Feedback-Free Circuits in the Algebra of Transients

Mihaela Gheorghiu and Janusz Brzozowski

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

Abstract. An efficient simulation algorithm using an algebra of transients for gate circuits was proposed by Brzozowski and Ésik. This algorithm seems capable of predicting all the signal changes that can occur in a circuit under worst-case delay conditions. We verify this claim by comparing simulation with binary analysis. For any feedback-free circuit consisting of 1- and 2-input gates and started in a stable state, we prove that all signal changes predicted by simulation occur in binary analysis, provided that wire delays are taken into account. Two types of finite automata play an important role in our proof.

1 Introduction

Detecting signal changes in digital circuits is important, because unwanted (hazardous) signal changes may affect the correctness of computations, and increase the computation time and energy consumption. To address this problem, Brzozowski and Ésik [1] proposed an infinite-valued algebra C of transients, and an efficient simulation algorithm for gate circuits based on this algebra. In a companion paper [2] we compare the simulation of a circuit in C to the traditional binary analysis. We show that simulation of an arbitrary circuit is sufficient: all the changes that occur in binary analysis are also predicted by the simulation. In general, however, simulation is more pessimistic than binary analysis. It is the purpose of this paper to determine how pessimistic the simulation can be.

Here we consider the class of feedback-free gate circuits with stable initial states. Although this is a special case, it is important in practice. For this case, we show that all the changes predicted by simulation also occur in binary analysis, provided that wire delays are taken into account. Our result is limited to gate circuits constructed with 1- or 2-input gates; the general case remains open.

This paper is as self-contained as possible. The reader should see [2] for more details, and [5] for complete background information and proofs.

2 Circuits, Networks, and Binary Analysis

For an integer $n > 0$, $[n]$ denotes $\{1, \ldots, n\}$. Boolean operations OR, NOT, and XOR are denoted $\vee$, $\overline{}$, and $\veebar$, respectively.

Figure 1 (left) shows a gate circuit consisting of an inverter and an OR gate. It has input variable X_1 and state variables s_a and s_b. Each state variable s_i has an *excitation*

S_i, which is the Boolean function of the corresponding gate. Here, $S_a = \overline{X_1}$, and $S_b = X_1 \vee s_a$. The value of a variable may be different from that of its excitation. This allows us to represent the delay of a gate. A variable normally follows its excitation. However, if the excitation changes quickly, the variable may fail to follow the excitation, because of the inertial nature of the delay.

To account for other delays, we construct the *complete* counterpart of a circuit by adding the following variables. For each input X_i, we add an *input-gate* variable s_i; we represent the input gate by a triangle. We also consider each fork as a *fork gate*,[1] and add a variable for each fork output; we represent a fork gate by a rectangle. Finally, we add a variable for each wire. The excitations of the added variables are identity functions. For our example, see Fig. 1. We add input-gate variable s_1, fork-gate variables s_3 and s_4, wire variables s_2, s_5, s_7, and s_8, and we relabel s_a as s_6 and s_b as s_9. The new excitations are: $S_1 = X_1,\ S_2 = s_1,\ S_3 = S_4 = s_2,\ S_5 = s_3,\ S_6 = \overline{s_5},\ S_7 = s_6,\ S_8 = s_4,\ S_9 = s_7 \vee s_8$.

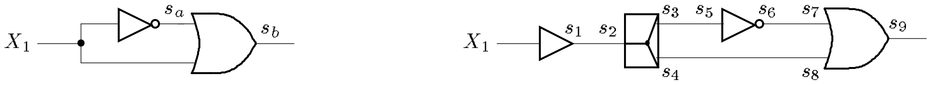

Fig. 1. Circuit C_1 and its complete version

Any circuit, complete or not complete, is modeled by a network.

Definition 1. *A* network *[3] is a tuple $N = \langle \mathcal{D}, \mathcal{X}, \mathcal{S}, \mathcal{E} \rangle$, where $\mathcal{D}$ is the domain of values, $\mathcal{X} = \{X_1, \ldots, X_n\}$, the set of inputs, $\mathcal{S} = \{s_1, \ldots, s_m\}$, the set of state variables with associated excitations $S_1, \ldots, S_m$, and $\mathcal{E} \subseteq (\mathcal{X} \times \mathcal{S}) \cup (\mathcal{S} \times \mathcal{S})$, a set of directed edges. There is an edge between x and y iff the excitation of y depends on x. The* network graph *is the digraph $(\mathcal{X} \cup \mathcal{S}, \mathcal{E})$.*

A *state* of N is an m-tuple b of values from $\mathcal{D}$ assigned to state variables $s_1, \ldots, s_m$. A *total state* is an $(n+m)$-tuple $c = a \cdot b$ of values from $\mathcal{D}$, the n-tuple a being the inputs, and the m-tuple b, the state. The value of S_i in $a \cdot b$ is denoted $S_i(a \cdot b)$. The tuple of all $S_i(a \cdot b)$, for $i \in [m]$, is denoted $S(a \cdot b)$. For any $a \cdot b$, the set of unstable state variables is $U(a \cdot b) = \{s_i \mid b_i \neq S_i(a \cdot b)\}$. Thus, $a \cdot b$ is *stable* iff $U(a \cdot b) = \emptyset$. For any state variable $s_i \in \mathcal{S}$, we define its *fan-in set* $\phi(s_i) = \{s_j \mid s_j \in \mathcal{S}, (s_j, s_i) \in \mathcal{E}\}$. We call state variables s_i and s_j *related* if $s_i \mathcal{E}^+ s_j$ or $s_j \mathcal{E}^+ s_i$, where $\mathcal{E}^+$ is the transitive closure of $\mathcal{E}$; otherwise, we call them *unrelated*.

For binary analysis we use the binary domain, $\mathcal{D} = \{0, 1\}$. We describe the behavior of a network started in a given state with the input kept constant at value $a \in \{0,1\}^n$ by defining a binary relation R_a on the set $\{0,1\}^m$ of states of N. For any $b \in \{0,1\}^m$, bR_ab, if $U(a \cdot b) = \emptyset$, and bR_ab^K, if $U(a \cdot b) \neq \emptyset$, and K is any nonempty subset of $U(a \cdot b)$, where by b^K we mean b with the variables in K complemented. No other pairs are related by R_a. As usual, we associate a digraph with the R_a relation.

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

[1] Our reason for calling input delays and forks *gates* will become clear later.

Example 1. For the complete circuit in Figure 2, with excitations $S_1 = X_1,\ S_2 = X_2,\ S_3 = s_1,\ S_4 = s_2,\ S_5 = s_3 \vee s_4$, we show $G_{01}(10101)$, where tuples are shown as words, unstable variables are underlined, and boldface features and edge labels are for later use.

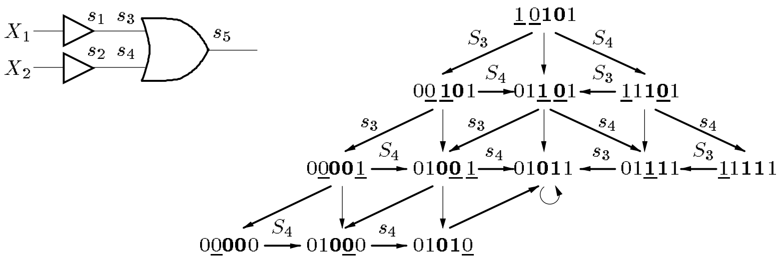

Fig. 2. Circuit C_2 and its binary analysis

3 Transients, Gate Automata, and Extended Functions

A *transient* [1] is a nonempty word over $\{0,1\}$ in which no two consecutive symbols are the same. Thus the set of all transients is $\mathbf{T} = 0(10)^* \cup 1(01)^* \cup 0(10)^*1 \cup 1(01)^*0$. Transients represent changing signals in a natural way; for instance, transient 010 represents a signal changing from 0 to 1 to 0. For any $\mathbf{t} \in \mathbf{T}$ we denote by $\alpha(\mathbf{t})$ and $\omega(\mathbf{t})$ its first and last characters, respectively, and by $|\mathbf{t}|$, its length. A transient can be obtained from any nonempty binary word by *contraction*, *i.e.*, the elimination of duplicates immediately following a symbol (*e.g.*, the contraction of 00100011 is 0101). For a binary word s we denote its contraction by $\hat{s}$. For any $\mathbf{t}, \mathbf{t}' \in \mathbf{T}$, we denote by $\mathbf{t} \circ \mathbf{t}'$ concatenation followed by contraction, *i.e.*, $\mathbf{t} \circ \mathbf{t}' = \widehat{\mathbf{t}\mathbf{t}'}$.

Extensions of Boolean functions to the domain $\mathbf{T}$ of transients were defined in [1]. Here we give an equivalent definition using finite automata, needed for later proofs. For common Boolean functions, [1] gives simpler formulas for computing extensions. For example, let $\oplus$ be the extension of the OR function. Then, for transients $\mathbf{w}, \mathbf{w}'$ of length > 1, $\mathbf{w} \oplus \mathbf{w}' = \mathbf{t}$, where $\mathbf{t}$ is such that $\alpha(\mathbf{t}) = \alpha(\mathbf{w}) \vee \alpha(\mathbf{w}')$, $\omega(\mathbf{t}) = \omega(\mathbf{w}) \vee \omega(\mathbf{w}')$, and $z(\mathbf{t}) = z(\mathbf{w}) + z(\mathbf{w}') - 1$, where $z(\mathbf{t})$ is the number of 0s in $\mathbf{t}$. Also, $\mathbf{t} \oplus 0 = 0 \oplus \mathbf{t} = \mathbf{t}$, and $\mathbf{t} \oplus 1 = 1 \oplus \mathbf{t} = 1$.

Let s_i be the state variable of a gate, let $\phi(s_i) = \{s_1, \dots, s_k\}$, and let the excitation S_i be the Boolean function $f : \{0,1\}^k \to \{0,1\}$. We extend f to $\mathbf{f} : \mathbf{T}^k \to \mathbf{T}$. For any tuple$(\mathbf{t}_1, \dots, \mathbf{t}_k)$ of transients, $\mathbf{f}(\mathbf{t}_1, \dots, \mathbf{t}_k)$ is the longest transient the gate could produce at its output, if its inputs changed as shown by the k transients. Formally, the definition uses a finite automaton to model the gate behavior. For any $j \in [k]$, we denote by e_j the j-th unit tuple in $\{0,1\}^k$ that has a 1 in position j and 0s elsewhere. By taking the component-wise exclusive OR of a tuple $t \in \{0,1\}^k$ with e_j, we obtain the tuple $t' \in \{0,1\}^k$ that differs from t only in the jth component.

Definition 2. *The* gate automaton *of a gate s_i with Boolean function $f : \{0,1\}^k \to \{0,1\}$ is an (uninitialized) Moore machine $\mathcal{G}_i = (I_i, O, \mathcal{P}_i, \tau_i, o_i)$, where $I_i = \phi(s_i)$ is* *the input alphabet,* $O = \{0, 1\}$*, the output alphabet,* $\mathcal{P}_i = \{0, 1\}^k$*, the set of states,* $\tau_i : \mathcal{P}_i \times I_i \to \mathcal{P}_i$*, the transition function defined for* $p \in \mathcal{P}_i$*,* $s_j \in I_i$ *as* $\tau_i(p, s_j) = p \veebar e_j$*, and* $o_i : \mathcal{P}_i \to O$*, the output function defined for* $p \in \mathcal{P}_i$ *as* $o_i(p) = f(p)$*. If* $\mathcal{G}_i$ *has an initial state* p_i^0*, then it is denoted by* $(\mathcal{G}_i, p_i^0)$*.*

Example 2. The automaton of an OR gate with inputs s_1 and s_2 is shown in Fig. 3.

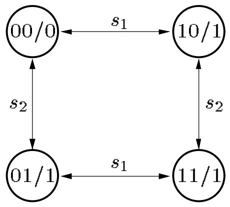

Fig. 3. OR gate automaton

We extend τ_i to words as follows: for the empty word ϵ, $\tau_i(p, \epsilon) = p$, and for any $p \in \mathcal{P}_i$ and $w \in I_i^*$, $\tau_i(p, ws_j) = \tau_i(p, w) \veebar e_j$. Any state of $\mathcal{G}_i$ can be the initial state, depending on the tuple of transients for which we compute the extension. Suppose the initial state is p_i^0. In $(\mathcal{G}_i, p_i^0)$, any input word $u \in I_i^*$ produces an output word v as follows. If u is the empty word, then $v = f(p_i^0)$. Otherwise, $u = u's_j$ produces $v = wf(\tau_i(p_i^0, u))$, where w is the output word of u'. The contraction $\hat{v}$ of v is the *output profile* of u. For any tuple $(\mathbf{t}_1, \ldots, \mathbf{t}_k)$ of transients, describing the changes of variables $s_1, \ldots, s_k$, we choose $p_i^0 = (\alpha(\mathbf{t}_1), \ldots, \alpha(\mathbf{t}_k))$. Thus p_i^0 shows the initial values of $s_1, \ldots, s_k$. An input word $u \in I_i^*$ *determines* the unique tuple $(\mathbf{t}_1, \ldots, \mathbf{t}_k)$ of transients if $|u|_{s_j} = |\mathbf{t}_j| - 1$, for all $j \in [k]$, where $|u|_{s_j}$ is the number of times s_j occurs in u. There may be several input words determining a given tuple of transients; let $\mathcal{U}(\mathbf{t}_1, \ldots, \mathbf{t}_k)$ be the set of all words determining $(\mathbf{t}_1, \ldots, \mathbf{t}_k)$. Let $\mathcal{V}(\mathbf{t}_1, \ldots, \mathbf{t}_k)$ be the set of output profiles of the words in $\mathcal{U}(\mathbf{t}_1, \ldots, \mathbf{t}_k)$, and let $v_{max}(\mathbf{t}_1, \ldots, \mathbf{t}_k)$ be the longest profile in $\mathcal{V}(\mathbf{t}_1, \ldots, \mathbf{t}_k)$.

Definition 3. *For any Boolean function* $f : \{0, 1\}^k \to \{0, 1\}$*, we define its extension* $\mathbf{f} : \mathbf{T}^k \to \mathbf{T}$ *by* $\mathbf{f}(\mathbf{t}_1, \ldots, \mathbf{t}_k) = v_{max}(\mathbf{t}_1, \ldots, \mathbf{t}_k)$*, for all* $(\mathbf{t}_1, \ldots, \mathbf{t}_k) \in \mathbf{T}^k$*.*

Definition 4. *For gate automaton* $(\mathcal{G}_i, p_i^0)$*, an input word* $u \in I_i^*$ *is called* worst-case *if* u *determines* $(\mathbf{t}_1, \ldots, \mathbf{t}_k)$*, and has the longest output profile among the words in* $\mathcal{U}(\mathbf{t}_1, \ldots, \mathbf{t}_k)$*, i.e., if* $\hat{v} = v_{max}(\mathbf{t}_1, \ldots, \mathbf{t}_k)$*, where* v *is the output word of* u*.*

Example 3. We illustrate the new concepts of this section. Consider the automaton of Fig. 3, with $p_i^0 = 00$. The output produced by $u = s_1s_2s_1$ is $v = 0111$, and its output profile is $\hat{v} = 01$. If $u = s_1s_1s_2s_2$, then $v = 01010$, and $\hat{v} = 01010$. Also

$$\mathcal{U}(010, 010) = \{s_1s_2s_1s_2,\ s_1s_2s_2s_1, s_1s_1s_2s_2,\ s_2s_1s_1s_2,\ s_2s_1s_2s_1,\ s_2s_2s_1s_1\}.$$

We compute $01 \oplus 010$. We find $\mathcal{U}(01, 010) = \{s_1s_2s_2,\ s_2s_1s_2,\ s_2s_2s_1\}$ and $\mathcal{V} = \{01, 0101\}$. Hence $01 \oplus 010 = 0101$. Word $s_2s_2s_1$ is worst-case, in contrast to $s_1s_2s_2$ or $s_2s_1s_2$.

4 Simulation

To simulate a circuit having binary network $N = \langle \{0,1\}, \mathcal{X}, \mathcal{S}, \mathcal{E} \rangle$, we use the *transient network* $\mathbf{N} = \langle \mathbf{T}, \mathcal{X}, \mathcal{S}, \mathcal{E} \rangle$. The inputs and state variables of N and $\mathbf{N}$ are the same, but in $\mathbf{N}$ they take values in the domain $\mathbf{T}$, and excitations in $\mathbf{N}$ are the extensions to $\mathbf{T}$ of the Boolean excitations in N. Binary variables, words, tuples and excitations in N are denoted by italic characters (*e.g.*, s, S). Transients, tuples of transients, and excitations in $\mathbf{N}$ are denoted by boldface characters (*e.g.*, $\mathbf{s}$, $\mathbf{S}$).

The simulation consists of Algorithm $\tilde{\mathrm{A}}$ [1] given below. We want to know what happens when the network starts in a stable binary initial state $\tilde{a} \cdot b$, and the input is changed to a. We set the input of network $\mathbf{N}$ to $\tilde{a} \circ a$, where $\circ$ is applied componentwise. We then change all variable values to the values of their excitations. For feedback-free circuits $\tilde{\mathrm{A}}$ always terminates. Let the sequence of states produced by $\tilde{\mathrm{A}}$ be $\mathbf{s}^0, \dots, \mathbf{s}^H$. This sequence is nondecreasing in the prefix order on $\mathbf{T}$; we say that $\tilde{\mathrm{A}}$ is *monotonic*.

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

$\mathbf{X}_1$	$\mathbf{s}_1$	$\mathbf{s}_2$	$\mathbf{s}_3$	$\mathbf{s}_4$	$\mathbf{s}_5$	$\mathbf{s}_6$	$\mathbf{s}_7$	$\mathbf{s}_8$	$\mathbf{s}_9$	state
01	0	0	0	0	0	1	1	0	1	$\mathbf{s}^0$
01	01	0	0	0	0	1	1	0	1	$\mathbf{s}^1$
01	01	01	0	0	0	1	1	0	1	$\mathbf{s}^2$
01	01	01	01	01	0	1	1	0	1	$\mathbf{s}^3$
01	01	01	01	01	01	1	1	01	1	$\mathbf{s}^4$
01	01	01	01	01	01	10	1	01	1	$\mathbf{s}^5$
01	01	01	01	01	01	10	10	01	1	$\mathbf{s}^6$
01	01	01	01	01	01	10	10	01	101	$\mathbf{s}^7$

Example 4. For the circuit of Fig. 1 (right), the extended excitations are: $\mathbf{S}_1 = \mathbf{X}_1,\ \mathbf{S}_2 = \mathbf{s}_1,\ \mathbf{S}_3 = \mathbf{S}_4 = \mathbf{s}_2,\ \mathbf{S}_5 = \mathbf{s}_3, \mathbf{S}_6 = \overline{\mathbf{s}_5},\ \mathbf{S}_7 = \mathbf{s}_6,\ \mathbf{S}_8 = \mathbf{s}_4,\ \mathbf{S}_9 = \mathbf{s}_7 \oplus \mathbf{s}_8$. Input $\mathbf{X}_1$ changes from $\tilde{a} = 0$ to $a = 1$, and $b = 000001101$. The result is in the table above.

Theorem 1. *Let $\tilde{\mathbf{N}} = \langle \mathbf{T}, \mathcal{X}, \tilde{\mathcal{S}}, \tilde{\mathcal{E}} \rangle$ be the complete version of a feedback-free network $\mathbf{N} = \langle \mathbf{T}, \mathcal{X}, \mathcal{S}, \mathcal{E} \rangle$. Let $\mathbf{s}^H$ be the result of Algorithm $\tilde{A}$ for $\tilde{\mathbf{N}}$ started in stable binary total state $\tilde{a} \cdot \tilde{b}$, with input $\tilde{a} \circ a$. Let $\mathbf{s}^G$ be the result of Algorithm $\tilde{A}$ for $\mathbf{N}$, started in stable binary total state $\tilde{a} \cdot b$, where $b_i = \tilde{b}_i$, for all $s_i \in \mathcal{S}$, with the same input. Then $\mathbf{s}^H$ and $\mathbf{s}^G$ agree on the variables in $\mathcal{S}$.*

The proof is given in [5]. The theorem shows that adding wire delays to $\mathbf{N}$ does not affect the number of signal changes in $\mathbf{N}$. In other words, simulation takes wire delays into account automatically.

5 Covering of Simulation by Binary Analysis

Let N be the binary network of a complete feedback-free circuit. We start N in stable total state $\tilde{a} \cdot b$, and change the input tuple $\tilde{a}$ to a. In the resulting state $a \cdot b$ only the input gates corresponding to the inputs that change are unstable, all other state variables being stable. Let $G_a(b)$ be the result of the binary analysis of N with initial state $a \cdot b$. Here, by a *path in $G_a(b)$* we always mean a path starting in state b.

Definition 5. *Let $\pi = s^0, \ldots, s^h$ be any path in $G_a(b)$. Recall that each $s^j \in V_a$ has the form $(s_1^j, \ldots, s_m^j)$. For any $i \in [m]$, we denote by $\boldsymbol{\sigma}_i^\pi$ the transient $\widehat{s_i^0 \ldots s_i^h}$ showing the changes of s_i along π. We call $\boldsymbol{\sigma}_i^\pi$ the* history *of s_i. We also define $\boldsymbol{\Sigma}_i^\pi$ to be $\widehat{E_i}$, where $E_i = S_i(a \cdot s^0) \ldots S_i(a \cdot s^h)$ is the transient showing the changes of S_i along π. We call $\boldsymbol{\Sigma}_i^\pi$ the* excitation history *of s_i. We denote by $\boldsymbol{\sigma}^\pi$ the tuple $(\boldsymbol{\sigma}_1^\pi, \ldots, \boldsymbol{\sigma}_m^\pi)$.*

Let $\mathbf{N}$ be the transient counterpart of N, and let the result of Algorithm $\tilde{\text{A}}$ for $\mathbf{N}$, with initial state $\tilde{a} \cdot b$, and input $\tilde{a} \circ a$ be $\mathbf{s}^0, \ldots, \mathbf{s}^H$. If we find a path π in $G_a(b)$ whose history matches the last state $\mathbf{s}^H$ of the simulation, then π covers all simulation states, due to the monotonicity of Algorithm $\tilde{\text{A}}$. Thus, we are looking for a *matching* path, defined next.

Definition 6. *Let π be a path in $G_a(b)$. Let $V \subseteq \mathcal{S}$ be a set of state variables in N. Path π is* matching on V *if $\boldsymbol{\sigma}_i^\pi = \mathbf{s}_i^H$, for all $s_i \in V$.*

Our main result is stated next. The rest of the paper is devoted to its proof.

Theorem 2. *Binary analysis covers simulation in the following sense. There exists a path π in $G_a(b)$ that is matching on $\mathcal{S}$, i.e., that satisfies $\boldsymbol{\sigma}^\pi = \mathbf{s}^H$.*

Let $G'_a(b)$ be the subgraph of $G_a(b)$ in which exactly one unstable variable changes at each step. For example, $G'_a(b)$ is shown in boldface in Fig. 2. The next proposition [5], stated without proof, allows us to restrict ourselves to $G'_a(b)$. Paths π and π' are *equivalent* if they have the same history, *i.e.*, if $\boldsymbol{\sigma}^\pi = \boldsymbol{\sigma}^{\pi'}$.

Proposition 1. *For any path π in $G_a(b)$ there exists an equivalent path π' in $G'_a(b)$.*

Since the circuit is feedback-free, we can arrange the state variables of N in levels as follows: level 0 consists of the input gates; level 1 is comprised of all variables whose fan-in set belongs to level 0, and in general level l consists of all variables whose fan-in variables belong to levels $< l$, and which have at least one fan-in variable in level $l-1$. This level assignment results in even levels containing gate variables and odd levels containing wire variables.[2] We use $\mathit{level}(s_i)$ to denote the level of s_i. The last level of any network is always a gate level, so N has an even number $2L$ of levels. Let $2l$, where $0 < l \leq L$, be a gate level of the circuit. Let $V_{2l} = \{s_i \in \mathcal{S} \mid \mathit{level}(s_i) = 2l\}$. Let $V = \bigcup_{s_i \in V_{2l}} \phi(s_i)$. Suppose $V = \{s_1, \ldots, s_K\}$. Note that $s_1, \ldots, s_K$ are all wire variables and are initially stable; they are not necessarily all of level $2l-1$, but they belong to levels $< 2l$. The circuit is partitioned by these variables into two areas, denoted $\mathcal{L}_{2l}$ and $\mathcal{R}_{2l}$. Area $\mathcal{L}_{2l}$ contains the gates of levels $< 2l$ together with their fan-in variables. Area $\mathcal{R}_{2l}$ contains the gates of level $2l$, and the gates of levels $> 2l$ together with their fan-in variables. Since the circuit is feedback-free, there are no signals flowing from $\mathcal{R}_{2l}$ to $\mathcal{L}_{2l}$, but there may be wires that connect outputs of gates in $\mathcal{L}_{2l}$ to inputs of gates of levels $> 2l$ in $\mathcal{R}_{2l}$. Formally, $\mathcal{L}_{2l} = \bigcup_{\mathit{level}(s_i) < 2l,\ \mathit{level}(s_i)\ \text{even}} (\{s_i\} \cup \phi(s_i))$, and $\mathcal{R}_{2l} = \bigcup_{\mathit{level}(s_i) = 2l} \{s_i\} \cup \bigcup_{\mathit{level}(s_i) > 2l,\ \mathit{level}(s_i)\ \text{even}} (\{s_i\} \cup \phi(s_i))$. Note that $\mathcal{L}_{2l}, V, \mathcal{R}_{2l}$ form a partition of

[2] For this reason we consider input delays and forks as gates.

the set $\mathcal{S}$ of state variables in N, $\mathcal{L}_2 = \bigcup_{level(s_i)=0} \{s_i\}$, $\mathcal{R}_{2L+2} = \emptyset$ and $\mathcal{L}_{2L+2} = \mathcal{S}$, if we assume a fictitious gate level $2L+2$.

Example 5. We illustrate the partition in Fig. 4. Here $l = 3$. We relabel the state variables in $\mathcal{L}_{2l}$ with subscripted λs, and those in $\mathcal{R}_{2l}$, with subscripted ρs.

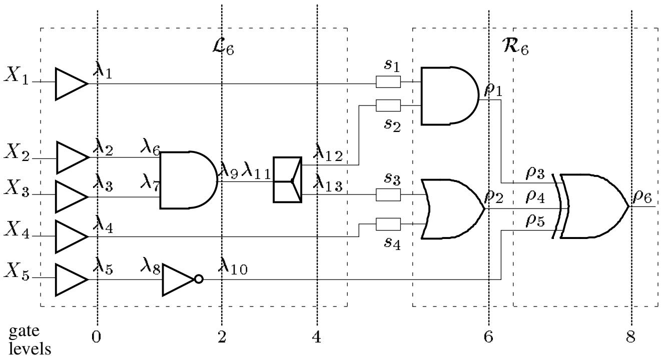

Fig. 4. Sample circuit with partition.

5.1 Proof of Theorem 2

The proof of Theorem 2 is by induction on l, where $1 \leq l \leq L$. We give only a sketch of the proof here (see [5] for details). We show there exists a path in $G'_a(b)$ that is matching on $\mathcal{L}_{2L+2} = \mathcal{S}$. The basis consists of showing there exists a path that is matching on $\mathcal{L}_2$, *i.e.*, on the input-gate variables. In the induction step we assume we have a path τ that is matching on $\mathcal{L}_{2l}$, and construct a path π that is matching on $\mathcal{L}_{2l+2}$. A preliminary result characterizes π in terms of the *hazard-preserving* and *worst-case* properties defined next.

Definition 7. *Let π be a path in $G_a(b)$, and s_i a state variable that is initially stable. We call π* hazard-preserving *on s_i if $\boldsymbol{\sigma}_i^\pi = \boldsymbol{\Sigma}_i^\pi$. For $V \subseteq \mathcal{S}$, path π is* hazard-preserving *on V if it is hazard-preserving on all $s_i \in V$.*

Definition 8. *Let π be a path in $G_a(b)$, and s_i a gate variable implementing Boolean function $f : \{0,1\}^k \longrightarrow \{0,1\}$ that depends only on state variables. Let $\phi(s_i) = \{s_1, \ldots, s_k\}$. We call π* worst-case *on s_i if $\boldsymbol{\Sigma}_i^\pi = \mathbf{f}(\boldsymbol{\sigma}_1^\pi, \ldots, \boldsymbol{\sigma}_k^\pi)$. For a set of gate variables $V \subseteq \mathcal{S}$, path π is* worst-case *on V if it is worst-case on each $s_i \in V$.*

Example 6. Consider the graph in Fig. 2. Path $\pi = 10101, 00101, 00001, 01001, 01011$ is hazard-preserving on s_3 and s_4, but not on s_5, since $\boldsymbol{\sigma}_5^\pi = 1$ and $\boldsymbol{\Sigma}_5^\pi = 101$. Path π is worst-case on s_5, but $\pi' = 10101, 11101, 11111, 01111, 01011$ is not, since $\boldsymbol{\Sigma}_5^{\pi'} = 1$ and $10 \oplus 01 = 101$.

Lemma 1. *Path π in $G'_a(b)$ is matching on $\mathcal{L}_{2l+2}$ iff it is matching on $\mathcal{L}_{2l}$, hazard-preserving on V and V_{2l}, and worst-case on V_{2l}.*

5.2 Hazard-Preserving Paths

We now characterize hazard-preserving paths by automata. For $V = \{s_1, \ldots, s_k\} \subseteq \mathcal{S}$, let $\Xi(V) = \{S_i \mid s_i \in V\}$, and $\Delta(V) = V \cup \Xi(V)$. Suppose the variables in V are initially stable in $G'_a(b)$, unrelated to each other, and have pairwise distinct excitations. Recall that every state variable represents a delay. We want to describe the hazard-preserving behavior of these delays in $G'_a(b)$, *i.e.*, we are interested in paths on which the k delays do not 'lose' any changes. We describe the hazard-preserving behavior of $s_i \in V$ by the automaton shown in Fig. 5 (left); this is $\mathcal{D}^i_V$, the *delay automaton for variable* s_i. The label on a transition of the automaton shows the excitation or variable that changes in that transition. Subscript j ranges over $[k] \setminus i$. The label of each state shows whether s_i is stable (label is $\emptyset$) or unstable (label is $\{s_i\}$) in that state. Changes of excitations or variables other than S_i and s_i do not alter the state, since variables in V are unrelated and have distinct excitations. Variable s_i changes each time it is unstable, so as not to lose any changes.

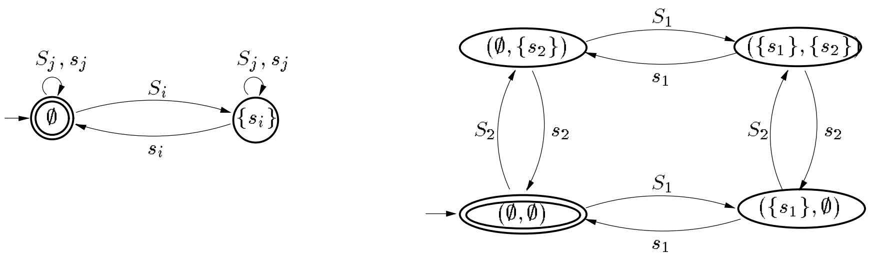

Fig. 5. Delay automata for variable s_i, and set $\{s_1, s_2\}$

To describe the hazard-preserving behavior of all $s_i \in V$ at the same time we take the *direct product* [4] $\mathcal{D}_V$ of $\mathcal{D}^1_V, \ldots \mathcal{D}^k_V$.

Example 7. The delay automaton of set $V = \{s_1, s_2\}$ is shown in Fig. 5 (right). The nonempty components of a state label show the variables that are unstable.

Let $\mathcal{L}(\mathcal{D}_V)$ be the language accepted by $\mathcal{D}_V$. We call the words in $\mathcal{L}(\mathcal{D}_V)$ *balanced on* V. By the definition of the direct product, we have $\mathcal{L}(\mathcal{D}_V) = \mathcal{L}(\mathcal{D}^1_V) \cap \ldots \cap \mathcal{L}(\mathcal{D}^k_V)$. From the definition of each $\mathcal{D}^i_V$, it follows that $w \in \Delta^*_V$ belongs to $\mathcal{L}(\mathcal{D}^i_V)$ iff $w{\downarrow}_{\{S_i, s_i\}} = (S_i s_i)^{c_i}$, for some integer $c_i \geq 0$, where $w{\downarrow}_{\mathcal{A}}$ is the projection of w to alphabet $\mathcal{A}$. Then a word $w \in \Delta^*_V$ is balanced on V iff, for all $s_i \in V$, $w{\downarrow}_{\{S_i, s_i\}} = (S_i s_i)^{c_i}$, for some integer $c_i \geq 0$. Language $\mathcal{L}(\mathcal{D}_V)$ is a regular subset of the *Dyck language* D_k [6].

We now establish the relation between hazard-preserving paths and delay automata. Let V be defined as before. We limit our interest to paths that are hazard-preserving on V.

To any path π in $G'_a(b)$ we associate $w^\pi \in \Delta^*_V$ called the *path-word on* V as follows: we label each step of the path with S_i if S_i changes, and with s_i if s_i changes in that step, for $S_i \in \Xi(V)$, $s_i \in V$. Other steps are labelled by ϵ. Since the variables of V are unrelated and have distinct excitations, each step has a single label. Path-word w^π is the concatenation of the labels along path π.

Example 8. Consider $G_a(b)$ of Example 1. The subgraph $G'_a(b)$ is shown in Fig. 2 by boldface edges. We choose $V = \{s_3, s_4\}$ and show the labels on edges. The values of s_3, s_4 are also in boldface in each state. For $\pi = 10101, 00101, 01101, 01111, 01011$, $w^\pi = S_3S_4s_4s_3$.

We denote by $\mathcal{H}_V$ the set of all paths in $G'_a(b)$ that are hazard-preserving on V. Let $\mathcal{W}_V = \{w^\pi \mid \pi \in \mathcal{H}_V\}$. The delay automaton is quite general, and applies to any network N and any $G'_a(b)$, as long as we find a set V that satisfies the necessary requirements. For a particular network N and $G'_a(b)$, not all words accepted by the automaton correspond to paths in $G'_a(b)$. We find a necessary and sufficient condition for a balanced word to be a path-word.

Definition 9. *A word $w \in \mathcal{L}(\mathcal{D}_V)$ is* relevant *to $G'_a(b)$ iff there exists a path π in $G'_a(b)$ such that $w{\downarrow}_{\Xi(V)} = w^\pi{\downarrow}_{\Xi(V)}$. We denote by $\mathcal{L}(\mathcal{D}_V){\downarrow}_{G'_a(b)}$ the set of all words in $\mathcal{L}(\mathcal{D}_V)$ that are relevant to $G'_a(b)$.*

Example 9. For the circuit and $G'_a(b)$ of Example 8, with $V = \{s_3, s_4\}$, the delay automaton $\mathcal{D}_V$ is in Fig. 5 (right), with S_3, s_3, S_4, s_4 taking the roles of S_1, s_1, S_2, s_2, respectively. For example, $S_4S_3s_4s_3$ and $S_3s_3S_4s_4$ are relevant, $S_3s_3S_3S_4s_3s_4$ and $S_3S_4s_4S_4s_3s_4$ are irrelevant.

We state without proof (see [5] for details) the following proposition that reduces finding a hazard-preserving path to finding a relevant balanced word.

Proposition 2. $\mathcal{W}_V = \mathcal{L}(\mathcal{D}_V){\downarrow}_{G'_a(b)}$.

5.3 Worst-Case Paths

We now characterize worst-case paths using gate automata. Let s_i be any gate variable in $\mathcal{S}$, with $\phi(s_i) = \{s_1, \ldots, s_k\}$, and let $(\mathcal{G}_i, p_i^0)$ be its gate automaton, with $p_i^0 = (b_1, \ldots, b_k)$ (the values of $s_1, \ldots, s_k$ in b). For any path π in $G'_a(b)$ we label with s_j each step in which s_j changes, for all $j \in [k]$. The word obtained by concatenating the labels along π is an input word $u^\pi \in I_i^*$ that shows how the fan-in variables of s_i change along π. The output word v^π produced by u^π shows how the excitation S_i changes on π. Let $\mathbf{t}_1, \ldots, \mathbf{t}_k$ be the transients determined by u^π. Then the following hold: 1) $\boldsymbol{\sigma}_j^\pi = \mathbf{t}_j$, for all $j \in [k]$, and 2) $\boldsymbol{\Sigma}_i^\pi = \widehat{v^\pi}$. Let π be a path in $G'_a(b)$ labelled with u^π as above.

Proposition 3. *Path π is worst-case on s_i iff u^π is a worst-case word for $(\mathcal{G}_i, p_i^0)$.*

5.4 Delay Automata and Gate Automata

Having reduced finding a hazard-preserving path to finding a relevant balanced word, and finding a worst-case path to finding a worst-case word, we now state an important lemma that relates these two kinds of words, and guarantees the existence of a path that is both hazard-preserving and worst-case. For any gate variable s_i of a feedback-free circuit, we have a delay automaton $\mathcal{D}_{\phi(s_i)}$ for its fan-in set, and a gate automaton $(\mathcal{G}_i, p_i^0)$.

For any alphabet $\mathcal{A}$, a word $r \in \mathcal{A}^*$ is called *prefix-restricted* if r has a prefix r' having exactly one occurrence of each letter of r. We call r' the *key prefix* of r. For example, word $abaabb$ is prefix-restricted, with key prefix ab, but $aabab$ is not prefix-restricted.

Recall that $I_i = \phi(s_i)$, $\Xi(I_i)$ is the set of the excitations of the variables in I_i, and $\Delta_{I_i} = I_i \cup \Xi(I_i)$ is the alphabet of $\mathcal{D}_{I_i}$. The lemma below relates words over $\Xi(I_i)$ to words over I_i in the following sense. Given any prefix-restricted word over $\Xi(I_i)$, we can always find a worst-case word over I_i, such that an interleaving of the two words is a balanced word. The result is limited to 1- and 2-input gates; we conjecture the result to be true in the general case.

Lemma 2. *Let $(\mathcal{G}_i, p_i^0)$ be the gate automaton of variable s_i, for a 1- or 2-input gate. For any prefix-restricted word $r \in \Xi(I_i)^*$ having key prefix r', there exists a balanced word $w \in \mathcal{L}(\mathcal{D}_{I_i})$ such that $w{\downarrow}_{\Xi(I_i)} = r$, and $w{\downarrow}_{I_i}$ is a worst-case word for $(\mathcal{G}_i, p_i^0)$. Also, w has a prefix w' such that $w'{\downarrow}_{\Xi(I_i)} = r'$, and the output profile of $w'{\downarrow}_{I_i}$ has length 1 if the output profile of $w{\downarrow}_{I_i}$ has length ≥ 1.*

Proposition 2 and Lemma 2 help us construct a path π satisfying the conditions of Lemma 1, and hence conclude the proof of Theorem 2. For details see [5].

6 Conclusions

Our results can be summarized as follows. Assume that we have a feedback-free gate network N, in which state variables are associated with gates only. We perform the binary analysis of N started in a stable state. We extend the Boolean functions in N to functions on transients. This gives us the transient version $\mathbf{N}$ of N. We now simulate $\mathbf{N}$ using algorithm $\tilde{\mathrm{A}}$; for feedback-free circuits this algorithm always terminates. Next, we add wire delays to N, obtaining the complete binary network $\tilde{N}$ and its transient counterpart $\tilde{\mathbf{N}}$. By Theorem 1, the simulation of $\tilde{\mathbf{N}}$ agrees with the simulation of $\mathbf{N}$ on the variables of $\mathbf{N}$. Finally, by Theorem 2, we know that binary analysis of $\tilde{N}$ covers the simulation of $\tilde{\mathbf{N}}$, and hence that of $\mathbf{N}$. In conclusion, we have shown that simulation of feedback-free circuits is not pessimistic, if wire delays are taken into account.

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

References

1. Brzozowski, J. A., Ésik, Z.: Hazard algebras. Maveric Res. Report 00–2, University of Waterloo, ON, Canada, http://maveric.uwaterloo.ca (2000; revised 2001)
2. Gheorghiu, M., Brzozowski, J. A.: Simulation of gate circuits in the algebra of transients. Maveric Res. Report 02–2, University of Waterloo, ON, Canada http://maveric.uwaterloo.ca (2002)
3. Brzozowski, J. A., Seger, C.-J. H.: Asynchronous circuits. Springer-Verlag (1995)
4. Eilenberg, S.: Automata, languages and machines, Academic Press, **A** (1974)
5. Gheorghiu, M.: Circuit simulation using a hazard algebra. MMath Thesis, Department of Computer Science, University of Waterloo, Waterloo, ON, Canada (2001)
6. Salomaa, A.: Formal languages, Academic Press (1973)