



## Hazard Algebras\*

J. BRZOZOWSKI  
*School of Computer Science, University of Waterloo, Canada*

brzozo@uwaterloo.ca

Z. ÉSIK  
*Department of Computer Science, University of Szeged, Hungary*

ze@inf.u-szeged.hu

*Received October 30, 2000; Revised December 18, 2001; Accepted July 9, 2002*

**Abstract.** We introduce algebras capable of representing, detecting, identifying, and counting static and dynamic hazard pulses that can occur in the worst case on any wire in a gate circuit. These algebras also permit us to count the worst-case number of signal changes on any wire. This is of interest to logic designers for two reasons: each signal change consumes energy, and unnecessary multiple signal changes slow down the circuit operation. We describe efficient circuit simulation algorithms based on our algebras and illustrate them by several examples. Our method generalizes Eichelberger's ternary simulation and several other algebras designed for hazard detection.

**Keywords:** algebra, circuit, gate, hazard, multivalued, simulation, transient

### 1. Introduction

The problem of hazards, i.e., unwanted short pulses on the outputs of gates in logic circuits, is of great importance. In an asynchronous circuit a hazard pulse may cause an error in the circuit operation. Synchronous circuits are protected from such errors, since all actions are controlled by a common clock, and all combinational circuits stabilize before the clock pulse arises. However, an unwanted change in a signal increases the energy consumption in the circuit. From the energy point of view it is necessary not only to detect the presence of unwanted signal changes, but also to count them, in order to obtain an estimate of the energy consumption. Such unwanted changes also add to the computation time. In this paper we address the problem of counting hazards and signal changes in gate circuits.

One of the earliest simulation methods for hazard detection is ternary simulation. For the detection of hazards, ternary algebra has been used since 1948 [16]. Ternary simulation was then used by many authors; see, for example, [3] for a list of early references on this subject, and also [8] for a detailed discussion of ternary methods. A two-pass ternary simulation method was introduced by Eichelberger in 1965 [12], and later studied by others [8]. Ternary simulation is capable of detecting static hazards and oscillations, but does not detect dynamic hazards. A quinary algebra was proposed by Lewis in 1972 for the detection of dynamic

\*This research was supported by Grant No. OGP0000871 from the Natural Sciences and Engineering Research Council of Canada, and Grant No. T30511 from the National Foundation of Hungary for Scientific Research. An extended abstract of this paper has been presented at the conference *Half Century of Automata Theory*, London, ON, July 26, 2000 [5].

hazards. A survey of various simulation algebras for hazard detection was given in 1986 by Hayes [18]. We show that several of these algebras are special cases of the hazard algebra presented here.

The remainder of the paper is structured as follows. In Section 2 we define our model of gate circuits, describe the binary analysis method, and define hazards. In Section 3 we discuss our representation of transients in gate circuits. Using this representation, in Section 4 we define Algebra  $C$  capable of counting an arbitrary number of signal changes on any wire in a gate circuit. To make the algebra more applicable, in Section 5 we modify it to Algebra  $C_k$ , where  $k$  is any positive integer; such an algebra is capable of counting and identifying up to  $k - 1$  signal changes. In Section 6 we describe circuit simulation algorithms based on Algebras  $C$  and  $C_k$ , and we illustrate our method by several examples. In Section 7 we extend our definitions to arbitrary Boolean functions. Complexity issues are then treated in Section 8. In Section 9 we discuss simulation with initial, middle, and final values, and Section 10 concludes the paper. Several additional results about algebra  $C$  and three proofs related to Section 7 are given in the appendix.

Characterizations of the simulation results are briefly mentioned at the end of Section 6, and are treated further in [7, 14, 15].

## 2. Static and dynamic hazards in logic circuits

We use  $\vee$ ,  $\wedge$ , and  $\neg$  for the Boolean OR, AND, and NOT operations, respectively.

Figure 1 shows a gate circuit that we will use several times to illustrate various concepts. We assume in this example that gates have delays, but wire delays are negligible. Hence, the internal state of the circuit is represented by the state of the four gate outputs  $s_1$ ,  $s_2$ ,  $s_3$ , and  $s_4$ . If the input  $X$  is 0 and the internal state is  $(s_1, s_2, s_3, s_4) = (1, 0, 0, 0)$ , each gate is stable, since the value of its output agrees with the value of the function computed by the gate. Thus, the inverter is stable, because  $s_1 = 1 = \bar{X}$ , the top AND gate is stable, because  $s_3 = 0 = 1 \wedge 0 = s_1 \wedge X$ , etc.

Now suppose that the input changes to  $X = 1$  and is held constant at that value. It is clear that eventually  $s_1$  becomes 0; consequently  $s_3$  remains 0. Also, since  $s_2$  becomes 1,  $s_4$  becomes 1 as well. Thus we know that the final state of the circuit is the stable state  $(s_1, s_2, s_3, s_4) = (0, 1, 0, 1)$ . Assume that the specification of this circuit requires that, when the initial state is  $(1, 0, 0, 0)$  and the input changes from 0 to 1,  $s_1$  should change once from 1 to 0,  $s_2$  and  $s_4$  should change once from 0 to 1, and  $s_3$  should not change. Does the circuit satisfy this specification?

If we take a closer look at the analysis above, we discover that the behavior of a circuit depends very much on the delays of its components. In the theory of asynchronous circuits,

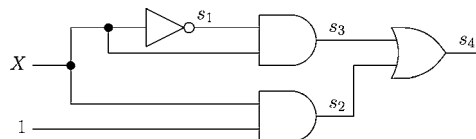


Figure 1. Circuit with hazards.

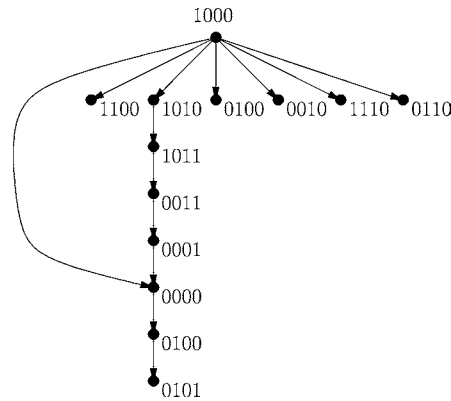


Figure 2. Analysis of circuit with hazards.

it is usually assumed that the sizes of these delays are not known [8, 20, 21, 23]. Therefore, the analysis considers all possible relative sizes of the delays. In our example, the initial state is  $(1, 0, 0, 0)$ . After the input change  $s_1$ ,  $s_2$ , and  $s_3$  are all unstable. If the delay of  $s_1$  is smaller than those of  $s_2$  and  $s_3$ , the next state is  $(0, 0, 0, 0)$ . If the delay of  $s_2$  is the smallest of the three, then the state becomes  $(1, 1, 0, 0)$ . If the delay of  $s_1$  is equal to that of  $s_2$  and smaller than that of  $s_3$ , state  $(0, 1, 0, 0)$  is reached. If all three delays are equal, the state becomes  $(0, 1, 1, 0)$ , etc. Altogether, there are seven possible successor states, as shown in figure 2, where we have deleted the parentheses and commas from the state tuples, for simplicity. Thus, 0100 represents  $(0, 1, 0, 0)$ , etc.

The binary analysis method described above is quite old—see, for example, the work of Muller [20, 21]—and the details are outside the scope of the present paper. We refer the reader to [8] for a formal treatment of this topic. Here we give only an example illustrating the main ideas. The general analysis step consists of finding all possible successors of a given state  $s = (s_1, \dots, s_n)$  in a circuit with  $n$  (internal) state variables. Any state obtained from  $s$  by changing any nonzero number of unstable variables is a possible successor. In this way we obtain a directed graph showing all the possible paths from the initial state. The graph is finite, since the total number of states is finite. Part of the behavior graph of our example circuit is shown in figure 2.

If we continue the analysis for our example, we find that one possible path is

$$p = 1000, 0000, 0100, 0101,$$

where the last state (0101) is stable. Along  $p$ , variable  $s_1$  changes once from 1 to 0, variables  $s_2$  and  $s_4$  change once from 0 to 1, and  $s_3$  does not change. This is in agreement with the specification. This path occurs when the initial “race” among the first three variables is “won” by  $s_1$ , that is, when the delay of  $s_1$  is the shortest of the three.

The Boolean function computed by a gate is called its “excitation.” Our model uses so called “inertial” delays [8, 23], in which short pulses in the excitation are ignored. For

example, when the input is  $X = 1$  and the internal state is 1000, variable  $s_3$  is unstable because both of its inputs are 1. Variables  $s_1$  and  $s_2$  are also unstable. If  $s_1$  wins this race, state 0000 is reached, and now  $s_3$  is no longer unstable. By assumption, the delay of  $s_3$  is longer than the period of time during which  $s_3$  had two 1s on its inputs; this period is the same as the delay of the fastest gate  $s_1$ . Thus the short pulse in the excitation has been ignored. This is in contrast to “pure” or “ideal” delay models [8, 23], in which every change in the excitation causes a corresponding change in the output.

Another possible path in figure 2 is

$$q = 1000, 1010, 1011, 0011, 0001, 0000, 0100, 0101.$$

Along  $q$  the inverter output  $s_1$  changes only once from 1 to 0, and variable  $s_2$  changes only once from 0 to 1. Variable  $s_3$  is 0 in both the initial and final state; thus, its value should be “static.” However, there is a time during which  $s_3$  has the value 1. This represents a “hazardous” behavior, since a device having  $s_3$  as input may react to this 1-signal, although this signal is not in the specification. Such a behavior is called a “static hazard.” Variable  $s_4$  has initial value 0 and final value 1; its behavior is therefore “dynamic.” However, it changes from 0 to 1 twice, and this represents a “dynamic hazard.”

In summary, our example circuit may or may not satisfy its specification, depending on the relative sizes of its delays.

We refer to the behavior of a circuit after an input change from a stable state as a “transition.” The set of states in which a circuit can be at the end of a transition is called the “outcome” of that transition. This concept is needed for the formal definition of hazards [8]. In our example, the outcome is the singleton set containing the stable state (0, 1, 0, 1). In general, the outcome may also contain states that appear in cycles, which represent oscillations. A *static hazard* is said to be present in a transition if there is a state variable that has the same value in all the states of the outcome as it has in the initial state, and there exists a path from the initial state to a state in the outcome along which the variable changes (necessarily an even number of times). A *dynamic hazard* exists if there is a variable which has some value  $v$  in the initial state and the complementary value  $\bar{v}$  in all the states of the outcome, but changes at least three times along some path from the initial state to a state in the outcome.

The binary analysis method above is exponential in the number of state variables. One objective of this paper is to find a more efficient method for detecting hazards. This method is described in Section 6.

### 3. Transients

We use waveforms to represent changing binary signals. In particular, we are interested in studying transient phenomena in circuits. For this application we consider waveforms with a constant initial value, a transient period involving a finite number of changes, and a constant final value. Waveforms of this type will be called *transients*.

Figure 3 gives four examples of transients. With each such transient we associate a binary word, i.e., a sequence of 0s and 1s, in a natural way. In this binary word a 0 (1) represents

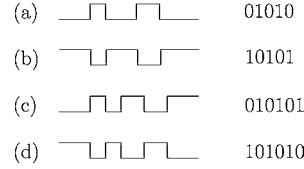


Figure 3. Transients: (a) constant 0 with static hazards; (b) constant 1 with static hazards; (c) change from 0 to 1 with dynamic hazards; (d) change from 1 to 0 with dynamic hazards.

a maximal interval during which the signal has the value 0 (1). Such an interval is called a *0-interval* (*1-interval*). Of course, no timing information is represented by the binary word. This is to our advantage, however, since we assume that the changes can happen at any time and that the intervals between successive changes can vary arbitrarily.

In general, transients are of the following types:

- If a signal is supposed to have the constant value 0, but has  $i \geq 0$  1-intervals, these intervals represent  $i$  static-hazard pulses. This transient is denoted by the word  $0(10)^i = (01)^i 0$ , has  $2i$  unwanted signal changes, and  $(i + 1)$  0-intervals. In regular expression notation [2, 8, 22], the set of all words of this type is  $0(10)^* = (01)^* 0$ .
- If a signal is supposed to have the constant value 1, but has  $i \geq 0$  0-intervals, these intervals represent  $i$  static-hazard pulses. This transient is denoted by the word  $1(01)^i = (10)^i 1$ , has  $2i$  unwanted signal changes, and  $(i + 1)$  1-intervals. The set of all words of this type is  $1(01)^* = (10)^* 1$ .
- If a signal is supposed to change from 0 to 1, but has  $i \geq 0$  unwanted 0-intervals after the first change, these 0-intervals represent  $i$  dynamic-hazard pulses. Such a transient is denoted by  $01(01)^i = 0(10)^i 1 = (01)^i 01$ , has  $2i$  unwanted signal changes,  $(i + 1)$  1-intervals, and  $(i + 1)$  0-intervals. The set of all words of this type is  $01(01)^* = 0(10)^* 1 = (01)^* 01$ .
- If a signal is supposed to change from 1 to 0, but has  $i \geq 0$  unwanted 1-intervals after the first change, these 1-intervals represent  $i$  dynamic-hazard pulses. Such a transient is denoted by  $10(10)^i = 1(01)^i 0 = (10)^i 10$ , has  $2i$  unwanted signal changes,  $(i + 1)$  1-intervals, and  $(i + 1)$  0-intervals. The set of all words of this type is  $10(10)^* = 1(01)^* 0 = (10)^* 10$ .

For the present, we assume that our circuits are constructed with 2-input OR gates, 2-input AND gates and inverters; these restrictions will be removed in Section 7. Given a transient at each input of a gate, we wish to find the longest possible transient at the output of that gate. The case of the inverter is the easiest one. If  $t = a_1 \dots a_j$  is the binary word of a transient at the input of an inverter, then its output has the transient  $\bar{t} = \bar{a}_1 \dots \bar{a}_j$ . For example, in figure 3, the first two transients are complementary, as are the last two.

For the OR and AND gates, we assume that the changes in each input signal can occur at arbitrary times. The following proposition permits us to find the largest number of changes possible at the output of an OR gate.

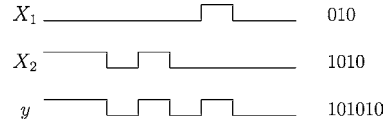


Figure 4. 0-intervals in OR GATE.

**Proposition 3.1.** *If the two inputs of an OR gate have  $m$  and  $n$  0-intervals respectively, then the maximum number of 0-intervals in the output signal is 0 if  $m = 0$  or  $n = 0$ , and is  $m + n - 1$ , otherwise.*

**Proof:** We postpone the proof until Section 7, where it is shown that this proposition is a special case of a result concerning OR gates with an arbitrary number of inputs.  $\square$

*Example 1.* Figure 4 shows waveforms of two inputs  $X_1$  and  $X_2$  and output  $y$  of an OR gate. The input transients are 010 and 1010, and the output transient is 101010. Here, the inputs have two 0-intervals each, and the output has three 0-intervals, as predicted by Proposition 3.1.

By an argument similar to that for Proposition 3.1, we obtain:

**Proposition 3.2.** *If the two inputs of an AND gate have  $m$  and  $n$  1-intervals respectively, then the maximum number of 1-intervals in the output signal is 0 if  $m = 0$  or  $n = 0$ , and is  $m + n - 1$ , otherwise.*

These two results will be used in the next section to define operations on transients.

#### 4. Change-counting algebra

Let  $T = 0(10)^* \cup 1(01)^* \cup 0(10)^*1 \cup 1(01)^*0$ ; this is the set of all nonempty words over 0 and 1, in which no two consecutive letters are the same. As explained above, elements of  $T$  are called transients. Note that every transient is uniquely determined by its first letter and length, by its last letter and length, by its first and last letters and the number of 0s, and by its first and last letters and the number of 1s. These characterizations may help the reader in understanding some proofs that follow.

We define the (signal) change-counting algebra  $C = (T, \oplus, \otimes, \bar{\phantom{x}}, 0, 1)$ . For any  $t \in T$  define  $z(t)$  ( $z$  for zeros) and  $u(t)$  ( $u$  for units) to be the number of 0s in  $t$  and the number of 1s in  $t$ , respectively. Let  $\alpha(t)$  and  $\omega(t)$  be the first and last letters of  $t$ , and let  $l(t)$  denote the length of  $t$ . For example, if  $t = 10101$ , then  $z(t) = 2$ ,  $u(t) = 3$ ,  $\alpha(t) = \omega(t) = 1$ , and  $l(t) = 5$ .

Operations  $\oplus$  and  $\otimes$  are binary operations on  $T$  intended to represent the worst-case OR-ing and AND-ing of two transients at the inputs of a gate. They are defined as follows:

$$t \oplus 0 = 0 \oplus t = t, \quad t \oplus 1 = 1 \oplus t = 1,$$

for any  $t \in T$ . Furthermore, if  $w$  and  $w'$  are words in  $T$  of length  $>1$ , their *sum*, denoted by  $w \oplus w'$ , is defined as the unique word  $t$  that begins with  $\alpha(w) \vee \alpha(w')$ , ends with  $\omega(w) \vee \omega(w')$  and has  $z(t)$  0s, where  $z(t) = z(w) + z(w') - 1$ . By Proposition 3.1,  $t$  is the longest transient that can be produced at the output of an OR gate, if transients  $w$  and  $w'$  appear at the inputs of the OR gate. For example,  $010 \oplus 1010 = 101010$ , as illustrated in figure 4.

Next, define

$$t \otimes 1 = 1 \otimes t = t, \quad t \otimes 0 = 0 \otimes t = 0,$$

for any  $t \in T$ . Consider now the *product* of two words  $w, w' \in T$  of length  $>1$ , and denote this product by  $t = w \otimes w'$ . Then  $t$  is the unique word in  $T$  that begins with  $\alpha(w) \wedge \alpha(w')$ , ends with  $\omega(w) \wedge \omega(w')$ , and has  $u(t) = u(w) + u(w') - 1$ , by Proposition 3.2. For example,  $0101 \otimes 10101 = 01010101$ .

The (*quasi-*)*complement*  $\bar{t}$  of a word  $t \in T$  is obtained by complementing each letter in  $t$ . For example,  $\bar{1010} = 0101$ . Finally, the constants 0 and 1 of  $C$  are the words 0 and 1 of length 1.

In this paper we use several algebraic structures. These structures are fully defined in the paper; consequently, the paper is self-contained. For more details and background material, we refer the reader to a text on universal algebra, for example [9, 17]. Also, a recent survey of various algebraic structures used for hazard detection and of the algebraic properties these structures satisfy appears in [6].

A *commutative bisemigroup* is an algebra  $C = (S, \oplus, \otimes)$ , where  $S$  is a set, and  $\oplus$  and  $\otimes$  are associative and commutative binary operations on  $S$ , i.e.,  $(S, \oplus)$  and  $(S, \otimes)$  are both commutative semigroups with the same underlying set. Thus a commutative bisemigroup satisfies equations L1, L2, L1', L2' in Table 1, where the laws are listed in dual pairs. A commutative bisemigroup is *de Morgan* if it has two constants 0 and 1, and a unary operation  $\bar{\phantom{x}}$  satisfying L3–L6, L3', L4', and L6'. All the laws of Table 1 are also satisfied by Boolean algebras, but several laws of Boolean algebras are not necessarily satisfied by de Morgan bisemigroups. Thus, de Morgan bisemigroups are generalizations of Boolean algebras.

Note that the set of laws in Table 1 is redundant. For example, one can obtain all the primed laws from L1–L6. Also,  $\bar{0} = 1$  and  $\bar{1} = 0$  hold in any commutative de Morgan bisemigroup.

Table 1. Laws of change-counting algebra.

L1	$x \oplus y = y \oplus x$	L1'	$x \otimes y = y \otimes x$
L2	$x \oplus (y \oplus z) = (x \oplus y) \oplus z$	L2'	$x \otimes (y \otimes z) = (x \otimes y) \otimes z$
L3	$x \oplus 1 = 1$	L3'	$x \otimes 0 = 0$
L4	$x \oplus 0 = x$	L4'	$x \otimes 1 = x$
L5	$\bar{\bar{x}} = x$		
L6	$\overline{x \oplus y} = \bar{x} \otimes \bar{y}$	L6'	$\overline{x \otimes y} = \bar{x} \oplus \bar{y}$

**Proposition 4.1.** *The change-counting algebra  $C = (T, \oplus, \otimes, \bar{\cdot}, 0, 1)$ , is a commutative de Morgan bisemigroup, i.e., it satisfies the equations of Table 1.*

**Proof:** Operation  $\oplus$  is commutative by definition. Clearly, associativity holds if one of the components is 0 or 1. Suppose now that  $u, v, w$  are all of length  $> 1$ . Then, using the associativity of disjunction,

$$\begin{aligned}\alpha(u \oplus (v \oplus w)) &= \alpha(u) \vee (\alpha(v) \vee \alpha(w)) \\ &= (\alpha(u) \vee \alpha(v)) \vee \alpha(w) \\ &= \alpha((u \oplus v) \oplus w).\end{aligned}$$

Similarly,  $\omega(u \oplus (v \oplus w)) = \omega((u \oplus v) \oplus w)$ . Thus we have  $u \oplus (v \oplus w) = (u \oplus v) \oplus w$  if the two words have the same number of zeros. But

$$\begin{aligned}z(u \oplus (v \oplus w)) &= z(u) + z(v \oplus w) - 1 \\ &= z(u) + (z(v) + z(w) - 1) - 1 \\ &= (z(u) + z(v) - 1) + z(w) - 1 \\ &= z(u \oplus v) + z(w) - 1 \\ &= z((u \oplus v) \oplus w).\end{aligned}$$

Laws L3 and L4 are obvious by definition of  $\oplus$ , and law L5 is immediate by the definition of quasi-complementation. Also L6 is clear when one of the two words has length 1. Suppose now that  $x, y$  have length  $> 1$ . Then, using de Morgan's law for the Boolean operations,

$$\begin{aligned}\alpha(\overline{x \oplus y}) &= \overline{\alpha(x) \vee \alpha(y)} \\ &= \overline{\alpha(x)} \wedge \overline{\alpha(y)} \\ &= \alpha(\bar{x} \otimes \bar{y}).\end{aligned}$$

In the same way,  $\omega(\overline{x \oplus y}) = \omega(\bar{x} \otimes \bar{y})$ . Since the number of ones in  $\overline{x \oplus y}$  is  $z(x) + z(y) - 1 = u(\bar{x}) + u(\bar{y}) - 1$ , it follows that  $\overline{x \oplus y} = \bar{x} \otimes \bar{y}$ . Finally, laws L1'–L4', and L6' can be derived from L1–L6.  $\square$

We note some further properties of the operations in  $C$ . For any two binary words  $t$  and  $t'$ , we denote by  $tt'$  the word obtained by concatenating  $t$  and  $t'$ , i.e., by writing the letters of  $t'$  after those of  $t$ . We say that  $t$  is a *prefix* of  $t'$  if there exists a (possibly empty) binary word  $t''$  such that  $t' = tt''$ . The prefix relation is a partial order on the set of binary words, i.e., it is reflexive, antisymmetric and transitive. We use  $\leq$  to denote the prefix order.

The prefix order restricted to  $T$  is represented by the inequalities below together with the reflexive and transitive laws:

$$\begin{aligned}0 &\leq 01 \leq 010 \leq 0101 \leq 01010 \leq \dots, \\ 1 &\leq 10 \leq 101 \leq 1010 \leq 10101 \leq \dots.\end{aligned}$$



Recall that a function  $f(x_1, \dots, x_n)$  is *nondecreasing* or *monotonic* with respect to a partial order  $\leq$  if and only if

$$x_1 \leq x'_1, \dots, x_n \leq x'_n \text{ implies } f(x_1, \dots, x_n) \leq f(x'_1, \dots, x'_n).$$

Similarly,  $f$  is *nonincreasing* if

$$x_1 \leq x'_1, \dots, x_n \leq x'_n \text{ implies } f(x_1, \dots, x_n) \geq f(x'_1, \dots, x'_n).$$

For example, we know that  $01 \otimes 10 = 010$ . Since  $\otimes$  is monotonic, as will be shown below, and since  $01 \leq 0101$  and  $10 \leq 10101$ , we know that  $010$  is a prefix of the result  $0101 \otimes 10101$ . In fact that result is  $01010101$ .

**Proposition 4.2.** *The  $\oplus$ ,  $\otimes$  and  $\bar{\phantom{x}}$  operations are monotonic with respect to the prefix order.*

**Proof:** This proposition is a special case of Proposition 7.2, proved in Section 7.  $\square$

The two transients  $01$  and  $10$  play an important role in Algebra  $C$ , since they represent signal changes from  $0$  to  $1$  and from  $1$  to  $0$ , respectively. Suppose that an arbitrary transient  $t$  occurs at one input of a gate, and a single change ( $01$  or  $10$ ) occurs at the other input. The next lemma completely characterizes the output of the OR and AND gates for these inputs.

**Lemma 4.3.** *For all words  $t \in T$ ,*

1. *If  $\alpha(t) = 0$ , then  $t \oplus 10 = 1t = \bar{t}\omega(t)$ ; otherwise,  $t \oplus 10 = t$ .*
2. *If  $\alpha(t) = 1$  then  $t \otimes 01 = 0t = \bar{t}\omega(t)$ ; otherwise,  $t \otimes 01 = t$ .*
3. *If  $\omega(t) = 0$  then  $t \oplus 01 = t1 = \alpha(t)\bar{t}$ ; otherwise,  $t \oplus 01 = t$ .*
4. *If  $\omega(t) = 1$  then  $t \otimes 10 = t0 = \alpha(t)\bar{t}$ ; otherwise,  $t \otimes 10 = t$ .*
5. *It follows from the observations above that either  $t$  or  $\bar{t}$  is a prefix of  $t \oplus 10$ ,  $t \otimes 01$ ,  $t \oplus 01$ , and  $t \otimes 10$ .*

**Proof:** First, consider the case where  $t$  begins and ends with a  $0$ , i.e., has the form  $t = (01)^i 0$ , for some  $i \geq 0$ . Adding  $10$  to  $t$  produces a word  $w$  that begins with  $1$ , ends with  $0$ , and has  $z(w) = z(t) + z(10) - 1 = z(t)$ , by Proposition 3.1. Therefore,  $w$  must be the word  $w = 1t = 1(01)^i 0 = (10)^i 10 = \bar{t}0 = \bar{t}\omega(t)$ . Similarly, if  $t$  ends with  $1$ , then  $t = (01)^i$ , for some  $i \geq 1$ , and  $w$  must be  $w = 1t = 1(01)^i = (10)^i 1 = \bar{t}\omega(t)$ . If  $t$  begins with a  $1$  and ends with a  $0$ , then adding  $10$  results in  $t$ , since the first letter, last letter, and number of  $0$ s in the output  $w$  is the same as it is in  $t$ . This proves the first claim; the remaining three cases follow by similar arguments. The last claim holds, since we have examined all the possible cases.  $\square$

The next lemma establishes the fact that for any  $t, s \in T$ ,  $t$  or  $\bar{t}$  is always a prefix of  $t \oplus s$ , if  $s \neq 1$ , and  $t$  or  $\bar{t}$  is always a prefix of  $t \otimes s$ , if  $s \neq 0$ . This lemma has an important corollary below.

**Lemma 4.4.** *Suppose that  $t, s \in T$ .*

1. *If  $\alpha(s) = 0$ , then  $t \leq t \oplus s$ .*

2. If  $\alpha(s) = 1$ , then  $t \leq t \otimes s$ .
3. If  $\alpha(s) = 1$  and  $s \neq 1$ , then  $t \leq t \oplus s$  or  $\bar{t} \leq t \oplus s$ .
4. If  $\alpha(s) = 0$  and  $s \neq 0$ , then  $t \leq t \otimes s$  or  $\bar{t} \leq t \otimes s$ .

**Proof:** If  $\alpha(s) = 0$ , then 0 is a prefix of  $s$ , i.e.,  $0 \leq s$ . Because  $\oplus$  is monotonic with respect to the prefix order, we have  $t = t \oplus 0 \leq t \oplus s$ . This proves the first claim, and the second claim follows similarly.

For the third claim, assume that  $\alpha(s) = 1$  and  $s \neq 1$ . Then  $s$  must have at least two letters, and begins with 10, i.e.,  $10 \leq s$ . By monotonicity of  $\oplus$ , we have  $t \oplus 10 \leq t \oplus s$ , and also  $\bar{t} \oplus 10 \leq \bar{t} \oplus s$ . By Lemma 4.3, either  $t \leq t \oplus 10$  or  $\bar{t} \leq t \oplus 10$ . Thus the claim holds. The last claim follows similarly.  $\square$

The next result shows that the length of a word  $t$  cannot be decreased by adding another word  $s \neq 1$ , or by multiplying it by another word  $s \neq 0$ .

**Corollary 4.5.** *Suppose that  $t, s \in T$ .*

1. If  $s \neq 1$ , then  $l(t) \leq l(t \oplus s)$ .
2. If  $s \neq 0$ , then  $l(t) \leq l(t \otimes s)$ .

**Proof:** If  $s \neq 1$ , then either  $t \leq t \oplus s$  or  $\bar{t} \leq t \oplus s$ , by Lemma 4.4. Hence,  $l(t) \leq l(t \oplus s)$ . The second claim follows by duality.  $\square$

For any word  $w$ , let  $w^{-1}$  denote the mirror image of  $w$ . It is clear that  $w \in T$  iff  $w^{-1} \in T$ . Moreover,

$$\begin{aligned} (t \oplus s)^{-1} &= t^{-1} \oplus s^{-1} \\ (t \otimes s)^{-1} &= t^{-1} \otimes s^{-1} \\ \bar{t}^{-1} &= (\bar{t})^{-1}, \end{aligned}$$

for all  $t \in T$ . Since also  $(t^{-1})^{-1} = t$ , we have:

**Proposition 4.6.** *The function  $w \mapsto w^{-1}$ ,  $w \in T$ , defines an automorphism of  $C$ , i.e., a one-to-one and onto mapping  $T \rightarrow T$  which preserves the operations  $\oplus, \otimes, \bar{\phantom{x}}$ , and constants 0 and 1.*

The suffix order on  $T$  is represented by the inequalities below together with the reflexive and transitive laws:

$$\begin{aligned} 0 &\leq 10 \leq 010 \leq 1010 \leq 01010 \leq \dots, \\ 1 &\leq 01 \leq 101 \leq 0101 \leq 10101 \leq \dots. \end{aligned}$$

It follows from Proposition 4.6 that the operations  $\oplus$  and  $\otimes$  also preserve the suffix order. Some additional properties of the algebra  $C$  are given in the appendix.

### 5. Counting changes to a threshold

Since the underlying set  $T$  of Algebra  $C$  is infinite, an arbitrary number of changes can be counted. An alternative is to count only up to some threshold  $k - 1$ ,  $k \geq 1$ , and consider all transients with length  $k$  or more as equivalent.

Recall that a congruence relation of an algebra is an equivalence relation on the underlying set of the algebra preserved by the operations. Since the  $\oplus$  operation in Algebra  $C$  is commutative, and by de Morgan's law L6, it follows that an equivalence relation  $\sim$  on  $T$  is a congruence relation of  $C$  if for all transients  $t, s, w \in T$  with  $t \sim s$  we have that  $(w \oplus t) \sim (w \oplus s)$  and  $\bar{t} \sim \bar{s}$ . When  $\sim$  is a congruence relation of  $C$ , there is a unique algebra  $C/\sim = (T/\sim, \oplus, \otimes, \bar{\phantom{x}}, 0, 1)$ , defined on the quotient set  $T/\sim$  of equivalence classes of  $T$  with respect to  $\sim$ , such that the map taking a transient to its equivalence class is a homomorphism, i.e., such that it preserves the operations and constants. For any equivalence classes  $[s]_{\sim}$  and  $[t]_{\sim}$  containing the transients  $s$  and  $t$ , respectively, we have in the quotient algebra  $C/\sim$  that  $[s]_{\sim} \oplus [t]_{\sim} = [s \oplus t]_{\sim}$ ,  $[s]_{\sim} \otimes [t]_{\sim} = [s \otimes t]_{\sim}$  and  $\overline{[t]_{\sim}} = [\bar{t}]_{\sim}$ . Moreover, the constants  $0, 1$  in  $C/\sim$  are the congruence classes  $[0]_{\sim}$  and  $[1]_{\sim}$ . The fact that the operations are well-defined is a consequence of the congruence property of  $\sim$ . It is known (see, e.g. [9, 17]) that  $C/\sim$  satisfies any equation that holds in  $C$ . Thus, when  $\sim$  is a congruence relation,  $C/\sim$  is a commutative De Morgan bisemigroup.

Suppose that  $k \geq 1$ . Relation  $\sim_k$  is defined on the set  $T$  of transients as follows: For  $t, s \in T$ ,  $t \sim_k s$  if either  $t = s$  or  $t$  and  $s$  are both of length  $\geq k$ .

**Proposition 5.1.** *Relation  $\sim_k$  is a congruence relation on  $C$ .*

**Proof:** Relation  $\sim_k$  is clearly an equivalence relation. Suppose now that  $t \sim_k s$ . It is clear that  $\bar{t} \sim_k \bar{s}$ . We argue by induction on the length of  $t$  to show that  $(w \oplus t) \sim_k (w \oplus s)$ , for all transients  $w$ . When the length of  $t$  is less than  $k$  we have  $t = s$ , and our claim is obvious. When  $t$  is of length  $\geq k$  then, by Corollary 4.5,  $w \oplus t$  and  $w \oplus s$  are both of length  $\geq k$ , so that  $(w \oplus t) \sim_k (w \oplus s)$ .  $\square$

The equivalence classes of the quotient algebra  $C_k = C/\sim_k$  are of two types. Each transient  $t$  with  $l(t) < k$  is in a class by itself, and all the words of length  $\geq k$  constitute a class, which is denoted by  $\Phi$ . We denote by  $T_k$  this set of equivalence classes, and we denote by  $t$  the equivalence class consisting of the singleton  $t$ . Thus  $T_k = \{t \mid l(t) < k\} \cup \{\Phi\}$ . The operations on equivalence classes are as follows. The complement of the class containing  $t$  is the class containing  $\bar{t}$ . The sum (product) of the class containing  $t$  and the class containing  $t'$  is the class containing  $t \oplus t'$  ( $t \otimes t'$ ). Thus, the quotient algebra  $C_k$  is a commutative de Morgan bisemigroup with  $2k - 1$  elements.

*Example 2.* For  $k = 2$ , the  $\oplus$  and  $\otimes$  operations are shown in Table 2. These are the operations of the well known 3-element ternary algebra [8]. In addition to laws L1–L6, and their duals, ternary algebra also satisfies the laws of Table 3. Ternary algebras are closely related to Boolean algebras. Ternary algebras obey L1–L9, and their duals, which all hold

Table 2. OR and AND operations for  $k = 2$ .

$\oplus$	0	$\Phi$	1
0	0	$\Phi$	1
$\Phi$	$\Phi$	$\Phi$	1
1	1	1	1

$\otimes$	0	$\Phi$	1
0	0	0	0
$\Phi$	0	$\Phi$	$\Phi$
1	0	$\Phi$	1

Table 3. Ternary laws.

---

L7	$x \oplus x = x$	L7'	$x \otimes x = x$
L8	$x \oplus (x \otimes y) = x$	L8'	$x \otimes (x \oplus y) = x$
L9	$x \oplus (y \otimes z) = (x \oplus y) \otimes (x \oplus z)$	L9'	$x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$
L10	$\bar{\Phi} = \Phi$		
L11	$(x \oplus \bar{x}) \oplus \Phi = x \oplus \bar{x}$	L11'	$(x \otimes \bar{x}) \otimes \Phi = x \otimes \bar{x}$
L12	$(x \oplus \bar{x}) \oplus (y \otimes \bar{y}) = x \oplus \bar{x}$	L12'	$(x \otimes \bar{x}) \otimes (y \oplus \bar{y}) = x \otimes \bar{x}$

---

Table 4. OR and AND operations for  $k = 3$ .

$\oplus$	0	01	$\Phi$	10	1
0	0	01	$\Phi$	10	1
01	01	01	$\Phi$	$\Phi$	1
$\Phi$	$\Phi$	$\Phi$	$\Phi$	$\Phi$	1
10	10	$\Phi$	$\Phi$	10	1
1	1	1	1	1	1

$\otimes$	0	01	$\Phi$	10	1
0	0	0	0	0	0
01	0	01	$\Phi$	$\Phi$	01
$\Phi$	0	$\Phi$	$\Phi$	$\Phi$	$\Phi$
10	0	$\Phi$	$\Phi$	10	10
1	0	01	$\Phi$	10	1

in Boolean algebras. Notably absent from ternary algebras are the laws for complements:

$$x \oplus \bar{x} = 1, \quad x \otimes \bar{x} = 0.$$

Laws L10–L12 and their duals do not hold in Boolean algebras.

*Example 3.* The OR and AND operations for the case  $k = 3$  are shown in Table 4. This is the quinary algebra introduced in 1972 by Lewis [19], and studied also in [4, 10, 18]. Note that both binary operations are idempotent, i.e., laws L7 and L7' hold; hence, we have a bisemilattice [4].

*Example 4.* For  $k = 4$ , the OR and AND operations are shown in Table 5. In this case, the binary operations are no longer idempotent.

In the following proposition we examine which of the ternary laws hold in algebras  $C_k$ , for  $k \geq 3$ .

Table 5. OR and AND operations for  $k = 4$ .

$\oplus$	0	01	010	$\Phi$	101	10	1
0	0	01	010	$\Phi$	101	10	1
01	01	01	$\Phi$	$\Phi$	101	101	1
010	010	$\Phi$	$\Phi$	$\Phi$	$\Phi$	$\Phi$	1
$\Phi$	$\Phi$	$\Phi$	$\Phi$	$\Phi$	$\Phi$	$\Phi$	1
101	101	101	$\Phi$	$\Phi$	101	101	1
10	10	101	$\Phi$	$\Phi$	101	10	1
1	1	1	1	1	1	1	1

$\otimes$	0	01	010	$\Phi$	101	10	1
0	0	0	0	0	0	0	0
01	0	01	010	$\Phi$	$\Phi$	010	01
010	0	010	010	$\Phi$	$\Phi$	010	010
$\Phi$	0	$\Phi$	$\Phi$	$\Phi$	$\Phi$	$\Phi$	$\Phi$
101	0	$\Phi$	$\Phi$	$\Phi$	$\Phi$	$\Phi$	101
10	0	010	010	$\Phi$	$\Phi$	10	10
1	0	01	010	$\Phi$	101	10	1

**Proposition 5.2.** *The following results apply to Algebras  $C_k$  for  $k \geq 3$ :*

- $C_3$  satisfies L7 and L7', but  $C_k$ , with  $k \geq 4$  does not.
- $C_k$  with  $k \geq 3$  does not satisfy L8, L9, L8', and L9'.
- For  $k \geq 3$ ,  $C_k$  satisfies L10.
- $C_3$  satisfies L11, L12, L11', and L12', but  $C_k$  does not satisfy these laws for  $k \geq 4$ .

**Proof:** We show the arguments only for the unprimed laws; the primed laws follow by duality.

- For  $C_3$ , the operation tables show that L7, L7' hold. For  $k \geq 4$ , we have  $k - 2 \geq 2$ . If  $k$  is even, then  $(k - 2)/2$  is an integer  $\geq 1$ . Let  $t = (01)^{(k-2)/2}0$ . Then  $l(t) = k - 1$ , and  $t$  by itself constitutes an equivalence class of  $\sim_k$ . By the definition of  $\oplus$  in Algebra  $C$ ,  $t \oplus t = (01)^{k-2}0$ . Thus  $l(t \oplus t) = 2k - 3 = k + (k - 3) \geq k$ , and  $t \oplus t$  is in the equivalence class  $\Phi$ , showing that  $t \neq t \oplus t$ . If  $k$  is odd, then  $k \geq 5$ . Let  $t = (01)^{(k-3)/2}0$ ; then  $l(t) = k - 2 < k$ . However,  $t \oplus t = (01)^{k-3}0$ , and  $l(t \oplus t) = 2k - 5 = k + (k - 5) \geq k$ , showing again that  $t \neq t \oplus t$ .
- Let  $x = 01$  and  $y = 10$ . In algebra  $C$ , we have  $x \oplus (x \otimes y) = 01 \oplus 010 = 0101$ . For each  $k \geq 3$ ,  $x$  and  $x \oplus (x \otimes y)$  are in different equivalence classes. Hence L8 does not hold.  
 For L9, use  $x = 01$ ,  $y = 10$ , and  $z = 0$ . Then in Algebra  $C$ ,  $x \oplus (y \otimes z) = 01 \oplus 0 = 01$ , while  $(x \oplus y) \otimes (x \oplus z) = 010 \otimes 01 = 010$ . Again, the two results are in different equivalence classes of  $\sim_k$  for all  $k \geq 3$ .
- L10 holds, because  $\sim_k$  is a congruence, and the complement of a word of length  $\geq k$  has length  $\geq k$ .
- For  $k = 3$ , L11 and L12 are easily verified from the operation tables.  
 For  $k \geq 4$ ,  $01 \oplus \overline{01} = 01 \oplus 10 = 101$ , but  $(01 \oplus \overline{01}) \oplus \Phi = 101 \oplus \Phi = \Phi \neq 101$ . Thus L11 fails for  $k \geq 4$ .  
 For L12,  $(01 \oplus \overline{01}) \oplus (01 \otimes \overline{01}) = 101 \oplus 010 = 10101$  and  $01 \oplus \overline{01} = 101$ . For  $k \geq 4$ , 101 and 10101 are in different equivalence classes of  $\sim_k$ . Hence L12 is not satisfied.

□

We now discuss some further properties of Algebras  $C_k$ . Roughly speaking, the next lemma shows that, if two words  $s$  and  $t$  are congruent with respect to a congruence  $\theta$  on  $C$  and end in the same letter, then we can always find longer words  $s'$  and  $t'$  which are also congruent with respect to  $\theta$ .

**Lemma 5.3.** *Suppose that  $t, s \in T$  with  $t \neq s$  and  $\omega(t) = \omega(s)$ . If  $\theta$  is a congruence relation on  $C$  with  $t\theta s$ , then for each  $m \geq 0$  there exist  $t', s' \in T$  with  $t' \neq s'$ ,  $\omega(t') = \omega(s')$ ,  $t'\theta s'$ ,  $l(t') = l(t) + m$  and  $l(s') = l(s) + m$ .*

**Proof:** It is sufficient to prove the claim for  $m = 1$ . If  $t$  and  $s$  both end in 0, then by Lemma 4.3  $t \oplus 01 = t1$  and  $s \oplus 01 = s1$ . Since  $\theta$  is a congruence,  $(t \oplus 01)\theta(s \oplus 01)$ . Thus,  $(t1)\theta(s1)$ . If  $t$  and  $s$  both end in 1, then take  $t' = t0 = t \otimes 10$  and  $s' = s0 = s \otimes 10$ .  $\square$

**Proposition 5.4.** *Suppose that  $k \geq 2$ . Then  $\sim_{k-1}$  is the smallest congruence relation of  $C$  strictly containing  $\sim_k$ .*

**Proof:** Suppose that  $\theta$  is a congruence relation of  $C$  strictly containing  $\sim_k$ . We first show that there is a transient  $u$  of length  $k - 1$  which is congruent modulo  $\theta$  to some transient  $v$  of length  $\geq k$ .

If  $\theta$  strictly contains  $\sim_k$ , then there exist distinct transients  $t, s \in T$  with  $t\theta s$ ,  $l(t) \leq l(s)$  and  $l(t) < k$ . Suppose first that  $\omega(s) \neq \omega(t)$ . If  $\omega(s) = 0$  and  $\omega(t) = 1$ , then  $s \oplus 01 = s1$  and  $t \oplus 01 = t$ , by Lemma 4.3(3). Thus  $t\theta s1$ . If  $\omega(s) = 1$  and  $\omega(t) = 0$ , then  $s \otimes 10 = s0$  and  $t \otimes 10 = t$ , by Lemma 4.3(4). Thus  $t\theta s0$ . In either case, we have two transients  $t$  and  $s'$ , where  $s'$  is  $s0$  or  $s1$ , congruent with respect to  $\theta$ , and such that  $l(t) < l(s')$ . If  $l(t) = k - 1$ , we are done: let  $u = t$  and  $v = s'$ . Otherwise, note that  $\omega(t) = \omega(s')$ . Consider the transients  $t\bar{b}$  and  $s'\bar{b}$ , where  $b$  denotes the last letter of  $t$ . By Lemma 5.3 we have that  $t\bar{b}\theta s'\bar{b}$ . Continuing in the same way, by repeated applications of Lemma 5.3 we can construct transients  $u$  and  $v$  with  $u\theta v$ ,  $l(u) = k - 1$  and  $l(v) > k$ .

In case  $\omega(s) = \omega(t)$ , consider  $t\bar{b}$  and  $s\bar{b}$ , where  $b = \omega(t)$ , and apply Lemma 5.3, as above.

Note that there are exactly two transients of any given length, one beginning with 0 and the other with 1. Having found  $u$  and  $v$  as above, we claim that all transients of length  $\geq k - 1$  are congruent with respect to  $\theta$ . Indeed, since  $\theta$  contains  $\sim_k$ , we have that  $u\theta w$  for all transients  $w$  of length  $\geq k$ . Thus, using the congruence property for  $\bar{\phantom{x}}$ , also  $\bar{u}\theta w$  for all transients  $w$  with length  $\geq k$ . Moreover, by transitivity, also  $u\theta\bar{u}$ . Thus,  $\theta$  contains  $\sim_{k-1}$ .  $\square$

**Corollary 5.5.** *The lattice of congruences of each  $C_k$  is isomorphic to a chain of length  $k$ .*

By Corollary 5.5, we immediately have:

**Corollary 5.6.** *Each  $C_k$  with  $k \geq 2$  is subdirectly irreducible.*

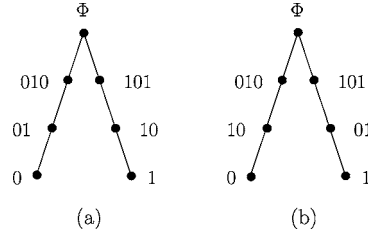


Figure 5. Partial orders on  $T_4$ : (a)  $\leq$ ; (b)  $\preceq$ .

The last result means that  $C_k$  is not a subalgebra of a direct product of algebras with fewer elements. Informally, this means that  $C_k$  cannot be constructed from simpler algebras. This implies, for example, that  $C_k$  cannot be expressed in any nontrivial way as an algebra of ordered triples [18], where each component of the triple is evaluated in its own algebra.

As we did for  $T$ , we define two partial orders on  $T_k$ ; these are derived from the prefix and suffix orders. The class consisting of  $t$  is  $\leq$  the class consisting of  $t'$  if and only if  $t$  is a prefix of  $t'$ , and every class is  $\leq \Phi$ . The second partial order is similarly defined using suffixes. For example, the Hasse diagrams of the two orders on  $T_4$  are shown in figure 5.

As was the case in  $C$ , the three operations  $\oplus$ ,  $\otimes$ , and  $-$  in  $C_k$  are monotonic with respect to both partial orders. This property plays an important role in the simulation algorithms described in the next section.

**6. Circuit simulations**

For a logic circuit with  $n$  gates, the binary analysis described in Section 2 may have as many as  $2^n$  states. If one can be satisfied with partial information about the circuit behavior, then simulation in a change-counting algebra is often an efficient method for finding that information.

For the detection of hazards, ternary algebra has been used since 1948 [16]. A two-pass ternary simulation method was introduced by Eichelberger in 1965 [12], and later studied by many authors (see [8] for further details). The following is an adaptation of these algorithms to change-counting algebra.

We denote vectors by unsubscripted letters and their components by subscripted letters. Let  $N$  be a circuit with  $X = (X_1, \dots, X_m)$  as the vector of input variables, and  $s = (s_1, \dots, s_n)$  as the vector of state variables. Each state variable has a Boolean excitation function  $S_j : \{0, 1\}^m \times \{0, 1\}^n \mapsto \{0, 1\}$ , i.e., the vector  $S(x, y) = (S_1(x, y), \dots, S_n(x, y))$  is the vector of excitations of the circuit.

For example, the circuit of figure 1 has input vector  $X = (X_1)$ , state vector  $s = (s_1, \dots, s_4)$ , and excitation functions given by the following Boolean expressions:

$$S_1 = \bar{X}, S_2 = 1 \wedge X, S_3 = X \wedge s_1, S_4 = s_2 \vee s_3,$$

where we have identified  $X$  with  $X_1$ , since  $X$  has only one component.

Suppose initially the input  $X$  has the value  $X = \hat{a} = (\hat{a}_1, \dots, \hat{a}_m)$ , and the state has the value  $s = b = (b_1, \dots, b_n)$ , where all the  $a_i$  and  $b_j$  are in the set  $\{0, 1\}$ . We assume that the circuit is initially stable, i.e., that  $S(\hat{a}, b) = b$ , and the input changes to  $a = (a_1, \dots, a_m)$ .

We define an operation  $\circ$  as follows. For  $a, b \in \{0, 1\}$ , if  $a = b$ , then  $a \circ b = a$ . For  $a \neq b$ , if the simulation is done in algebra  $C_2$ , which is ternary algebra, then  $a \circ b = b \circ a = \Phi$ . Otherwise, if  $a \neq b$ , and the simulation uses algebra  $C$  or algebra  $C_k$  with  $k > 2$ , then  $a \circ b = ab$ , where  $ab$  represents the concatenation of  $a$  and  $b$ . This notation is extended to vectors. If  $\hat{a} = (\hat{a}_1, \dots, \hat{a}_m)$  and  $a = (a_1, \dots, a_m)$

$$\hat{a} \circ a = (\hat{a}_1 \circ a_1, \dots, \hat{a}_m \circ a_m).$$

For example,  $(1, 0, 0, 1) \circ (1, 1, 0, 0) = (1, \Phi, 0, \Phi)$  in  $C_2$ ; otherwise,  $(1, 0, 0, 1) \circ (1, 1, 0, 0) = (1, 01, 0, 10)$ . In case of  $C_2$ ,  $\hat{a} \circ a$  indicates by a  $\Phi$  all those variables that change in going from  $\hat{a}$  to  $a$ . In the other cases,  $\hat{a} \circ a$  specifies the change as being either from 0 to 1 or from 1 to 0. Such detail is not possible in the case of  $C_2$ , since only one value  $\Phi$  is available for denoting a value that is neither 0 nor 1.

For the simulation algorithms we use the extensions of Boolean functions to transients. This topic is discussed in Section 7; for now we use circuits constructed with OR gates, AND gates, and inverters, for which the extensions are  $\oplus$ ,  $\otimes$ , and complement in the appropriate algebra. Variables and their values in a change-counting algebra are denoted in boldface. For example, for the circuit of figure 1, the excitation equations become

$$\mathbf{S}_1 = \bar{\mathbf{X}}, \mathbf{S}_2 = 1 \otimes \mathbf{X}, \mathbf{S}_3 = \mathbf{X} \otimes \mathbf{s}_1, \mathbf{S}_4 = \mathbf{s}_2 \oplus \mathbf{s}_3.$$

Our simulation consists of two parts, called Algorithms A and B. Algorithm A starts with the circuit in the stable (binary) initial state  $(\hat{a}, b)$ . The input is then set to  $\mathbf{a} = \hat{a} \circ a$ , and is kept constant at that value for the duration of the algorithm. After the input change, some state variables become unstable. We change all unstable variables at the same time to their excitations. This can be viewed as the “unit-delay” model, in which all gates have the same (unit) delay. We obtain a new internal state (a vector of transients from the set  $T$  or  $T_k$  of the change-counting algebra used), and the process is then repeated. Formally, Algorithm A is specified below.

**Algorithm A**

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

Recall that  $\leq$  is the prefix relation on transients. We extend this notion to vectors of transients. Thus, if  $\mathbf{t} = (t_1, \dots, t_m)$  and  $\mathbf{t}' = (t'_1, \dots, t'_m)$ , then  $\mathbf{t} \leq \mathbf{t}'$  iff  $t_i \leq t'_i$  for



$i = 1, \dots, m$ . Note that  $\hat{a} \leq \hat{a} \circ a = \mathbf{a}$ . By the stability of the initial state we have  $b = S(\hat{a}, b)$ . Since the transient operations  $\oplus, \otimes$ , and complement agree with their Boolean counterparts on binary values, we have  $S(\hat{a}, b) = \mathbf{S}(\hat{a}, b)$ . Hence,  $\mathbf{s}^0 = b = S(\hat{a}, b) = \mathbf{S}(\hat{a}, b) \leq \mathbf{S}(\mathbf{a}, b) = \mathbf{S}(\mathbf{a}, \mathbf{s}^0) = \mathbf{s}^1$ , where the inequality follows by Proposition 4.2, which states that  $\oplus, \otimes$ , and complement are monotonic with respect to the prefix order. It then follows by induction on  $h$  that Algorithm A results in a nondecreasing state sequence:

$$\mathbf{s}^0 \leq \mathbf{s}^1 \leq \dots \leq \mathbf{s}^h \leq \dots$$

Algorithm A may not terminate in Algebra  $C$ , but it must terminate in every algebra  $C_k$ ,  $k \geq 2$ . Let the result of Algorithm A be state  $\mathbf{s}^A$ , if the algorithm terminates. Note that  $\mathbf{s}^A = \mathbf{S}(\mathbf{a}, \mathbf{s}^A)$ , i.e., that the circuit is again in a stable state at the end of Algorithm A.

Example 5 below illustrates Algorithm A for  $C_2$ , and further examples follow.

The second part of the simulation consists of Algorithm B, which is applicable when Algorithm A terminates. Let the result of Algorithm A be state  $\mathbf{s}^A$ . Algorithm B starts with the circuit in state  $\mathbf{s}^A$ , and input  $\mathbf{a}$ , and the input is changed to  $a$ . Algorithm B is defined below.

**Algorithm B**

```

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

```

Recall now that  $\preceq$  is the suffix order on transients, and that  $\oplus, \otimes$ , and complement are monotonic with respect to the suffix order, as a consequence of Proposition 4.6. It is easy to verify that Algorithm B results in a nonincreasing sequence of states:

$$\mathbf{s}^A = \mathbf{t}^0 \succeq \mathbf{t}^1 \succeq \dots \succeq \mathbf{t}^B.$$

Again we reach a stable state, since  $\mathbf{t}^B = \mathbf{S}(a, \mathbf{t}^B)$ .

Both algorithms compute fixed points.

**Proposition 6.1.** *Suppose that Algorithm A terminates with state  $\mathbf{s}^A$ . Then  $\mathbf{s}^A$  is the least fixed point of the function  $\mathbf{S}(\mathbf{a}, \mathbf{x})$  over  $b$ , i.e.,*

$$\mathbf{S}(\mathbf{a}, \mathbf{s}^A) = \mathbf{s}^A, \quad \text{and} \quad (1)$$

$$b \leq \mathbf{s} \ \& \ \mathbf{S}(\mathbf{a}, \mathbf{s}) = \mathbf{s} \Rightarrow \mathbf{s}^A \leq \mathbf{s}. \quad (2)$$

**Proof:** Recall that Algorithm A terminates if and only if a state vector  $\mathbf{s}^h$  is reached with  $\mathbf{s}^h = \mathbf{S}(\mathbf{a}, \mathbf{s}^{h-1})$ , and then  $\mathbf{s}^A = \mathbf{s}^h = \mathbf{S}(\mathbf{a}, \mathbf{s}^{h-1}) = \mathbf{S}(\mathbf{a}, \mathbf{s}^A)$ . This proves (1).

Suppose now that the premisses of (2) hold for a state vector  $\mathbf{s}$ . We prove by induction on  $h$  using the monotonicity of the  $\oplus, \otimes$  and complement operations with respect to the prefix

order that  $\mathbf{s}^h \leq \mathbf{s}$ . The basis case  $h = 0$  is obvious, since  $\mathbf{s}^0 = b$  and  $b \leq \mathbf{s}$ , by assumption. Assuming the claim for  $h - 1$ , where  $h > 0$ , we obtain  $\mathbf{s}^h = \mathbf{S}(\mathbf{a}, \mathbf{s}^{h-1}) \leq \mathbf{S}(\mathbf{a}, \mathbf{s}) = \mathbf{s}$ , by the induction hypothesis.  $\square$

As noted above, Algorithm A always terminates in any Algebra  $C_k$ . Moreover, by Proposition 6.1, it computes the least fixed point of the function  $\mathbf{S}(\mathbf{a}, \mathbf{x})$  over  $b$  with respect to the prefix order.

In Algebra  $C$ , we have:

**Proposition 6.2.** *Algorithm A terminates in Algebra  $C$  if and only if there is a fixed point of the function  $\mathbf{S}(\mathbf{a}, \mathbf{x})$  over  $b$ .*

**Proof:** One direction follows from Proposition 6.1. Indeed, if the algorithm terminates, then it computes a fixed point of  $\mathbf{S}(\mathbf{a}, \mathbf{x})$  over  $b$ . For the opposite direction, suppose that  $\mathbf{s}$  is a fixed point of  $\mathbf{S}(\mathbf{a}, \mathbf{x})$  over  $b$ . As above, we have that  $\mathbf{s}^h \leq \mathbf{s}$ , for each  $h$ . Since there are only a finite number of vectors  $\leq \mathbf{s}$ , the nondecreasing sequence  $\mathbf{s}^0 \leq \mathbf{s}^1 \leq \dots$  must be eventually constant.  $\square$

In Section 8 we prove that Algorithm A terminates in Algebra  $C$  if the circuit is feedback-free.

Algorithm B always terminates, independently of the algebra used. The proof of the following fact is similar to that of Proposition 6.1, and is therefore omitted.

**Proposition 6.3.** *Algorithm B computes the greatest fixed point of function  $\mathbf{S}(\mathbf{a}, \mathbf{x})$  below  $\mathbf{s}^A$  with respect to the suffix order.*

*Example 5.* Consider the circuit shown in figure 1. Refer now to Table 6. The values of the input variable  $X$  and the state variables  $s_1, \dots, s_4$  are shown in rows as the simulation progresses. We begin in the initial state 01000, which is stable. We wish to study the behavior of the circuit when the input changes from 0 to 1 and is kept constant at 1.

In Algorithm A, the input changes to the uncertain or unknown value  $\Phi$ . Instead of Boolean functions, we now use the ternary extensions of these functions as they are defined in Algebra  $C_2$ . After  $X$  changes to  $\Phi$ , Gates 1, 2, and 3, become unstable in the ternary

Table 6. Ternary simulation.

	$X$	$s_1$	$s_2$	$s_3$	$s_4$
Initial state	0	1	0	0	0
	$\Phi$	1	0	0	0
	$\Phi$	$\Phi$	$\Phi$	$\Phi$	0
Result A	$\Phi$	$\Phi$	$\Phi$	$\Phi$	$\Phi$
	1	$\Phi$	$\Phi$	$\Phi$	$\Phi$
	1	0	1	$\Phi$	$\Phi$
Result B	1	0	1	0	1

model. After all unstable variables are changed, we obtain the second row of Table 6. Now Gate 4 becomes unstable and changes, to yield the third row. Since this state is stable, Algorithm A terminates here.

In the case of  $C_2$  the two partial orders  $\leq$  and  $\preceq$  coincide and are given by

$$0 \leq \Phi, 1 \leq \Phi, 0 \leq 0, 1 \leq 1, \Phi \leq \Phi.$$

This partial order is known as the *uncertainty* partial order [8].

In our example, the input became more uncertain by changing from 0 to  $\Phi$ . Since the gate operations preserve this order, the gate outputs can only become more uncertain. Thus the sequence of states produced by Algorithm A is nondecreasing, and the process terminates in at most  $n$  steps if there are  $n$  gates in the circuit. Intuitively, in Algorithm A we introduce uncertainty in the circuit inputs and we see how this uncertainty spreads throughout the circuit.

In Algorithm B we start in the state produced by Algorithm A, but now we change the inputs to their final values from  $\Phi$ , thus reducing uncertainty. In our example,  $X$  becomes 1. We again use the ternary excitation functions to see whether the uncertainty will be removed from any gates. This time Algorithm B produces a nonincreasing sequence of states, which must terminate in no more than  $n$  steps. The final result is the vector 10101, showing that each gate reaches a binary value after the transient is over. Of course, this is the answer we expect since the circuit is feedback-free.

It is clear from the circuit diagram that  $s_1$  changes from 1 to 0 and  $s_2$  changes from 0 to 1 without any hazards. As we have shown in Section 2, there is a static hazard in  $s_3$ , and a dynamic hazard in  $s_4$ .

This example illustrates that ternary simulation is capable of detecting static hazards. Gate  $s_3$  is 0 at the beginning and 0 at the end, but it is  $\Phi$  at the end of Algorithm A, and this indicates a static hazard [8]. Our example also clearly shows that ternary simulation is not capable of detecting dynamic hazards. Gates  $s_2$  and  $s_4$  both change from 0 to  $\Phi$  to 1, yet  $s_2$  has no dynamic hazard, while  $s_4$  has one.

In the examples that follow, we show how the accuracy of the simulation improves when we use Algebra  $C_k$  as  $k$  increases.

*Example 6.* In Table 7 we repeat the simulation, this time using quinary extensions of gates as defined in Table 4. Instead of changing  $X$  to  $\Phi$ , we change it to 01 in Algorithm A, since this is the change we wish to study. From the result of Algorithm A we see that  $s_1$  changes from 1 to 0 and  $s_2$  changes from 0 to 1, both without hazards. The static hazard in  $s_3$  is detected as before, as is the dynamic hazard in  $s_4$ . This example shows that quinary simulation is capable of detecting both static and dynamic hazards.

*Example 7.* We repeat the simulation now using Algebra  $C_4$  with seven values. Refer to Table 8. This time Algorithm A not only reveals that there is a static hazard in  $s_3$ , but also identifies it as 010. Note, however, that the dynamic hazard is still not identified.

*Example 8.* We repeat the simulation using Algebra  $C_5$  with nine values. Refer to Table 9. This time Algorithm A identifies the dynamic hazard as 0101.

Table 7. Quinary simulation.

	$X$	$s_1$	$s_2$	$s_3$	$s_4$
Initial state	0	1	0	0	0
	01	1	0	0	0
	01	10	01	01	0
	01	10	01	$\Phi$	01
Result A	01	10	01	$\Phi$	$\Phi$
	1	10	01	$\Phi$	$\Phi$
	1	0	1	10	$\Phi$
Result B	1	0	1	0	1

Table 8. Septenary simulation.

	$X$	$s_1$	$s_2$	$s_3$	$s_4$
Initial state	0	1	0	0	0
	01	1	0	0	0
	01	10	01	01	0
	01	10	01	010	01
Result A	01	10	01	010	$\Phi$
	1	10	01	010	$\Phi$
	1	0	1	10	$\Phi$
Result B	1	0	1	0	1

Table 9. Nonary simulation.

	$X$	$s_1$	$s_2$	$s_3$	$s_4$
Initial state	0	1	0	0	0
	01	1	0	0	0
	01	10	01	01	0
	01	10	01	010	01
Result A	01	10	01	010	0101
	1	10	01	010	0101
	1	0	1	10	0101
Result B	1	0	1	0	1

Table 10. Simulation of circuit with transient oscillations.

	$X$	$s_1$	$s_2$	$s_3$
Initial state	0	0	1	1
	01	0	1	1
	01	01	1	10
	01	01	10	101
	01	01	10	10101
	...	...	...	...

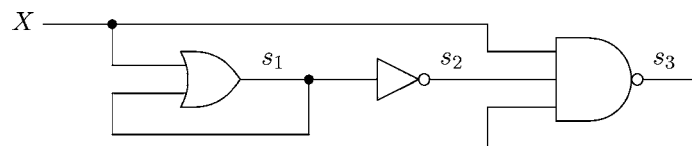


Figure 6. Circuit with transient oscillations.

Observe that the same table results if we simulate our circuit in Algebra  $C_k$  with any  $k \geq 5$  or in Algebra  $C$ . Note also, that for  $k \geq 4$  Algorithm B is no longer necessary for this circuit, since the entire history of worst-case signal changes is recorded on each gate output.

*Example 9.* The simulation of the circuit of figure 6 is shown in Table 10. Here, if we use Algebra  $C$ , Algorithm A does not terminate. Binary analysis of this circuit shows that there are transient oscillations during this input change. After the input changes to 1, the NAND gate can oscillate while the OR gate is unstable. For further details see [8]. The OR gate must eventually change, since it has a finite delay. However, it is not possible to bound the number of oscillations of the NAND gate without the knowledge of the relative sizes of the gate delays. Hence the result obtained from the simulation is correct. If one uses Algebra  $C_5$ , for example, Algorithm A does terminate in four steps after the input change, and Algorithm B predicts state 101 as the final outcome of the input change.

*Example 10.* Consider the circuit of figure 7. As in Example 9, Algorithm A using Algebra  $C$  does not terminate, this time because there is a nontransient oscillation in the OR gate and the wire delay in the feedback loop. For more details see [8]. Simulation in Algebra  $C_k$  does, of course, terminate. In Table 11 we show the simulation with  $C_4$ . The nontransient

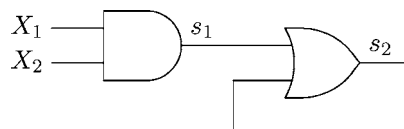


Figure 7. Circuit with nontransient oscillation.

Table 11. Simulation of circuit with nontransient oscillation.

	$X_1$	$X_2$	$s_1$	$s_2$
Initial state	1	0	0	0
	10	01	0	0
	10	01	010	0
	10	01	010	010
Result A	10	01	010	$\Phi$
	0	1	010	$\Phi$
Result B	0	1	0	$\Phi$

Table 12. Simulation of circuit with unknown initial values.

	$X_1$	$X_2$	$s_1$	$s_2$
Initial state	1	$\Phi$	$\Phi$	$\Phi$
Result A	$\Phi$	$\Phi$	$\Phi$	$\Phi$
	0	$\Phi$	$\Phi$	$\Phi$
Result B	0	$\Phi$	0	$\Phi$

oscillation is detected by the presence of  $\Phi$  in the result of Algorithm B. This example also shows that multiple input changes can be handled by our simulation.

*Example 11.* Consider the circuit of figure 7, again. Suppose that  $X_1 = 1$ , but input  $X_2$  and the initial state of the circuit are initially unknown. This is accounted for by setting  $X_2 = s_1 = s_2 = \Phi$ . The initial state is now stable in Algebra  $C_2$ . Suppose now input  $X_1$  changes to 0. No information is obtained from Algorithm 1, since the gate variables are in a state of maximal uncertainty. Algorithm B, does however provide the information that  $s_1$  becomes 0, as is shown in Table 12.

The detailed discussion of the relation between the simulations described above and binary analysis of Section 2 is beyond the scope of the present paper. A complete characterization of ternary simulation is given in [8]. For simulation in  $C$ , it is proved in [7, 14] that the results of Algorithm A cover the results of binary analysis, in the sense that every transient predicted by binary analysis is a prefix of the corresponding transient predicted by the simulation. In [14, 15] it is shown that the result of Algorithm A for a feedback-free circuit  $N$  in Algebra  $C$  agrees with the result of binary analysis of a circuit  $\tilde{N}$ , which is obtained from  $N$  by adding a sufficient number of wire delays.

## 7. Extensions of Boolean functions

In this section we show that simulation in Algebra  $C$  or  $C_k$  is not limited to 2-input OR and AND gates and inverters. Let  $B = \{0, 1\}$ . We now show how to extend any Boolean

function  $f: B^n \rightarrow B$  to a function  $\hat{f}: T^n \rightarrow T$ . We use the notation  $[n]$  to mean  $\{1, \dots, n\}$ .

Consider a simple example first. Suppose that a 2-input gate performs Boolean function  $f$ , and that the two inputs have transients 01 and 101, respectively. We want to find the maximum number of changes that can appear at the output of the gate, assuming that the input changes may occur at any time. Initially, the gate ‘sees’ the first letters of the two transients, i.e., 0 and 1; hence the output of the gate is  $f(0, 1)$ . One possibility is that the second transient occurs while the first input has the value 0. Then the gate would see the consecutive ordered pairs  $(0, 1)$ ,  $(0, 0)$ ,  $(0, 1)$ , and finally  $(1, 1)$ . Another possible order would be  $(0, 1)$ ,  $(0, 0)$ ,  $(1, 0)$ , and  $(1, 1)$ , and the third possible order would be  $(0, 1)$ ,  $(1, 1)$ ,  $(1, 0)$ , and  $(1, 1)$ . Note that we do not need to consider sequences like  $(0, 1)$ ,  $(1, 0)$ ,  $(1, 1)$ , in which both inputs change in some steps, since each such sequence is a subsequence of one of the three sequences above. The corresponding output sequence in the first case would be  $f(0, 1)$ ,  $f(0, 0)$ ,  $f(0, 1)$ ,  $f(1, 1)$ , in the second case,  $f(0, 1)$ ,  $f(0, 0)$ ,  $f(1, 0)$ ,  $f(1, 1)$ , and in the third case,  $f(0, 1)$ ,  $f(1, 1)$ ,  $f(1, 0)$ ,  $f(1, 1)$ . If  $f$  is the OR function, we would have the sequences 1011, 1011, and 1111, respectively. The corresponding transients would be 101, 101, and 1. Since we are looking for the longest possible transient, we have 101 as the result of the operation  $01 \oplus 101$ , which, of course, agrees with our definition in Section 4. Similarly, if  $f$  is the AND function, we get the result 0101, and if it is the XOR function, we have 1010.

This example is generalized as follows. Suppose that  $x = (x_1, \dots, x_n)$ , where  $x_i \in T$ , for each  $i \in [n]$ . Define the directed graph  $D(x)$  to have as vertices  $n$ -tuples  $y = (y_1, \dots, y_n)$ , where each  $y_i$  is a prefix of length  $> 0$  of  $x_i$ , for each  $i \in [n]$ . There is an edge from vertex  $y = (y_1, \dots, y_n)$  to vertex  $y' = (y'_1, \dots, y'_n)$  iff  $y$  and  $y'$  differ in exactly one coordinate, say  $i$ , and  $y'_i = y_i a$ , where  $a \in B$ . It is clear that the graph  $D(x)$  shows all possible orders in which the  $n$  variables can change, while undergoing a transition from the initial values  $(\alpha(x_1), \dots, \alpha(x_n))$  to the final values  $(x_1, \dots, x_n)$ . For the example given above, we have the graph of figure 8.

Let  $x = b_1 \dots b_n \in T$ . Call a word  $y$  an *expansion* of  $x$  if  $y$  results from  $x$  by replacing each letter  $b_i$ ,  $i \in [n]$  by  $b_i^{m_i}$ , for some  $m_i > 0$ . Note that for any nonempty word  $y$  over  $B$  there is a unique word  $x \in T$  such that  $y$  is an expansion of  $x$ . We call  $x$  the *contraction* of  $y$ . For example,  $y = 000110010$  is an expansion of  $x = 01010$ , and  $x$  is the contraction of  $y$ .

Returning now to graph  $D(x)$ , we label each vertex  $y = (y_1, \dots, y_n)$  of  $D(x)$  with the value  $f(a_1, \dots, a_n)$ , where  $a_i$  is the last letter of  $y_i$ , i.e.,  $a_i = \omega(y_i)$ , for each  $i \in [n]$ .

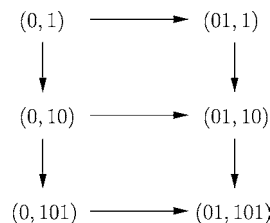


Figure 8. Graph  $D(01, 101)$ .

*Definition 1.* Given a Boolean function  $f : B^n \rightarrow B$ , we define function  $\hat{f}$  to be that function from  $T^n$  to  $T$  which, for any  $n$ -tuple  $(x_1, \dots, x_n)$  of transients, produces the longest transient when  $x_1, \dots, x_n$  are applied to the inputs of a gate performing function  $f$ .

The following proposition is now clear from the definition of  $\hat{f}$  and the graph  $D(x)$ .

**Proposition 7.1.** *The value of  $\hat{f}(x_1, \dots, x_n)$  is the contraction of the label sequence of those paths in  $D(x)$  from  $(\alpha(x_1), \dots, \alpha(x_n))$  to  $(x_1, \dots, x_n)$  which have the largest number of alternations between 0 and 1.*

We now give a complete description of the extension of the  $n$ -input XOR function  $f : B^n \rightarrow B$  to function  $\hat{f} : T^n \rightarrow T$ . For this function, no two adjacent vertices have the same label. Hence, the length of the transient  $\hat{f}(x_1, \dots, x_n)$  is the length of any path in  $D(x)$  from  $(\alpha(x_1), \dots, \alpha(x_n))$  to  $(x_1, \dots, x_n)$  (all such paths have the same length). One easily verifies that  $y = \hat{f}(x_1, \dots, x_n)$  is that word in  $T$  satisfying the conditions

$$\begin{aligned}\alpha(y) &= f(\alpha(x_1), \dots, \alpha(x_n)) \\ \omega(y) &= f(\omega(x_1), \dots, \omega(x_n)) \\ l(y) &= 1 + \sum_{i=1}^n (l(x_i) - 1) = 1 - n + \sum_{i=1}^n l(x_i),\end{aligned}$$

where the expression for the length of  $y$  has 1 for the initial state and  $l(x_i) - 1$  for the changes in variable  $i$ , for each  $i$ .

For the OR function, consider first an example with two inputs  $x_1$  and  $x_2$ . Suppose we have reached a vertex  $(y_1, y_2)$  in  $D(x) = D((x_1, x_2))$ , where  $(y_1, y_2)$  is not the final vertex and  $f(\omega(y_1), \omega(y_2)) = 0$ . Then we must have  $\omega(y_1) = \omega(y_2) = 0$ . The two successor vertices must be  $(y_1 1, y_2)$  and  $(y_1, y_2 1)$ , because of the alternating nature of transients. Thus, the next value of  $f$  must be 1, independently of the successor vertex. Suppose that vertex  $(y_1, y_2)$  is labeled 1, so that  $f(\omega(y_1), \omega(y_2)) = 1$ . Then it has a successor labeled 0 exactly when only one of  $y_1$  and  $y_2$  ends in 1. Moreover, if  $\omega(y_1) = 1$  and  $\omega(y_2) = 0$ , say, then the vertex has a successor labeled 0 if and only if  $(y_1 0, y_2)$  is a successor, so that  $y_1 0$  is a prefix of  $x_1$ . It follows that to obtain the maximum number of alternations in the output sequence, we should take a path with the largest number of vertices labeled 0, since any change is caused by entering, or leaving such a vertex.

In general, for any  $n$ -tuple  $(x_1, \dots, x_n)$  of transients, the maximum number of 0s is  $z = 1 + (z(x_1) - 1) + \dots + (z(x_n) - 1)$ . This holds, because we get the first 0 when we reach the first 0s in all the  $n$  transients, and then each variable  $i$  contributes  $z_i - 1$  additional 0s, while the remaining variables are held at 0. For example, for  $x = (01010, 0101)$  there are three sequences with  $1 + 2 + 1 = 4$  zeros:

$$\begin{aligned}(0, 0), (01, 0), (010, 0), (0101, 0), (01010, 0), (01010, 01), (01010, 010), \\ (01010, 0101),\end{aligned}$$



and

$$(0, 0), (0, 01), (0, 010), (01, 010), (010, 010), (0101, 010), (01010, 010), \\ (01010, 0101),$$

and

$$(0, 0), (01, 0), (010, 0), (010, 01), (010, 010), (0101, 010), (01010, 010), \\ (01010, 0101).$$

In summary,  $y = \hat{f}(x_1, \dots, x_n)$  is the word in  $T$  determined by the conditions

$$\begin{aligned} \alpha(y) &= \alpha(x_1) \vee \dots \vee \alpha(x_n) \\ \omega(y) &= \omega(x_1) \vee \dots \vee \omega(x_n) \\ z(y) &= \begin{cases} 0 & \text{if } \exists i \in [n] x_i = 1 \\ 1 + \sum_{i=1}^n (z(x_i) - 1) & \text{otherwise.} \end{cases} \end{aligned}$$

Dually, if  $f$  is the  $n$ -input AND function, then  $y = \hat{f}(x_1, \dots, x_n) \in T$  is given by

$$\begin{aligned} \alpha(y) &= \alpha(x_1) \wedge \dots \wedge \alpha(x_n) \\ \omega(y) &= \omega(x_1) \wedge \dots \wedge \omega(x_n) \\ u(y) &= \begin{cases} 0 & \text{if } \exists i \in [n] x_i = 0 \\ 1 + \sum_{i=1}^n (u(x_i) - 1) & \text{otherwise.} \end{cases} \end{aligned}$$

One easily verifies that the extension of the NOR (NAND) function of any number of arguments is the complement of the extension of the OR (AND) function. Note, however, that function composition does not preserve extensions in general. For a two input XOR gate with inputs 01 and 101 the output is 1010. Suppose now that the XOR gate is constructed with OR gates, AND gates and inverters. Then

$$(\overline{01} \otimes 101) \oplus (01 \otimes \overline{101}) = (10 \otimes 101) \oplus (01 \otimes 010) = 1010 \oplus 010 = 101010.$$

Thus, the hazard properties of the XOR function are quite different from those of the network  $N$  consisting of an OR gate, two AND gates, and two inverters. This difference can be explained as follows. The network  $N$  has wires connecting the inverters to AND gates and the AND gates to the OR gate. In our simulation algorithm we compute the worst-case transient for each gate output, given the transients at the gate inputs. In computing this transient we assume that the input changes may occur in all possible orders. This is equivalent to taking into account the wire delays. In contrast to this, if we consider the XOR function as being realized by a single component, these “intermediate” wire delays are not taken into account. Hence, a shorter transient may result in the latter case.

The following proposition and its corollary show that the monotonicity results of Section 4 for two-input OR gates, AND gates, and inverters, and consequently the monotonicity of Algorithms A and B, apply also to gates realizing arbitrary Boolean functions.

**Proposition 7.2.** *Let  $f$  be a Boolean function and  $\hat{f}$  its extension to transients. Then  $\hat{f}$  is monotonic with respect to the prefix order.*

**Proof:** When  $x = (x_1, \dots, x_n)$  and  $x' = (x'_1, \dots, x'_n)$  with  $x_i \leq x'_i$ , for all  $i \in [n]$ ,  $D(x)$  is a subgraph of  $D(x')$ . Also  $\alpha(x_i) = \alpha(x'_i)$ , for all  $i \in [n]$ , so that any path from  $\alpha(x) = (\alpha(x_1), \dots, \alpha(x_n))$  to  $x$  in  $D(x)$  can be continued to a path from  $\alpha(x') = (\alpha(x'_1), \dots, \alpha(x'_n))$  to  $x'$ . Thus, each label sequence corresponding to a path from  $\alpha(x)$  to  $x$  is a prefix of the label sequence corresponding to a path from  $\alpha(x')$  to  $x'$ . It follows that  $\hat{f}(x_1, \dots, x_n) \leq \hat{f}(x'_1, \dots, x'_n)$ .  $\square$

**Corollary 7.3.**  *$\hat{f}$  is monotonic with respect to the suffix order.*

It also follows from the definition of  $\hat{f}$  that the length of  $\hat{f}(x_1, \dots, x_n)$  is bounded by

$$1 + \sum_{i=1}^n (l(x_i) - 1) = 1 - n + \sum_{i=1}^n l(x_i).$$

When  $n = 1$ , this bound can be achieved for the identity function and function  $\bar{\phantom{x}}$ , and for  $n > 1$ , for the  $n$ -input XOR function.

The next proposition and two lemmas are technical results needed in the proof of Proposition 7.7.

**Proposition 7.4.** *Suppose that  $f$  depends on each of its arguments and none of the  $x_i$  is a single letter. Then the length of  $\hat{f}(x_1, \dots, x_n)$  is at least the maximum of the lengths of the  $x_i$ .*

**Proof:** See Appendix.  $\square$

For a function  $f : B^n \rightarrow B$ , integer  $i \in [n]$  and  $b \in B$ , let  $f_{i,b} : B^{n-1} \rightarrow B$  denote the function obtained by fixing the  $i$ th argument of  $f$  at  $b$ , i.e.,

$$f_{i,b}(b_1, \dots, b_{i-1}, b_{i+1}, \dots, b_n) = f(b_1, \dots, b_{i-1}, b, b_{i+1}, \dots, b_n),$$

for all  $b_j \in B$ ,  $j \in [n]$ ,  $j \neq i$ .

**Lemma 7.5.** *For all  $x_j \in T$ ,  $j \in [n]$ , if  $x_i = b$  is a single letter, then*

$$\hat{f}(x_1, \dots, x_n) = \hat{f}_{i,b}(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n). \quad (3)$$

**Proof:** See Appendix.  $\square$

**Lemma 7.6.** *If  $f : B^n \rightarrow B$  does not depend on its  $i$ th argument, then  $\hat{f}$  does not depend on that argument either.*

Table 13. Operation of a XOR gate for  $k = 3$ .

XOR	0	01	$\Phi$	10	1
0	0	01	$\Phi$	10	1
01	0	$\Phi$	$\Phi$	$\Phi$	10
$\Phi$	$\Phi$	$\Phi$	$\Phi$	$\Phi$	$\Phi$
10	10	$\Phi$	$\Phi$	$\Phi$	01
1	1	10	$\Phi$	01	1

**Proof:** See Appendix. □

**Proposition 7.7.** *Let  $k \geq 1$ , and let  $\sim_k$  be the equivalence relation on  $T$ , defined before. Then  $\sim_k$  is a congruence in the sense that, for any Boolean function  $f: B^n \rightarrow \{0, 1\}$ , and  $x_1, \dots, x_n, x'_1, \dots, x'_n \in T$ , if  $x_1 \sim_k x'_1, \dots, x_n \sim_k x'_n$ , then  $\hat{f}(x_1, \dots, x_n) \sim_k \hat{f}(x'_1, \dots, x'_n)$ .*

**Proof:** Our claim is clear for  $k = 1$ , since all transients are equivalent with respect to  $\sim_1$ . Hence we may assume that  $k > 1$ . It suffices to show that, if  $x_i \sim_k x'_i$  and  $x_j = x'_j$  for  $j \neq i$ , then  $\hat{f}(x_1, \dots, x_n) \sim_k \hat{f}(x'_1, \dots, x'_n)$ . The claim then follows by applying this result to each  $x_i$ . (For example, for  $n = 2$ , we first show that  $\hat{f}(x_1, x_2) \sim_k \hat{f}(x'_1, x_2)$ , and then that  $\hat{f}(x'_1, x_2) \sim_k \hat{f}(x'_1, x'_2)$ , and each step involves changing just one variable.) Since  $k > 1$ ,  $x_i$  is a single letter iff  $x'_i$  is a single letter, in which case  $x_i = x'_i$  and our claim is obvious. By Lemmas 7.5 and 7.6, we may thus assume that  $f$  depends on all of its arguments and none of the  $x_j$  is a single letter, so that  $x'_i$  is not a single letter either. But then Proposition 7.4 applies and the result follows. □

Since  $\sim_k$  is a congruence relation on  $T$ , it is meaningful to extend any Boolean function to  $T_k$ . For example, the XOR table for  $k = 3$  is shown in Table 13. It follows that the extension preserves the prefix and suffix order on  $T_k$ .

## 8. Complexity issues

In this section we give an estimate of the worst case performance of the simulation algorithms.

Using Algebra  $C$ , Algorithm A may not terminate unless the circuit is feedback-free. So suppose that we are given an  $m$ -input feedback-free circuit with  $n$  gates. We assume that each gate is an OR or AND gate, or an inverter, or more generally, a gate with a bounded number of inputs such that the extension of the binary gate function to  $T$  can be computed in linear time when each transient  $t$  is represented by the triple  $(\alpha(t), z(t), \omega(t))$ . Here, we assume that  $z(t)$ , the number of 0s in  $t$  is stored in binary.

Suppose the circuit is in a stable state and some of the inputs are supposed to change. If an input is to change from 0 to 1 (1 to 0), we set it to 01 (10). It takes  $O(m)$  time to

record these changes. Let  $s_1, \dots, s_n$  denote the state variables corresponding to the gates. Initially, each  $s_i$  has a binary value. For each  $i \in [n]$ , let  $h_i$  denote the height of the  $i$ th gate, i.e., the length of the longest path from an input to the gate corresponding to  $s_i$ . We clearly have that  $h_i \leq n$ , for each  $i \in [n]$ . In the first step of Algorithm A, all state variables  $s_i$  with  $h_i = 1$  will be set to their respective final values. More generally, after step  $j$ , all state variables  $s_i$  with  $h_i \leq j$  will assume their final values. Thus, Algorithm A terminates in  $O(n)$  steps. In each step, each variable  $s_i$  is set to a value according to its current excitation. Of course, this value may be the same as the value currently attained by the variable. For example, when the  $i$ th gate is an OR gate which takes its inputs from the  $j$ th and  $k$ th gates, then  $s_i$  is set to  $s_j \oplus s_k$ , the sum of the current values of  $s_j$  and  $s_k$ . The  $\oplus$  operation is that of Algebra  $C$ . Using the triple representation of transients, the middle component of the new value of  $s_i$  is at most twice the maximum of the middle components of  $s_j$  and  $s_k$ , so that its binary representation is at most one longer than the maximum of the lengths of the middle components of  $s_j$  and  $s_k$ . Since the algorithm terminates in  $O(n)$  steps, it follows that the value of any state variable can be stored in  $O(n)$  space. Moreover, we see that updating the value of  $s_i$  in a step takes  $O(n)$  time, and since there are  $n$  state variables, each step requires  $O(n^2)$  time. Thus, in  $O(n)$  steps, we can set all  $n$  state variables to their final values in  $O(n^3)$  time, showing that Algorithm A runs in  $O(m + n^3)$  time.

If the gates are given in topological order so that each  $s_i$  depends at most on the input variables and the state variables  $s_j$  with  $j < i$ , then a better performance can be achieved by an alternative algorithm that runs in  $n$  steps. In step  $i$ , it sets  $s_i$  to its final value according to its excitation. The time required is now  $O(m + n^2)$ . Since topological sort can be done within this time limit, the same bound applies if the gates are given in an arbitrary order.

If in Algorithm A we use Algebra  $C_k$ , for a fixed  $k$ , then at each step, the value of any state variable can be stored in space  $O(\log k)$  and updated in time  $O(k \log k)$  by a simple table look-up. In this case, Algorithm A terminates in  $O(n)$  steps for all circuits, including those with feedback, since  $T_k$  is finite, and the subsequent values of each state variable form a nondecreasing sequence with respect to the prefix order. This follows from the fact, proved in Section 7, that the extension of any Boolean function preserves the prefix order on  $T_k$ . Thus, using Algebra  $C_k$ , Algorithm A runs in  $O(m + n^2 k \log k)$  time, even for circuits with feedback.

If we use Algorithm A with Algebra  $C$  for a feedback-free circuit, then Algorithm B becomes unnecessary. The last letter of the final value of each state variable gives the response of the circuit to the intended change in the input. Moreover, the final value of each state variable is the transient that describes all of the (unwanted) intermediate changes that can take place in worst case at the respective gate. The same holds if we use  $C_k$  for any circuit, which now may contain cycles, such that upon termination of Algorithm A, each state variable assumes a value other than  $\Phi$ . However, if the final value of a state variable is  $\Phi$ , Algorithm B does become necessary. It will stop in no more than  $n$  steps, since the subsequent values of each state variable now form a nonincreasing sequence with respect to the suffix order. The total time required by Algorithm B is  $O(n^2 k \log k)$ .

It follows from the arguments presented above that the following decision problem is decidable in polynomial time: For a given circuit in a stable initial state, a given input change, and an integer  $k$ , are there  $k$  or more (unwanted) signal changes on the output of

any given gate, or in the entire circuit, during that input change? Also, it is decidable in nondeterministic polynomial time, for a given circuit in a stable initial state and an integer  $k$ , whether there exists an input change that would cause  $k$  or more (unwanted) signal changes on the output of any given gate, or in the entire circuit.

**Proposition 8.1.** *It is NP-complete to decide for an  $n$ -input Boolean function  $f$  given in conjunctive normal form, a tuple of transients  $x = (x_1, \dots, x_n)$  and integer  $k$  whether the length of  $\hat{f}(x_1, \dots, x_n)$  is  $> k$ .*

**Proof:** We can guess a path in the graph  $D(x)$  and verify in polynomial time if the contraction of the associated label sequence is longer than  $k$ , showing that the problem belongs to NP. As for NP-hardness, consider an  $n$ -input Boolean function  $f$  given by a conjunctive normal form  $\varphi$  such that  $f(0, \dots, 0) = 0$ . Then  $\varphi$  is satisfiable iff  $\hat{f}(01, \dots, 01)$  is not 0, i.e., when the length of  $\hat{f}(01, \dots, 01)$  is  $> 1$ . To see this, suppose first that  $\varphi$  is satisfiable and let  $x = (01, \dots, 01)$ . Since  $\varphi$  is satisfiable, there is some  $b = (b_1, \dots, b_n) \in \{0, 1\}^n$  such that  $f(b) = 1$ . Now let  $y_i = 0$  if  $b_i = 0$ , and let  $y_i = 01$ , if  $b_i = 1$ , for each  $i \in [n]$ . Thus,  $\omega(y) = (\omega(y_1), \dots, \omega(y_n)) = b$ , so that  $y = (y_1, \dots, y_n)$  is a vertex of  $D_x$  labeled by  $f(b) = 1$ . Take any path from  $\alpha(x) = (0, \dots, 0)$  to  $x$  going through  $y$ . Since vertex  $(0, \dots, 0)$  is labeled by 0 and  $y$  is labeled by 1, the contraction of the label sequence associated with this path has length at least 2, showing that the length of  $\hat{f}(01, \dots, 01)$  is  $> 1$ . On the other hand, if the length of  $\hat{f}(01, \dots, 01)$  is  $> 1$ , then at least one vertex  $y = (y_1, \dots, y_n)$  of  $D(x)$  is labeled by 1. Thus, letting  $b = \omega(y) = (\omega(y_1), \dots, \omega(y_n))$ , we have that  $f(b) = 1$ , so that  $\varphi$  is satisfiable.  $\square$

## 9. Simulation with initial, middle, and final values

In a number of simulators [10, 18], the signal values are ordered triples containing the initial, transient, and final values of a signal. We now show how such algebras can be described in our framework.

For  $k \geq 1$ , relation  $\approx_k$  in the algebra  $C = (T, \oplus, \otimes, ^-, 0, 1)$  is defined as follows: For  $t, s \in T$ ,  $t \approx_k s$  if either  $t = s$  or  $\alpha(t) = \alpha(s)$ ,  $\omega(t) = \omega(s)$ , and  $t$  and  $s$  are both of length  $\geq k$ . Denote by  $\lambda$  (for left) and  $\rho$  (for right) the congruences defined by

$$\begin{aligned} t \lambda s &\text{ iff } \alpha(t) = \alpha(s), \\ t \rho s &\text{ iff } \omega(t) = \omega(s). \end{aligned}$$

Then

$$\approx_k = \lambda \cap \sim_k \cap \rho.$$

**Proposition 9.1.** *Relation  $\approx_k$  is a congruence relation on  $C$ .*

**Proof:** Since  $\lambda$ ,  $\rho$ , and  $\sim_k$  are congruences, so is  $\approx_k$ .  $\square$

Table 14. Simulation with initial, middle, and final values.

	$X$	$s$
Initial state	0	0
	$0\Phi 1$	0
	$0\Phi 1$	$0\Phi 1$
	$0\Phi 1$	$0\Phi 0$
	$0\Phi 1$	$0\Phi 1$
	...	...

The quotient algebra  $C'_k = C/\approx_k$  is a commutative de Morgan bisemigroup with  $2(k-1) + 4 = 2k + 2$  elements. Each word  $t \in T$  with  $l(t) < k$  determines a singleton congruence class. In addition, for any  $b_1, b_2 \in B$ , the words  $t \in T$  with  $l(t) \geq k$ ,  $\alpha(t) = b_1$ , and  $\omega(t) = b_2$  determine a congruence class that we denote by  $b_1 \Phi b_2$ . Since  $\approx_k \subseteq \sim_k$ ,  $C_k$  is a quotient of  $C'_k$ . It can be constructed from  $C'_k$  by identifying the four elements  $0\Phi 0$ ,  $0\Phi 1$ ,  $1\Phi 0$ , and  $1\Phi 1$ . Also, if  $k \leq m$ , then  $C'_k$  is a quotient of  $C'_m$ .

**Proposition 9.2.**  $C'_k$  is isomorphic to a subdirect product (i.e., is a subalgebra of the direct product) of two copies of the 2-element Boolean algebra  $B$  and Algebra  $C_k$ .

**Proof:** This follows from the fact that  $\approx_k = \lambda \cap \sim_k \cap \rho$ , and that  $C/\lambda$  and  $C/\rho$  are both isomorphic to  $B$ .  $\square$

Let  $T'_k = T_k \setminus \{\Phi\} \cup \{0\Phi 0, 0\Phi 1, 1\Phi 0, 1\Phi 1\}$ . We can extend any Boolean function  $B^n \rightarrow B$  to a function  $(T'_k)^n \rightarrow T$  by using the congruence property of  $\approx_k$ .

*Example 12.* Consider the circuit consisting of a 2-input XOR gate with output  $s$  and inputs  $X$  and  $s$ . If the circuit is started in state  $X = 0, s = 0$ , it is stable. The simulation in  $C'_2$  is shown in Table 14 when the input changes to  $0\Phi 1$ . This simulation does not terminate.

## 10. Conclusions

We conclude the paper with a short summary of our results. Our main contribution is a general treatment of signal changes and hazards that encompasses the existing methods and permits a systematic study and comparison of these approaches. Some detailed properties of our method are highlighted below.

- *Hazards*

We have presented a general theory of simulation of gate circuits for the purpose of hazard detection, identification, and counting.

- *Energy estimation*

The same simulation algorithms can be used to count the number of signal changes during a given input change of a circuit. This provides an estimate of the worst-case energy consumption of that input change.

- *Efficiency*  
If a circuit has  $m$  inputs and  $n$  gates, our simulation algorithms run in  $O(m + n^2)$  time, if the gates are sorted, and in  $O(m + n^3)$  time, otherwise.
- *Accuracy*  
Our unified model shows in a very natural way the fact that capability for identifying hazards increases monotonically with the number of symbols used in multivalued extensions of Boolean algebras. By choosing the value of the threshold  $k$  one can count signal changes and hazards to any degree of accuracy.
- *Feedback-free circuits*
  - In Algebra  $C$  Algorithm A always terminates, and Algorithm B is not required.
  - In Algebra  $C_k$  Algorithm A produces a result without  $\Phi$ s if  $k$  is sufficiently large. In that case, Algorithm B is not needed.
  - Simulation with (initial, middle, final) values terminates in Algebra  $C'_k$ .
- *Circuits with feedback*
  - In Algebra  $C$  Algorithm A may not terminate.
  - Simulation in Algebra  $C_k$  always terminates.
  - Simulation with (initial, middle, final) values may not terminate; hence, it is not suitable for such circuits.
- *Multivalued algebras*  
Many known algebras are included as special cases in our theory:
  - Ternary algebra is isomorphic to Algebra  $C_2$ .
  - The quinary algebra of Lewis [19] is isomorphic to Algebra  $C_3$ .
  - The 6-valued algebra  $H_6$  of Hayes [18] is isomorphic to  $C'_2$ .
  - The 8-valued algebra of Breuer and Harrison [1] and Fantauzzi [13, 18] is isomorphic to  $C'_3$ .
- *Decision problems*
  - For a given circuit in a stable initial state, a given input change, and an integer  $k$ , it is decidable in polynomial time whether there are  $k$  or more (unwanted) signal changes on the output of any given gate, or in the entire circuit, during that input change.
  - For a given circuit in a stable initial state and an integer  $k$ , it is decidable in non-deterministic polynomial time whether there exists an input change that would cause  $k$  or more (unwanted) signal changes on the output of any given gate, or in the entire circuit.

### Appendix: Additional properties of Algebra $C$

We now state without proof some basic properties of  $C$ . Each word in  $T$  is generated by the two transients 01 and 10 representing up and down changes. Since  $C$  is a bisemigroup, both operations are used. The word 0 is the trivial sum of zero generators, and 1 is the trivial product of zero generators.<sup>1</sup> Words 10 and 01 are the generators themselves. Next  $010 = 01 \otimes 10$ ,  $101 = 10 \oplus 01$ ,  $0101 = 010 \oplus 01 = 01 \otimes 10 \oplus 01$ , etc.

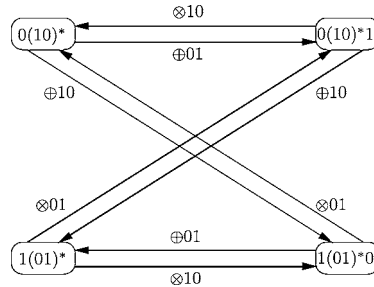


Figure 9. Operations in  $C$ .

We refer to any expression of the form  $10 \oplus 01 \otimes 10 \oplus 01 \dots$  or  $01 \otimes 10 \oplus 01 \otimes 10 \dots$  as an *alternation* of generators. One can verify that the order in which the operations are applied is immaterial, for example,  $(10 \oplus 01) \otimes (10 \oplus 01) = ((10 \oplus 01) \otimes 10) \oplus 01 = (10 \oplus (01 \otimes 10)) \oplus 01$ .

Every word in  $t \in T$  is an alternation of generators. In general, we can obtain this representation of  $t$  as follows. Duplicate all the letters in  $t$ , except the first and the last. This expresses  $t$  as a concatenation of up and down changes. Insert  $\oplus$  between every pair of 0s, and  $\otimes$  between every pair of 1s. The resulting expression is a representation of  $t$  in terms of the two generators. If  $l(t) = n$ , then  $t$  is an alternation of  $n - 1$  generators. For example, let  $t = 01010101$ . We first find  $01100110011001$  and then insert the operations, to obtain  $01 \otimes 10 \oplus 01 \otimes 10 \oplus 01 \otimes 10 \oplus 01$ .

Figure 9 shows how addition of a generator or multiplication by a generator affects any word in  $T$ . The figure shows a transition graph with four states, each labeled by a regular expression. The languages denoted by the four regular expressions are pairwise disjoint. Given a word  $t \in T$ , start in that state  $q$  whose expression contains  $t$ . To find  $t \odot g$ , where  $\odot$  is  $\oplus$  or  $\otimes$  and  $g$  is a generator, look for a transition labeled  $\odot g$ . If there is no such transition, then  $t \odot g = t$ . Otherwise, follow the transition indicated; let the state reached be  $q'$ . Then  $t \odot g$  is the word of length  $l(t) + 1$  that is contained in the regular expression of  $q'$ . For example,  $1010$  belongs to state  $1(01)^*0$ . From that state we find  $1010 \oplus 01 = 10101$ ,  $1010 \oplus 10 = 1010$ ,  $1010 \otimes 01 = 01010$ , and  $1010 \otimes 10 = 1010$ .

From the graph of figure 9 one can also read off  $t \oplus t'$  and  $t \otimes t'$  for any two words  $t, t' \in T$ . Start in the state containing  $t$ , and follow the transitions in the alternation of  $t'$ . For example, consider  $t = 1010$  and  $t' = 010$ . The alternation of  $t'$  is  $01 \otimes 10$ . We start in the state labeled  $1(01)^*0$  which contains  $1010$ . To find  $t \otimes t'$  we first look for  $\otimes 01$ ; this leads us to  $01010$ . Next we look for  $\otimes 10$ ; there is no change of state. Hence, the result is  $01010$ .

**Proof of Proposition 7.4:** Let  $x_i$  be one of the longest of the words  $x_1, \dots, x_n \in T$ , and let  $m$  denote the length of  $x_i$ . For each  $k \in [m]$ , let  $z_k$  denote the prefix of  $x_i$  of length  $k$ . Since  $f$  depends on its  $i$ th argument, there exist  $b_j \in B$ ,  $j \neq i$  such that

$$f(b_1, \dots, b_{i-1}, 0, b_{i+1}, \dots, b_n) \neq f(b_1, \dots, b_{i-1}, 1, b_{i+1}, \dots, b_n). \tag{4}$$

Since none of the  $x_j$  is a single letter, there is a vertex in  $D(x)$  of the form

$$(y_1, \dots, y_{i-1}, z_1, y_{i+1}, \dots, y_n)$$



such that  $\omega(y_j) = b_j$ , for all  $j \neq i$ . It follows that for each  $k \in [m]$ ,

$$v_k = (y_1, \dots, y_{i-1}, z_k, y_{i+1}, \dots, y_n)$$

is a vertex of  $D(x)$ ; moreover, there is a path  $p$  from  $v_1$  to  $v_m$  whose internal vertices are  $v_2, \dots, v_{m-1}$ . By (4), the label sequence associated with this path is alternating. Thus, the contraction of the label sequence of any path from  $(\alpha(x_1), \dots, \alpha(x_n))$  to  $(x_1, \dots, x_n)$  which contains  $p$  as a subpath has length at least  $k$ .  $\square$

**Proof of Lemma 7.5:** Any vertex  $(y_1, \dots, y_n) \in D(x)$ , where  $x = (x_1, \dots, x_n)$ , has a single  $b$  as its  $i$ th component, i.e.,  $y_i = b$ . Thus, denoting  $x' = (x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$ , it holds that  $(y_1, \dots, y_{i-1}, y_{i+1}, \dots, y_n) \in D(x')$  and

$$f(\omega(y_1), \dots, \omega(y_n)) = f_{i,b}(\omega(y_1), \dots, \omega(y_{i-1}), \omega(y_{i+1}), \dots, \omega(y_n)). \quad (5)$$

Hence any label sequence corresponding to a path from  $\alpha(x) = (\alpha(x_1), \dots, \alpha(x_n))$  to  $x$  in  $D(x)$  determined by function  $f$  is also the label sequence of a path from  $\alpha(x') = (\alpha(x_1), \dots, \alpha(x_{i-1}), \alpha(x_{i+1}), \dots, \alpha(x_n))$  to  $x'$  in  $D(x')$  determined by function  $f_{i,b}$ .

Conversely, if the  $(n-1)$ -tuple  $(y_1, \dots, y_{i-1}, y_{i+1}, \dots, y_n)$  is in  $D(x')$ , then

$$(y_1, \dots, y_{i-1}, b, y_{i+1}, \dots, y_n)$$

is in  $D(x)$ . It follows that any word which is the label sequence of a path from  $\alpha(x')$  to  $x'$  in  $D(x')$  determined by  $f_{i,b}$  is a label sequence of a path from  $\alpha(x)$  to  $x$  in  $D(x)$  determined by  $f$ .  $\square$

**Proof of Lemma 7.6:** If  $f : B^n \rightarrow B$  does not depend on its  $i$ th argument, then for all  $x_1, \dots, x_n \in T$ , for any  $(y_1, \dots, y_n)$  in  $D(x)$ , where  $x = (x_1, \dots, x_n)$ , and for any  $b \in B$ ,

$$f(\omega(y_1), \dots, \omega(y_n)) = f(\omega(y_1), \dots, \omega(y_{i-1}), b, \omega(y_{i+1}), \dots, \omega(y_n)).$$

It follows that

$$\hat{f}(x_1, \dots, x_n) = \hat{f}(x_1, \dots, x_{i-1}, b, x_{i+1}, \dots, x_n),$$

proving that  $\hat{f}$  is independent of  $x_i$ .  $\square$

### Acknowledgment

The authors thank Steve Nowick for his careful reading of our paper and for his constructive comments.

### Note

1. This agreement is standard in algebra, and is analogous to considering any nonnegative integer  $n$  as a sum of  $n$  1s; then the integer 0 is the sum of zero 1s.

## References

1. M.A. Breuer and L. Harrison, "Procedures for eliminating static and dynamic hazards in test generation," *IEEE Trans. on Computers*, Vol. C-23, pp. 1069–1078, 1974.
2. J.A. Brzozowski, "A survey of regular expressions and their applications," *IRE Trans. on Electronic Computers*, Vol. EC-11, No. 3, pp. 324–335, 1962.
3. J.A. Brzozowski, "Some applications of ternary algebras," *Publicationes Mathematicae (Debrecen)*, Vol. 54 (Suppl.), pp. 583–599, 1999.
4. J.A. Brzozowski, "De Morgan bisemilattices," in *Proc. 30th Int. Symp. on Multiple-Valued Logic*, Portland, OR, IEEE Computer Society Press, Los Alamitos, CA, May 2000, pp. 173–178.
5. J.A. Brzozowski and Z. Ésik, "Hazard algebras (Extended Abstract)," in *A Half-Century of Automata Theory*, A. Salomaa, D. Wood, and S. Yu (Eds.), World Scientific, Singapore, 2001, pp. 1–19.  
To put a crossline over L
6. J.A. Brzozowski, Z. Ésik, and Y. Iland, "Algebras for hazard detection," in *Proc. 31st Int. Symp. on Multiple-Valued Logic*, Warsaw, Poland, IEEE Computer Society Press, Los Alamitos, CA, May 2000, pp. 3–12. Also reprinted in M. Fitting and E. Orłowska (Eds.), *Beyond Two: Theory and Applications of Multiple-Valued Logic*, Physica-Verlag, 2003, pp. 3–24.
7. J.A. Brzozowski and M. Gheorghiu, "Simulation of gate circuits in the algebra of transients," in *7th Int. Conf. on Implementation and Application of Automata*, Tours, France, July 2002. Lecture Notes in Computer Science, vol. 2608, Springer 2003, pp. 57–66.
8. J.A. Brzozowski and C.-J.H. Seger, *Asynchronous Circuits*, Springer-Verlag, 1995.
9. S. Burris and H.P. Sankappanavar, *A Course in Universal Algebra*, Springer-Verlag, 1981.
10. S. Chakraborty and D.L. Dill, "More accurate polynomial-time min-max simulation," in *Proc. 3rd Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, Eindhoven, The Netherlands, IEEE Computer Society Press, Los Alamitos, CA, April 1997, pp. 112–123.
11. B.A. Davey and H.A. Priestley, *Introduction to Lattices and Order*, Cambridge University Press, 1990.
12. E.B. Eichelberger, "Hazard detection in combinational and sequential switching circuits," *IBM J. Research and Development*, Vol. 9, pp. 90–99, 1965.
13. G. Fantauzzi, "An algebraic model for the analysis of logic circuits," *IEEE Trans. on Computers*, Vol. C-23, pp. 576–581, 1974.
14. M. Gheorghiu, "Circuit simulation using a hazard algebra," MMath Thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1, Dec. 2001. <http://maveric.uwaterloo.ca/publication.html>.
15. M. Gheorghiu and J.A. Brzozowski, "Feedback-free circuits in the algebra of transients," in *7th Int. Conf. on Implementation and Application of Automata*, Tours, France, July 2002. Lecture Notes in Computer Science, vol. 2608, Springer 2003, pp. 106–116.
16. M. Goto, "Application of three-valued logic to construct the theory of relay networks" (in Japanese), in *Proceedings of the Joint Meeting of IEE, IECE, and I. of Illumination E. of Japan*, 1948.
17. G. Grätzer, *Universal Algebra*, 2nd ed., Springer-Verlag, 1979.
18. J.P. Hayes, "Digital simulation with multiple logic values," *IEEE Trans. on Computer-Aided Design*, Vol. CAD-5, No. 2, 1986.
19. D.W. Lewis, "Hazard detection by a quinary simulation of logic devices with bounded propagation delays," MSc Thesis, Electrical Engineering, Syracuse University, Syracuse, NY, Jan. 1972.
20. D.E. Muller, "A theory of asynchronous circuits," Technical Report 66, Digital Computer Laboratory, University of Illinois, Urbana-Champaign, Illinois, USA, 1955.
21. D.E. Muller and W.S. Bartky, "A theory of asynchronous circuits," in *Proceedings of an International Symposium on the Theory of Switching*, Annals of the Computation Laboratory of Harvard University, Harvard University Press, 1959, pp. 204–243.
22. A. Salomaa, *Theory of Automata*, Pergamon Press, Oxford, 1969.
23. S.H. Unger, *Asynchronous Sequential Switching Circuits*, Wiley-Interscience, 1969.