in binary analysis are also detected by simulation. In a companion paper [8] we prove a partial converse of this result.

Acknowledgments: This research was supported by the Natural Sciences and Engineering Research Council of Canada under grant No. OGP0000871.

References

1. Brzozowski, J. A., Ésik, Z.: Hazard algebras. Maveric Res. Report 00–2, University of Waterloo, ON, Canada, http://maveric.uwaterloo.ca (2000; revised 2001)
2. Brzozowski, J. A., Ésik, Z., Iland, Y.: Algebras for hazard detection. Proc. 31st Int. Symp. Multiple-Valued Logic, IEEE Comp. Soc. (2001) 3–12
3. Brzozowski, J. A., Seger, C.-J. H.: Asynchronous circuits. Springer-Verlag (1995)
4. Coates, W. S., Lexau, J. K., Jones, I. W., Fairbanks, S. M., Sutherland, I. E.: A FIFO data switch design experiment. Proc. ASYNC '98, IEEE Comp. Soc. (1998) 4–16
5. Eichelberger, E. B.: Hazard detection in combinational and sequential circuits. IBM J. Res. and Dev. **9** (1965) 90–99
6. Garside, J. D.: AMULET3 revealed. Proc. ASYNC '99, IEEE Comp. Soc. (1999) 51–59
7. Gheorghiu, M.: Circuit simulation using a hazard algebra. MMath Thesis, Department of Computer Science, University of Waterloo, Waterloo, ON, Canada (2001)
8. Gheorghiu, M., Brzozowski, J. A.: Feedback-free circuits in the algebra of transients. Maveric Res. Report 02–3, University of Waterloo, ON, Canada, http://maveric.uwaterloo.ca (2002)
9. Kessels, J., Marston, P.: Designing asynchronous standby circuits for a low-power pager. Proc. ASYNC '97, IEEE Comp. Soc. (1997) 268–278
10. Muller, D. E., Bartky, W. C.: A theory of asynchronous circuits. Proc. Int. Symp. on Theory of Switching, Annals of Comp. Lab., Harvard University **29** (1959) 204–243