Simulation of Gate Circuits with Feedback in Multi-Valued Algebras

Janusz Brzozowski
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada N2L 3G1
email: brzozo@uwaterloo.ca

Yuli Ye
Department of Computer Science
University of Toronto
Toronto, ON, Canada M5S 3G4
email: y3ye@cs.toronto.edu

Abstract

Simulation of gate circuits is an efficient method of detecting hazards and oscillations that may occur because of delays. Ternary simulation consists of two algorithms, A and B, and is well understood. It has been generalized to an infinite algebra $\mathcal{C}$ and finite algebras $\mathcal{C}_k$, $k \geq 2$, where $\mathcal{C}_2$ is ternary algebra. Simulation in $\mathcal{C}$ has been studied extensively for feedback-free circuits, for which Algorithm A always terminates and Algorithm B is unnecessary. We study the simulation of gate circuits with feedback in finite algebras $\mathcal{C}_k$. The gate functions are restricted to a set that includes all the 1- and 2-variable functions and multi-input AND, OR, NAND, NOR, XOR *and* XNOR *functions. We prove that Algorithm B in Algebra $\mathcal{C}_k$, for $k > 2$, provides no more information than in ternary algebra. Thus, for any gate in any circuit, the final result of Algorithm B is always one of the binary values, 0 or 1, or the "uncertain" value; the remaining values of $\mathcal{C}_k$ never appear. This permits us to replace Algorithm B in $\mathcal{C}_k$ by the same algorithm in ternary algebra, and to reduce the simulation time.*

1 Introduction

Gate circuits with feedback continue to be of interest; see, for example, [2, 3, 6, 7]. We study the following analysis problem. Suppose a circuit is started in a stable state and some of its inputs change once, and are kept at their final values. We call this a *transition* of the circuit. We wait an appropriately long time [3] and observe the gate outputs. Some gates may remain in their initial states, others may change one or more times, and still others may *oscillate*, that is, they may take on the values 0 and 1 infinitely often.

Hazards are undesirable pulses that may occur during a transition under some distributions of gate and wire delays. A *static hazard* exists in a gate if the gate output is the same before and after the transition, but takes on the complementary value temporarily during the transition. A *dynamic hazard* exists if a gate is supposed to change from its initial value to the complementary value only once, but changes several times instead. In general, hazards and oscillations are undesirable, because they may lead to errors.

It is possible to detect hazards and oscillations using an analysis based on Boolean algebra [3]. Because such methods are exponential in the number of gates and wires, several multi-valued algebras have been proposed to find an efficient simulation to detect these phenomena [2].

The first algebra with more than two values was ternary. Ternary simulation [4] uses the "uncertain" value Φ in addition to the binary values 0 and 1. It has two parts, called Algorithms A and B, which always terminate for any circuit; the resulting states of the gates after the algorithms are denoted $\mathbf{y}^A$ and $\mathbf{y}^B$. The circuit is started in a stable binary state and some inputs change. The changing inputs are set to Φ in Algorithm A. The circuit is then analyzed in ternary algebra to determine whether some gate outputs become Φ as well. If a gate retains its binary value in $\mathbf{y}^A$, then it does not change during the transition. If it becomes Φ, then it may change, depending on the relative delays in the circuit. In Algorithm B, the circuit starts in state $\mathbf{y}^A$, and the inputs that are Φ are set to their final binary values. The circuit is again analyzed in ternary algebra. If a gate is Φ in $\mathbf{y}^B$, then it may have a nontransient oscillation – one that can persist indefinitely. If a gate has the same value in the initial state and in $\mathbf{y}^B$, but is Φ in $\mathbf{y}^A$, then it has a static hazard. Ternary simulation is unable to detect dynamic hazards.

Ternary simulation was generalized [2] to an infinite algebra $\mathcal{C}$ and finite algebras $\mathcal{C}_k$, for $k = 2, 3, \ldots$. These algebras include all the successful multi-valued algebras used in the past, and $\mathcal{C}_2$ is ternary algebra. Algebra $\mathcal{C}$ uses the set of *transients* as the underlying set, where a transient is a nonempty word of alternating 0s and 1s.

Simulation in Algebra $\mathcal{C}$ was studied for feedback-free circuits [2, 5, 8]. Algorithm A in $\mathcal{C}$ always terminates for these circuits, detects all hazards, permits us to count the number of signal changes occurring under worst-case conditions, and has time complexity that is polynomial in the

number of gates. The results of the simulation are easily understood. For example, if a gate has the transient 1010 in $\mathbf{y}^A$, then the gate's initial value is 1, its final value is 0, and it may change three times during the transition.

If Algorithm A does not terminate for a circuit with feedback, Algorithm B is not applicable. However, it is possible to use an algebra $\mathcal{C}_k$, for any $k > 1$, in which Algorithm A always terminates. The underlying set of values for $\mathcal{C}_k$ is the set of all transients of length $< k$ together with Φ_k, which represents all the transients of length $\geq k$.

Since Algorithm A always terminates in $\mathcal{C}_k$, Algorithm B is again applicable. Here we characterize the results of Algorithm B in Algebra $\mathcal{C}_k$, for $k > 2$. We study functions from the set $\mathcal{G}$, where $\mathcal{G} = \mathcal{H} \cup \tilde{\mathcal{H}}$, $\mathcal{H} = \{\text{OR, XOR}\}$, and $\tilde{\mathcal{H}}$ is the set of functions obtained by complementing any number of inputs and/or the output of functions from $\mathcal{H}$. Since we allow OR and XOR gates with any number of inputs, including one, $\mathcal{G}$ contains all the 1- and 2-variable functions and multi-input AND, NAND, NOR and XNOR functions. We prove that the results obtained from Algorithm B in Algebra $\mathcal{C}_k$ contain the same information as those obtained using the ternary algebra $\mathcal{C}_2$.

2 Gate Networks

Gate networks are defined in terms of directed graphs [1] and Boolean functions. A *digraph (directed graph)* D is an ordered triple $D = (V, E, \psi)$, where V is a nonempty set of *vertices*, E is a set (disjoint from V) of *arcs*, and ψ is an *incidence function* assigning to each arc of D an ordered pair of (not necessarily distinct) vertices of D. If $e \in E$, $u, v \in V$, and $\psi(e) = (u, v)$, then e *joins* u to v, $t(e) = u$ is the *tail* of e, and $h(e) = v$ is the *head*.

For $r \geq 1$, we define $[r] = \{1, \ldots, r\}$. A *directed walk* in D is a finite, nonempty sequence $W = v_0, e_1, v_1, \ldots, e_r, v_r$, whose terms are alternately vertices and arcs, such that for each $i \in [r]$, e_i joins v_{i-1} to v_i; v_0 is the *origin* of W, v_r is its *terminus*, and $v_1, \ldots, v_{r-1}$ are *internal vertices*. The *length* of a walk W is r; note that v_0 is a walk of length 0. A *directed trail* is a directed walk in which all the arcs are distinct. A *directed path* is a directed trail in which all the vertices are distinct. A directed walk is *closed* if $r \geq 1$ and $v_r = v_0$. A *directed cycle* is a closed directed trail whose origin and internal vertices are distinct. Since we deal only with directed graphs, we say *walk* for *directed walk, etc.*

Suppose $D = (V, E, \psi)$ is a digraph. We assume that D has at least one vertex of indegree 0 and at least one vertex of indegree > 0. The vertices with indegree 0 are *(external) inputs*, and are labeled $x_1, \ldots, x_m$; let $V_x = \{x_1, \ldots, x_m\}$. The remaining vertices are labeled $y_1, \ldots, y_n$ and are *gates*. Let $V_y = \{y_1, \ldots, y_n\}$; then $V = V_x \cup V_y$. Thus $D = (V_x \cup V_y, E, \psi)$, where $V_x \cap V_y = \emptyset$.

Let $B = \{0, 1\}$ be the set of the two binary values. A *total state* is an assignment $s : V \to B$, and is denoted by an $(m+n)$-tuple $s = (s_1, \ldots, s_{m+n})$. The state s_i of vertex v_i in total state s is also denoted by $s(v_i)$. The *state of an arc* e from u to v in total state s is denoted by $s(e)$ and is defined as the state of vertex u, that is, $s(e) = s(t(e))$.

Arcs $e_{i_1}, \ldots, e_{i_{n_i}}$ such that $h(e_{i_1}) = \ldots = h(e_{i_{n_i}}) = y_i \in V_y$ are the n_i *inputs* of gate y_i, $n_i > 0$; these are all the arcs with head y_i. We assign a Boolean function $f_i : B^{n_i} \to B$ to each gate y_i; this is its *excitation function*.

A *(gate) network* is a 4-tuple $N = (V_x \cup V_y, E, \psi, f)$, where $(V_x \cup V_y, E, \psi)$ is a digraph as above, and $f = (f_1, \ldots, f_n)$ is the n-tuple of excitation functions. We denote OR by $\vee$, AND by $\wedge$, and complement by $^-$.

3 Extensions of Boolean Functions

A *transient* [2] is a nonempty binary word of alternating 0s and 1s, that is, it is an element of the set $\mathbf{T} = (01)^* \cup (10)^* \cup (01)^*0 \cup (10)^*1$. Transients are denoted by boldface letters. If $\mathbf{x}$ is a transient, then $\alpha(\mathbf{x})$, and $\omega(\mathbf{x})$ are the first and last letters of $\mathbf{x}$, respectively. Also $z(\mathbf{x})$ and $u(\mathbf{x})$ are the numbers of 0s and 1s in $\mathbf{x}$, respectively, and $l(\mathbf{x})$ is the length of $\mathbf{x}$.

We extend a Boolean function $f : B^n \to B$ to a Boolean function $\mathbf{f} : \mathbf{T}^n \to \mathbf{T}$ as follows. For an n-tuple $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ of transients, the digraph $F(\mathbf{x})$ of a Boolean function f has as vertices all the n-tuples $\mathbf{v} = (\mathbf{v}_1, \ldots, \mathbf{v}_n)$ of transients, where $\mathbf{v}_i$ is a nonempty prefix of $\mathbf{x}_i$, for all $i \in [n]$. There is an arc from vertex $\mathbf{v} = (\mathbf{v}_1, \ldots, \mathbf{v}_n)$ to vertex $\mathbf{v}' = (\mathbf{v}'_1, \ldots, \mathbf{v}'_n)$ if and only if $\mathbf{v}$ and $\mathbf{v}'$ differ in exactly one coordinate, say i, and $\mathbf{v}'_i = \mathbf{v}_i c$, where $c \in B$. Graph $F(\mathbf{x})$ shows all possible orders in which the n variables can change along paths from the initial vertex $(\alpha(\mathbf{x}_1), \ldots, \alpha(\mathbf{x}_n))$ to the final vertex $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$. Moreover, with each vertex $\mathbf{v}$ we associate an output $f(\omega(\mathbf{v}_1), \ldots, \omega(\mathbf{v}_n))$.

A *contraction* of a binary word w is a transient obtained by replacing all sequences of consecutive 0s by a single 0 and all sequences of consecutive 1s by a single 1. For example, the contraction of 011000111 is 0101. In the graph $F(\mathbf{x})$ for a Boolean function f, the *output of a path* π is the sequence of outputs of the vertices of π; it is denoted as $\mathbf{w}(\pi)$. The *transient of a path* π is the contraction $\mathbf{z}(\pi)$ of $\mathbf{w}(\pi)$. The value of the extension $\mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ of f is the longest possible path transient in $F(\mathbf{x})$. It represents the longest transient that might occur during the input change from $(\alpha(\mathbf{x}_1), \ldots, \alpha(\mathbf{x}_n))$ to $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

Proofs of the results stated below can be found in [J. A. Brzozowski and Y. Ye, *Gate Circuits with Feedback in Finite Algebras of Transients*, TR 2006–1, September 2006. http://maveric.uwaterloo.ca/publication.html.]

Proposition 1 *Suppose $n \geq 1$, and let $f, g : B^n \to B$ be any two Boolean functions. If, for all $x_1, \ldots, x_n \in B^n$, $f(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n) = g(x_1, \ldots, x_{i-1}, \overline{x_i}, x_{i+1}, \ldots, x_n)$, then, for all $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbf{T}$, $\mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_n) = \mathbf{g}(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, \overline{\mathbf{x}_i}, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_n)$.*

This result permits us to consider a gate realizing $\mathbf{f}$ as a composition of an inverter for $\mathbf{x}_i$ and a gate realizing $\mathbf{g}$.

Proposition 2 *Let $n \geq 1$ and let $f, g : B^n \to B$ be Boolean functions. If, for all $x_1, \ldots, x_n \in B^n$, $f(x_1, \ldots, x_n) = \overline{g}(x_1, \ldots, x_n)$, then for all $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbf{T}$, $\mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \overline{\mathbf{g}}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$.*

This proposition permits us to consider a gate realizing $\mathbf{f}$ as a composition of a gate realizing $\mathbf{g}$ and an inverter.

Definition 1 *Let $n > 1$ and let $f : B^n \to B$ be a Boolean function. A value $d \in B$ of argument x_i of f is* dominant *for f if the value of f is independent of the other arguments when $x_i = d$, that is, if $f(x_1, \ldots, x_{i-1}, d, x_{i+1}, \ldots, x_n) = f(y_1, \ldots, y_{i-1}, d, y_{i+1}, \ldots, y_n)$, for all $x_j, y_j \in B^{n-1}$, $j \in [n], j \neq i$. Let the value of f when $x_i = d$ be $f_{x_i=d}$.*

If $\mathbf{f}$ is the extension of f, then a transient value $\mathbf{t}$ of argument $\mathbf{x}_i$ is dominant *for $\mathbf{f}$ if the value of $\mathbf{f}$ is independent of the other arguments when $\mathbf{x}_i = \mathbf{t}$.*

For example, if $f(x_1, x_2) = \overline{x_1} \vee x_2$, then $x_1 = 0$ and $x_2 = 1$ are dominant.

Proposition 3 [2] *Let $n \geq 1$, let $f : B^n \to B$ be a Boolean function that depends on each of its arguments. If none of the $\mathbf{x}_i$ is a letter, then the length of $\mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is at least the maximum of the lengths of the $\mathbf{x}_i$.*

Proposition 4 *Let $n > 1$, let $f : B^n \to B$ be a Boolean function that depends on each of its arguments. A transient value $\mathbf{x}_i = \mathbf{t}$ is dominant for $\mathbf{f}$ if and only if it is binary, say $\mathbf{t} = d$, and $x_i = d$ is dominant for f. Also, if $\mathbf{x}_i = d$ is dominant for $\mathbf{f}$, then $\mathbf{f}_{\mathbf{x}_i=d} = f_{x_i=d}$.*

Remark 1 *Definition 1 and Proposition 4 can be extended to* tuples *of argument values. For example, consider $f(x_1, x_2, x_3) = (x_1 \wedge \overline{x_3}) \vee (x_2 \wedge x_3)$, which is not in $\mathcal{G}$. If $x_1 = 1$ and $x_2 = 1$; then $f(1, 1, x_3) = 1$, independently of x_3, but neither $x_1 = 1$ nor $x_2 = 1$ is dominant.*

In general, for $h < n$, we define an h-tuple t of argument values of a Boolean function of n variables to be dominant *if the value of the function depends only on the values in the h-tuple, and no subtuple of t has this property.*

Note that the OR *function with two or more arguments has only dominant 1-tuples, namely, every $x_i = 1$ is a dominant value. The* XOR *function with two or more arguments has no dominant tuples. It follows that every function in $\mathcal{G}$ with two or more arguments has either no dominant tuples or only dominant 1-tuples.* □

In what follows, we single out one of the inputs, say $\mathbf{x}_i$, of a function $f : \mathbf{T}^n \to \mathbf{T}$, $n > 1$, as the *main input*, and the remaining inputs are called *side inputs*. We study the relation between the length of $\mathbf{x}_i$ and the length of $\mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, with the side inputs as parameters.

An OR function of n variables is 1 if at least one of the variables is 1; thus a 1-argument OR function is the identity function. Recall [2] that, if $f : B^n \to B$ is the OR function, $n \geq 1$, then $\mathbf{y} = \mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is the word in $\mathbf{T}$ determined by the conditions

$$\alpha(\mathbf{y}) = \alpha(\mathbf{x}_1) \vee \ldots \vee \alpha(\mathbf{x}_n) \quad (1)$$

$$\omega(\mathbf{y}) = \omega(\mathbf{x}_1) \vee \ldots \vee \omega(\mathbf{x}_n) \quad (2)$$

$$z(\mathbf{y}) = \begin{cases} 0, & \text{if } \exists h \in [n]\ \mathbf{x}_h = 1 \\ 1 + \sum_{h=1}^{n} (z(\mathbf{x}_h) - 1), & \text{otherwise.} \end{cases} \quad (3)$$

Lemma 1 *If $f : B^n \to B$, $n > 1$, is the* OR *function, then*

1. *$l(\mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n)) < l(\mathbf{x}_i)$, if and only if $l(\mathbf{x}_i) > 1$ and one of the side inputs is 1.*
2. *$l(\mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n)) = l(\mathbf{x}_i)$, if and only if one of the following conditions holds:*
   - *(a) $\mathbf{x}_i = 1$.*
   - *(b) $\mathbf{x}_i = 0$ and one (or more) of the side inputs is 1.*
   - *(c) $\mathbf{x}_i = 0$ and all the side inputs are 0.*
   - *(d) $\mathbf{x}_i = (01)^i$, $i > 0$, and all the side inputs are either 01 or 0.*
   - *(e) $\mathbf{x}_i = (01)^i 0$, $i > 0$, and all the side inputs are 0.*
   - *(f) $\mathbf{x}_i = (10)^i$, $i > 0$, and all the side inputs are either 10 or 0.*
   - *(g) $\mathbf{x}_i = (10)^i 1$, $i > 0$, and all the side inputs are 101, 01, 10 or 0.*
3. *$l(\mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n)) > l(\mathbf{x}_i)$ if and only if the conditions in Parts 1 and 2 are not satisfied.*

A XOR function of n variables is 1 if an odd number of the variables is 1; thus a 1-argument XOR function is the identity function. Recall [2] that, if $n \geq 1$ and $f : B^n \to B$ is the XOR function, then $\mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is the word $\mathbf{y}$ in $\mathbf{T}$ satisfying the conditions

$$\alpha(\mathbf{y}) = f(\alpha(\mathbf{x}_1), \ldots, \alpha(\mathbf{x}_n)) \quad (4)$$

$$\omega(\mathbf{y}) = f(\omega(\mathbf{x}_1), \ldots, \omega(\mathbf{x}_n)) \quad (5)$$

$$l(\mathbf{y}) = 1 + \sum_{h=1}^{n} (l(\mathbf{x}_h) - 1). \quad (6)$$

Lemma 2 *If $f : B^n \to B$, $n > 1$, is the* XOR *function, then*

1. *$l(\mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n)) = l(\mathbf{x}_i)$, if and only if all the side inputs are binary.*
2. *$l(\mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n)) > l(\mathbf{x}_i)$ if and only if at least one side input is not binary.*

The following results are consequences of Lemmas 1 and 2, and Propositions 1 and 2.

Proposition 5 *If $n > 1$ and $f : B^n \to B$ is in $\mathcal{G}$, then $l(\mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n)) < l(\mathbf{x}_i)$ if and only if $l(\mathbf{x}_i) > 1$ and one of the side-input values of $\mathbf{f}$ is dominant.*

Corollary 1 *Let $n > 1$, and let $f : B^n \to B$ be in $\mathcal{G}$. If $\mathbf{f}$ has no dominant side-input values, and $l(\mathbf{x}_i) \geq k > 1$, then $l(\mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n)) \geq k$.*

4 Simulation in Algebra $\mathcal{C}_k$

The *change-counting algebra* [2] (also called *algebra of transients* [5]) is defined as $\mathcal{C} = (\mathbf{T}, \vee, \wedge, ^-, 0, 1)$, where $\vee$ is defined by Equations 1–3, the complement of a transient $\mathbf{t} = c_1 \ldots c_r$ is $\overline{\mathbf{t}} = \overline{c_1} \ldots \overline{c_r}$, and $\mathbf{t} \wedge \mathbf{s} = \overline{\overline{\mathbf{t}} \vee \overline{\mathbf{s}}}$.

For $k \geq 2$, relation $\sim_k$ is defined on $\mathbf{T}$: For $\mathbf{x}, \mathbf{y} \in \mathbf{T}$, $\mathbf{x} \sim_k \mathbf{y}$ if either $\mathbf{x} = \mathbf{y}$ or $\mathbf{x}$ and $\mathbf{y}$ are both of length $\geq k$. This relation is a congruence on $\mathcal{C}$, and the quotient algebra $\mathcal{C}_k$ is defined over the set $\mathbf{T}_k = \{[\mathbf{x}] \mid l(\mathbf{x}) < k\} \cup \{\Phi_k\}$, where $\Phi_k = \{x \mid l(x) \geq k\}$. Since all the classes except Φ_k are singletons, we simply write $\mathbf{T}_k = \{\mathbf{x} \mid l(\mathbf{x}) < k\} \cup \{\Phi_k\}$, and refer to elements of $\mathbf{T}_k$ as transients.

Every function $\mathbf{f} : \mathbf{T}^n \to \mathbf{T}$ in $\mathcal{C}$ is now also defined in $\mathcal{C}_k$ as follows: $\mathbf{f}([\mathbf{x}_1], \ldots, [\mathbf{x}_n]) = [\mathbf{f}(\mathbf{y}_1, \ldots, \mathbf{y}_n)]$, where $\mathbf{y}_i$ is any representative of the class $[\mathbf{x}_i]$, for $i \in [n]$.

We define the *prefix order* $\leq$ on $\mathbf{T}_k$: $[\mathbf{y}] \leq [\mathbf{x}]$ if either (a) $[\mathbf{y}]$ and $[\mathbf{x}]$ are singleton classes and $\mathbf{y}$ is a prefix of $\mathbf{x}$, or (b) $[\mathbf{x}] = \Phi_k$. The length $l([\mathbf{x}])$ of a class $[\mathbf{x}]$ is $l(\mathbf{x})$ if $[\mathbf{x}] = \{\mathbf{x}\}$, and $l(\Phi_k) = k$. The *suffix order* $\preceq$ is similar.

A network is initially in the stable total state $(\hat{a}, b)$, that is, $f(\hat{a}, b) = b$, and then the input m-tuple $\hat{a}$ changes to a. If $\hat{a}_i, a_i \in B$, and $\hat{a}_i = a_i$, then $\hat{a}_i \circ a_i = a_i$. Otherwise, $\hat{a}_i \circ a_i = \hat{a}_i a_i$, if $k > 2$, and $\hat{a}_i \circ a_i = \Phi_2$, if $k = 2$. For tuples, the $\circ$ operation is performed component-wise.

Algorithm A is shown below, where $\mathbf{f} : \mathbf{T}_k^{m+n} \to \mathbf{T}_k^n$ is the tuple of excitation functions. Each gate variable is non-decreasing in the prefix order in this algorithm, and the algorithm always terminates. Its result is denoted by $\mathbf{y}^A$. It follows from [2] that $\mathbf{y}^A$ is the least fixed point of the function $\mathbf{f}(\mathbf{a}, \mathbf{x})$ over b with respect to the prefix order, i.e., $\mathbf{f}(\mathbf{a}, \mathbf{y}^A) = \mathbf{y}^A$ and $\mathbf{y} \geq b \;\&\; \mathbf{f}(\mathbf{a}, \mathbf{y}) = \mathbf{y} \Rightarrow \mathbf{y}^A \leq \mathbf{y}$.

In Algorithm B below, each gate variable is non-increasing in the suffix order. The result of the algorithm is $\mathbf{y}^B$. It follows from [2] that Algorithm B in Algebra $\mathcal{C}_k$ computes the greatest fixed point of function $\mathbf{f}(a, \mathbf{x})$ below $\mathbf{y}^A$ with respect to the suffix order, that is, $\mathbf{f}(a, \mathbf{y}^B) = \mathbf{y}^B$ and $\mathbf{y} \preceq \mathbf{y}^A \;\&\; \mathbf{f}(a, \mathbf{y}) = \mathbf{y} \Rightarrow \mathbf{y}^B \succeq \mathbf{y}$.

Algorithm A

$h := 0$;
$\mathbf{a} := \hat{a} \circ a$;
$\mathbf{y}^0 := b$;
repeat
  $h := h + 1$;
  $\mathbf{y}^h := \mathbf{f}(\mathbf{a}, \mathbf{y}^{h-1})$;
until $\mathbf{y}^h = \mathbf{y}^{h-1}$;

Algorithm B

$h := 0$;
$\mathbf{y}^0 := \mathbf{y}^A$;
repeat
  $h := h + 1$;
  $\mathbf{y}^h := \mathbf{f}(a, \mathbf{y}^{h-1})$;
until $\mathbf{y}^h = \mathbf{y}^{h-1}$;

Complexity issues related to Algorithms A and B are discussed in detail in [2]. In Algebra $\mathcal{C}_k$, Algorithm A runs in $O(m + n^2 k \log k)$ time, where m is the number of network inputs and n is the number of gates; this applies to networks with or without feedback. The total time required for Algorithm B is $O(n^2 k \log k)$.

In Tables 1 and 2, we show the simulations in $\mathcal{C}_2$ and $\mathcal{C}_5$ of the network defined by the excitation equations: $\mathbf{f}_1 = \overline{\mathbf{x}}$, $\mathbf{f}_2 = \mathbf{x} \wedge \mathbf{y}_1$, $\mathbf{f}_3 = \mathbf{x} \vee \mathbf{y}_2$, $\mathbf{f}_4 = \mathbf{y}_3 \vee \mathbf{y}_4$, $\mathbf{f}_5 = \overline{\mathbf{y}_4}$, $\mathbf{f}_6 = \mathbf{y}_4 \wedge \mathbf{y}_5$, $\mathbf{f}_7 = \mathbf{y}_6 \vee \mathbf{y}_7$. From Algorithm A in $\mathcal{C}_2$ we learn that each variable may change during the transition. From Algorithm B we know that $\mathbf{y}_1$ changes from 1 to 0, and $\mathbf{y}_3$, from 0 to 1 (but we don't know whether there are dynamic hazards), there is a static hazard in $\mathbf{y}_2$, and $\mathbf{y}_4, \ldots, \mathbf{y}_7$ may oscillate. From Algorithm A in $\mathcal{C}_5$, we learn that the change in $\mathbf{y}_1$ is free of hazards, the static hazard in $\mathbf{y}_2$ is 010, and $\mathbf{y}_3$ has a dynamic hazard, 0101.

5 Active Paths and Cycles

In this section, the total state is $\mathbf{s} = (\mathbf{s}_1, \ldots, \mathbf{s}_{m+n}) \in \mathbf{T}^{m+n}$. The state of vertex v_i is $\mathbf{s}(v_i) = \mathbf{s}_i$, and the state of arc e from u to v is $\mathbf{s}(e) = \mathbf{s}(t(e))$. There are three stable states: the initial state $(\hat{a}, b)$, the state $(\mathbf{a}, \mathbf{y}^A)$ at the end

Table 1. Simulation in $\mathcal{C}_2$.

	$\mathbf{x}$	$\mathbf{y}_1$	$\mathbf{y}_2$	$\mathbf{y}_3$	$\mathbf{y}_4$	$\mathbf{y}_5$	$\mathbf{y}_6$	$\mathbf{y}_7$
$(\hat{a}, b)$	0	1	0	0	0	1	0	0
	Φ_2	1	0	0	0	1	0	0
	Φ_2	Φ_2	Φ_2	Φ_2	0	1	0	0
	Φ_2	Φ_2	Φ_2	Φ_2	Φ_2	1	0	0
	Φ_2	Φ_2	Φ_2	Φ_2	Φ_2	Φ_2	Φ_2	0
$(\mathbf{a}, \mathbf{y}^A)$	Φ_2	Φ_2	Φ_2	Φ_2	Φ_2	Φ_2	Φ_2	Φ_2
	1	Φ_2	Φ_2	Φ_2	Φ_2	Φ_2	Φ_2	Φ_2
	1	0	Φ_2	1	Φ_2	Φ_2	Φ_2	Φ_2
$(a, \mathbf{y}^B)$	1	0	0	1	Φ_2	Φ_2	Φ_2	Φ_2

Table 2. Simulation in $\mathcal{C}_5$.

	$\mathbf{x}$	$\mathbf{y}_1$	$\mathbf{y}_2$	$\mathbf{y}_3$	$\mathbf{y}_4$	$\mathbf{y}_5$	$\mathbf{y}_6$	$\mathbf{y}_7$
$(\hat{a}, b)$	0	1	0	0	0	1	0	0
	01	1	0	0	0	1	0	0
	01	10	01	01	0	1	0	0
	01	10	010	01	01	1	0	0
	01	10	010	0101	01	10	01	0
	01	10	010	0101	0101	10	010	01
	01	10	010	0101	Φ_5	1010	Φ_5	0101
$(\mathbf{a}, \mathbf{y}^A)$	01	10	010	0101	Φ_5	Φ_5	Φ_5	Φ_5
	1	10	010	0101	Φ_5	Φ_5	Φ_5	Φ_5
	1	0	10	1	Φ_5	Φ_5	Φ_5	Φ_5
$(a, \mathbf{y}^B)$	1	0	0	1	Φ_5	Φ_5	Φ_5	Φ_5

of Algorithm A, and the state $(a, \mathbf{y}^B)$ at the end of Algorithm B. We refer to these as stable states b, $\mathbf{y}^A$ and $\mathbf{y}^B$.

Let $T = v_0, e_1, y_1, \ldots, e_r, y_r$ be a path or cycle in a network, where v_0 is an input or a gate, and $y_1, \ldots, y_r$ are gates. Any arc joining a node $w \notin \{v_0, y_1, \ldots, y_r\}$ to a node y_i, $1 \leq i \leq r$ is a *side input* of gate y_i. Let $e'_i = e'_{i_1}, \ldots, e'_{i_{n_i-1}}$ be the $(n_i - 1)$-tuple of side inputs of gate y_i; the inputs of gate y_i are the side inputs together with the *main input* e_i: the arc from v_0 to y_1, if $i = 1$, and the arc from y_{i-1} to y_i, otherwise. There is no loss of generality in assuming that the main input is the last input of y_i.

Definition 2 *An arc e from u to v is* active *in stable state* $\mathbf{s}$ *if no side input to gate v is dominant in* $\mathbf{s}$*; otherwise, e is* inactive. *A path or cycle $T = v_0, e_1, v_1, \ldots, e_r, v_r$ is* active *in stable state* $\mathbf{s}$ *if all of its arcs are active. We say that vertex v_i* activates *vertex v_j in stable state* $\mathbf{s}$ *if there is an active path or cycle of length greater than 0 from v_i to v_j.*

'Activates' is a transitive relation. An arc leading to an identity gate or an inverter is always active, since such a gate has no side inputs. An arc leading to a XOR gate is always active, since XOR has no dominant inputs.

In a cycle $C = v_0, e_1, v_1, \ldots, e_r, v_0$, if C is active in some state, then every vertex activates itself, and 'activates' is reflexive. If v_i and v_j are two vertices in C, then v_i activates v_j, and *vice versa*; thus 'activates' is also symmetric, and hence an equivalence relation.

Proposition 6 *Let N be a network with excitation functions in $\mathcal{G}$. If N is in a stable state* $\mathbf{s}$ *and C is an active cycle, then all the vertices in C have transients of the same length.*

6 Interior Values

Let $k \geq 2$; a value $\mathbf{x} \in \mathbf{T}_k$ is *exterior* if $\mathbf{x} \in \{0, 1, \Phi_k\}$; otherwise, it is *interior*. In ternary algebra, $k = 2$ and there are no interior values.

Lemma 3 *If a vertex has an interior value in $\mathbf{y}^B$, then it has the same value in $\mathbf{y}^A$.*

Proof: Since Algorithm B is non-increasing in the suffix order, if a variable has values $\mathbf{s}_i$ in $\mathbf{y}^A$ and $\mathbf{t}_i$ in $\mathbf{y}^B$, then $\mathbf{t}_i$ is a nonempty suffix of $\mathbf{s}_i$. If $\mathbf{y}^B(v_i)$ is interior, then $\mathbf{y}^A(v_i)$ could only be Φ_k or interior. If $\mathbf{y}^A(v_i) \neq \mathbf{y}^B(v_i)$, the length of v_i must decrease during Algorithm B. Thus suppose the value of v_i is the same at step $j-1$ of Algorithm B as it is in $\mathbf{y}^A$, but the length of v_i decreases in step j. By Proposition 5, v_i has a dominant input in step $j-1$. But then v_i also has that dominant input in $\mathbf{y}^B$, since a binary value cannot change in Algorithm B. This would make $\mathbf{y}^B(v_i)$ binary, contradicting that $\mathbf{y}^B(v_i)$ is interior. Thus the value of v_i in $\mathbf{y}^A$ is the same as in $\mathbf{y}^B$. □

Let I be the set of all vertices of a network N that have interior values in $\mathbf{y}^B$. For $u, v \in I$, let $u \sim v$ if $u = v$ or u activates v and v activates u. Clearly, $\sim$ is an equivalence relation on I. Equivalence class $[u]$ *activates* equivalence $[v]$, if and only if $u = v$ or u activates v. The relation 'activates' on $I/\sim$ is a partial order. An equivalence class $[u]$ is *primary* in this partial order if $[v]$ activates $[u]$ implies $[v] = [u]$. Since we are dealing with finite network graphs, there always exists at least one primary class. A primary class cannot consist of a single state that does not appear in any cycle, because such a vertex must depend only on the external network inputs, and its value is binary in $\mathbf{y}^B$.

Theorem 1 *For every $k > 2$, the result of Algorithm B is the same in $\mathcal{C}_k$ as in ternary simulation.*

Proof: Let P be a primary equivalence class of interior values in $\mathbf{y}^B$. All the vertices of P are binary in state b, but interior in $\mathbf{y}^B$ and, by Lemma 3, also interior in $\mathbf{y}^A$. Suppose the values of all the vertices in P are the same at step $j-1$ of Algorithm A as they are in b, but v_i is interior in step j. Then at least one of the vertices, say $u_i \notin P$, joined

by an arc to v_i has an interior value in step $j-1$, and also in $\mathbf{y}^A$. (It cannot be Φ_k, for then v_i would be Φ_k in $\mathbf{y}^A$.)

Now $\mathbf{y}^B(u_i)$ cannot be Φ_k, for then the values in P would be Φ_k, by Corollary 1. It cannot be interior, for then it would be in another equivalence class Q that activates P, contradicting that P is primary. Thus $\mathbf{y}^B(u_i)$ must be binary. Since Algorithm B is non-increasing in the suffix order, $\mathbf{y}^A(u_i)$ ends in $\mathbf{y}^B(u_i)$. Since Algorithm A is non-decreasing in the prefix order, $\mathbf{y}^A(u_i)$ begins with $b(u_i)$.

Suppose first that v_i is an OR gate. Because v_i belongs to an active cycle in $\mathbf{y}^B$, we have $\mathbf{y}^B(u_i) = 0$, as shown in Fig. 1(c) and (f). (The gate may have other inputs that are not shown; all such inputs must be 0 in $\mathbf{y}^B$.) Let the in-cycle predecessor of v_i be v_{i-1}; in case the cycle consists of a single state, we have $v_i = v_{i-1}$. Clearly, v_{i-1} is also in P, and cannot be 1 in state b, because this would prevent v_i from changing. Hence $b(v_{i-1}) = 0$, as in Fig. 1(a) and (d).

Case 1: $b(v_i) = 0$. Since $(\hat{a}, b)$ is stable, $b(u_i) = 0$, as in Fig. 1(a). Since $\mathbf{y}^A(u_i)$ must begin with $b(u_i)$, end with $\mathbf{y}^B(u_i)$ and be of length at least 2, $\mathbf{y}^A(u_i)$ has the form $(01)^h0$, where $h \geq 1$, as in Fig. 1(b). Also, $\mathbf{y}^A(v_{i-1})$ must begin with $b(v_{i-1})$, and be of length at least 2; thus v_{i-1} begins with 01. Since v_i has no dominant inputs, Part 1 of Lemma 1 does not apply. Also, the inputs $01\mathbf{x}$ and $(01)^h0$ do not fit any of the patterns of Part 2. Thus Part 3 applies, v_i cannot be stable in $\mathbf{y}^A$, and this case cannot occur.

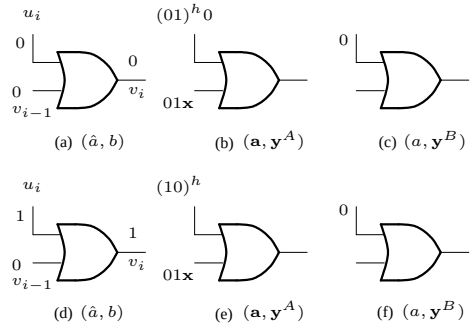

Figure 1. OR **gate in an interior cycle.**

Case 2: $b(v_i) = 1$. Then there is a vertex, say u_i, connected to v_i by a side-input arc, that is initially 1 (there may be other such vertices) and is interior in step $j-1$ (as must be the other such vertices). Now $\mathbf{y}^A(u_i)$ begins with 1, ends with 0 and is of length at least 2; thus it has the form $(10)^h$, where $h \geq 1$. As in Case 1, v_{i-1} must be of the form $01\mathbf{x}$. Part 1 of Lemma 1 does not apply. If $h > 1$, then Part 2 does not apply. Thus v_i cannot be stable in $\mathbf{y}^A$, and this case cannot occur. There remains the possibility that $h = 1$, that is, $\mathbf{y}^A(u_i) = 10$. By Part 2 of Lemma 1, v_{i-1} must be of the form $(10)^h$ or $(10)^h1$, $h \geq 1$, for in all other cases v_i cannot be stable in $\mathbf{y}^A$. This contradicts that v_{i-1} begins with 0, and this case cannot occur. In summary, the gate which changes first in Algorithm A cannot be an OR gate.

Now suppose that v_i is a XOR gate; then it has no dominant inputs. Since $\mathbf{y}^A(u_i)$ is interior, by Lemma 2, the cycle cannot be stable in $\mathbf{y}^A$. This is a contradiction.

The argument for other functions in $\mathcal{G}$ follows from Propositions 1 and 2. Altogether, we have shown that no variable can have an interior value in $\mathbf{y}^B$. It is now easy to verify that Algorithm B in $\mathcal{C}_2$ produces the same value for each variable as Algorithm B in $\mathcal{C}_k$ for every $k > 2$. □

7 Conclusions

We have the following three types of simulation results:
1) Algorithm A terminates in $\mathcal{C}$, and Algorithm B is unnecessary. There may be static and dynamic hazards, but no oscillations. Every feedback-free circuit has this behavior.
2) Algorithm A does not terminate in $\mathcal{C}$. For any k, after Algorithm A in $\mathcal{C}_k$, Algorithm B results in binary values. The circuit has feedback, but its behavior is combinational. Hazards and transient oscillations [3] may exist.
3) Algorithm A does not terminate in $\mathcal{C}$. For any k, after Algorithm A in $\mathcal{C}_k$, Algorithm B results in at least one Φ_k. Thus, at least one variable has a nontransient oscillation.

In view of our result, the worst-case time complexity of Algorithm B can be reduced from $O(n^2k \log k)$ to $O(2n^2)$, if ternary algebra $\mathcal{C}_2$ is used instead of $\mathcal{C}_k$.

Acknowledgement This research was supported by the Natural Sciences and Engineering Research Council of Canada under grant No. OGP0000871 and under a Postgraduate Scholarship, and by a Graduate Award from the Department of Computer Science, University of Toronto.

References

[1] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications* American Elsevier, 1976.

[2] J. A. Brzozowski and Z. Ésik, "Hazard Algebras," *Formal Methods in System Design,* **23(3)**, pp. 223–256, 2003.

[3] J. A. Brzozowski and C-J. H. Seger, *Asynchronous Circuits*, Springer, 1995.

[4] E. B. Eichelberger, "Hazard detection in combinational and sequential switching circuits," *IBM J. Research and Development,* **9**, pp. 90–99, 1965.

[5] M. Gheorghiu and J. Brzozowski, "Simulation of feedback-free circuits in the algebra of transients," *Int. J. of Found. of Comp. Sci.,* **14(6)**, 1033–1054, 2003.

[6] M. D. Riedel, *Cyclic Combinational Circuits*, PhD Dissertation, Department of Electrical Engineering, California Institute of Technology, May 2004.

[7] T. R. Shiple, "Formal Analysis of Synchronous Circuits," PhD Thesis, Univ. of California, Berkeley 1996.

[8] Y. Ye and J. A. Brzozowski, "Covering of Transient Simulation of Feedback-Free Circuits by Binary Analysis," *Int. J. Found. of Comp. Sci.*, **17(4)**, 949-973, 2006.