A Unified Theory of Asynchronous Networks†

J.A. Brzozowski and C-J. Seger

Department of Computer Science
University of Waterloo
Waterloo, Ontario
Canada N2L 3G1

ABSTRACT

A unified framework is developed for the study of asynchronous circuits of both gate and MOS type. A basic network model consisting of a directed graph and a set of vertex excitation functions is introduced. A race analysis model, using three values (0, 1, and ×), is developed for studying state transitions in the network. It is shown that the results obtained using this model are equivalent to those using ternary simulation. It is also proved that the set of state variables can be reduced to a minimal set of feedback variables, and the analysis still yields both the correct state transitions and output hazard information. Finally, it is shown how the general results above are applicable to both gate and MOS circuits.

1. Introduction

The theory of asynchronous networks had its beginning in the 1950's with the work of Huffman [12] and Muller and Bartky [16, 17]. Huffman used what we will call the *feedback-delay* model where a set of feedback variables represents the state of a circuit. These variables correspond to a set of wires with the property that cutting them would break all the loops in the circuit. With this set of state variables Huffman used a binary race model to analyze state transitions. Muller and Bartky used what we will call the *gate-delay* model, where the outputs of all the gates constitute the state variables, and a binary model to analyze races. Huffman's race analysis is rather informal, whereas Muller's is formally defined. Both models use what has been later called the *general multiple winner* (GMW) model, in the sense that the delays are arbitrary and any nonempty subset of the set of unstable gates is allowed to change. Usually, the minimum number of feedback variables is much smaller than the number of gates; consequently, the feedback variable model has been, and continues to be, quite widely used.

† This research was supported by the Natural Sciences and Engineering Research Council of Canada under grant A0871 and by an Institute for Computer Research Scholarship.

While the feedback-delay model gives the correct stable states, the state transitions predicted by this model are not always correct. These observations were made quite early [12, 15], and various types of hazards were then used in an attempt to explain these discrepancies. For a detailed discussion of this approach the reader is referred to the book by Unger [19]. Roughly speaking, one first obtains a flow table using the feedback variables. The stable entries in this table are correct, but one has to perform a series of corrections to the unstable entries if certain hazards are present — not a very attractive theory. An even stronger reason for not using such a theory is Langdon's example [13], which demonstrates that different sets of feedback variables may lead to different flow tables, and these differences cannot be accounted for by any known hazards. Thus the feedback variable approach does not quite fit.

In view of these difficulties with the feedback variable approach, a return to Muller's gate-delay model, using gate outputs as state variables, seemed necessary. This approach was advocated, for example, by Langdon [13], but still with binary race models. Precise mathematical formulations of such race models were developed by Brzozowski and Yoeli [6, 7]. In particular, they formalized the GMW race model. We will return to binary race models shortly.

In the mid-1960's ternary models were introduced for the analysis of races and hazards in asynchronous gate circuits [10, 20]. In particular, Eichelberger proposed a ternary simulation of a circuit using a third value, $\times$, denoting an intermediate or unknown signal. His method can be used to predict state transitions and detect static hazards. Since the algorithm is quite efficient, it has been widely used. Unfortunately, some discrepancies were noted between the results predicted by ternary simulation and those predicted by the binary analysis [1, 7]. Again it seemed that the theories do not quite fit. A detailed discussion of these problems was presented by Brzozowski and Yoeli [7], who also conjectured that the theories would fit if one used not only gate outputs but also wire signals as state variables — the *gate- and wire-delay* model. This conjecture was finally proved by Brzozowski and Seger [8]. Thus a fit has been found between ternary simulation of a network using the gate-delay model and the binary GMW analysis of the gate- and wire-delay model. The proof of this result is quite involved.

In the last decade the digital circuit technology has undergone tremendous changes and MOS circuits have become widespread as a result of the VLSI revolution. In some ways these circuits resemble relay contact circuits more than gate circuits, and it has been recognized that the theory based on gates is no longer adequate for some aspect of MOS circuits. To remedy this, switch-level models like that of Bryant [3] have been developed. In the MOS technology it appears necessary to introduce an intermediate $\times$ value in order to properly describe states in which the output is neither 0 nor 1. Bryant [2] adapted Eichelberger's ternary simulation to switch-level MOS circuits, but justified the use of this technique only

by examples. Lengauer and Näher [14] defined a sort of race analysis model for MOS circuits which uses the three values 0, 1, and $\times$, and proved that this race model corresponds exactly to ternary simulation, thus providing a mathematical justification for its use. At this point it appeared that the theory of MOS circuits was diverging from that of gate circuits, and that the latter would cease to have much significance.

In this paper we develop a theory where everything fits. First, we abstract the properties of both gate and MOS circuits, and each such circuit is represented as a mathematical asynchronous network consisting of a directed graph with an associated set of *vertex excitation functions*. Second, we define a new race model, called the *extended multiple winner* (XMW) model, which incorporates the multiple-winner aspect of the GMW model and the third $\times$-value used by Bryant, and Lengauer and Näher. Third, we show that the use of feedback variables is completely correct in this race model, not only for predicting state transitions, but also for static output hazards. Fourth, we prove that the results of the XMW analysis of an asynchronous network correspond exactly to those of the ternary simulation of the network. This proof is quite natural and substantially simpler than that relating ternary simulation to the GMW analysis.

The results have the following applications to gate circuits. Let N be the gate-delay model of a circuit, let $\dot{N}$ be the feedback-delay model, and $\tilde{N}$ the gate- and wire-delay model. The following theorem, proved in Section 7, summarizes the theory:

Theorem 5 The following analysis techniques are all equivalent for gate circuits from the point of view of nontransient state behavior and static output hazards:

1. GMW analysis of $\tilde{N}$.
2. XMW analysis of (a) $\tilde{N}$, (b) N, and (c) $\dot{N}$.
3. Ternary simulation of (a) $\tilde{N}$, (b) N, and (c) $\dot{N}$.

For MOS circuits we use a *node-delay* model, i.e. we associate state variables with all internal nodes. We present several alternatives for calculating node excitations; these representing different design philosophies for CMOS and NMOS circuits. Let N be such a model; let $\dot{N}$ be the same model reduced to feedback variables; and let $\tilde{N}$ be the *node- and transistor-delay* model where state variables are associated with both nodes and transistors. Our results are summarized in the following theorem proved in Section 9:

Theorem 6 The following analysis techniques are all equivalent for switch-level circuits, using any one of the node excitation models described in Section 8, from the point of view of nontransient state behavior and static output hazards:

1. XMW analysis of: (a) $\tilde{N}$, (b) N, and (c) $\dot{N}$.
2. Ternary simulation of: (a) $\tilde{N}$, (b) N, and (c) $\dot{N}$.

Although this work settles many previously open problems, further work remains to be done. One serious difficulty with both the XMW model and its efficient ternary simulation is that the model permits arbitrary delays and is consequently overly "pessimistic", rejecting many designs which are acceptable in practice. Thus more realistic race models need to be developed. The results of the present paper constitute a consistent framework in which such problems can be studied.

The paper is structured as follows. Section 2 contains the definition of the graph-theoretic model of asynchronous networks and Section 3 describes the XMW race model for these networks. Section 4 contains the description of the ternary simulation algorithm for our network model and Theorems 1 and 2 which establish the correspondence between the basic XMW analysis and ternary simulation. The main result of Section 5 is Theorem 3 which establishes the correctness of the feedback-delay approach. Output hazards are considered in Section 6, where it is summarized in Theorem 4 how static hazards are detected by ternary simulation. The general unified theory of Sections 2-6 is specialized to gate circuits in Section 7. Section 8 contains a description of switch-level models and a detailed discussion of various options available for the definition of excitation functions for MOS circuits. In Section 9 we specialize the results of Sections 2-6 to MOS circuits. Finally, Section 10 concludes with a brief discussion of further work required.

2. Asynchronous Networks

A rather general concept of a network is introduced in this section. As will be shown later, this model provides a common framework for representing both gate networks and switch-level MOS networks.

A network N is a finite directed labeled graph $N = <V, E, x, y, Y>$, where
$V = \{1, \cdots, m\}$ is a set of vertices,
$E \subseteq V \times V$ is a set of edges,
$x = x_1, \cdots, x_n$, $n \leq m$, is a vector of *input variables* taking values from a set T,
$y = y_1, \cdots, y_m$ is a vector of *vertex variables* taking values from T,
$Y = Y_1, \cdots, Y_m$ is a vector of *excitation functions*.
Vertices $1, \cdots, n$ are all of indegree 0, and are called *input vertices*. Vertices $n+1, \cdots, m$ are *function vertices* and are all of indegree ≥ 1. The excitation function of a vertex i is a function $Y_i : T^{n+m} \rightarrow T$. The vertex variable y_i is interpreted as the present state of a vertex, whereas the excitation function $Y_i(x, y)$ computes the value to which the vertex is trying to change, when the present input is x and the present state is y. The vector x, y is called the *total state* of N.

While it is convenient to write $Y_i(x,y)$ for the excitation of a general vertex i, the functions are somewhat simpler in reality. In fact, for an input vertex i, the excitation function is simply $Y_i = x_i$. For function vertices, an edge $(i,j) \in E$ shows that Y_j is a function of y_i. Thus, for a function vertex, Y_j depends only on some subset of $\{y_1, \cdots, y_m\}$.

In the examples throughout the paper we use three ternary functions over the set $T = \{0, \times, 1\}$, as defined in Fig. 1. In fact, these functions are natural ternary extensions of their Boolean counterparts: OR, AND, and complement. More will be said about this later.

+	0	×	1
0	0	×	1
×	×	×	1
1	1	1	1

·	0	×	1
0	0	0	0
×	0	×	×
1	0	×	1

a	0	×	1
a'	1	×	0

Figure 1. Ternary OR, AND, and complement.

To illustrate the definition of a network consider Fig. 2. The excitation functions are:

$$Y_1 = x_1 \quad Y_2 = x_2 \quad Y_3 = (y_1 + y_4)' \quad Y_4 = (y_2 + y_3)'.$$

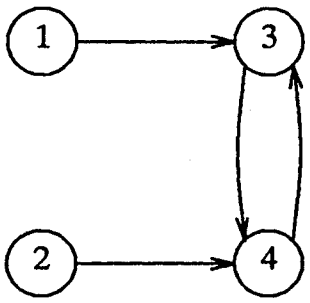


Figure 2. Network N_1.

If $y_i = Y_i(x,y)$ then vertex i is *stable*; otherwise it is *unstable*. A given total state x,y is *stable* if each vertex is stable. A network will remain in a stable total state indefinitely, unless the input changes, in which case the state becomes unstable. In general, there may be several possible successor states for a given unstable state. This set of possible successor states depends on the transition model used, as elaborated below. Any situation in which two or more vertices are unstable is called a *race*.

Given a fixed input vector $a \in T^n$ and an initial state $b^0 \in T^m$, a binary relation R on T^m is a *transition relation* iff

(i) if $b^0 R^* b$, then $b R c$ for some $c \in T^m$,
(Each state reachable from b^0 has at least one successor.)

(ii) if a, b is a stable total state then $b R b$,
(Every stable state has itself as a successor.)

(iii) if $b R c$ and $b_i = Y_i(a,b)$, then $c_i = b_i$.
(If a vertex i is stable then it cannot change. Note that if all the vertices are stable, i.e. if a, b is stable, then the only successor of b is b itself.)

In (i) above R^* denotes the reflexive and transitive closure of R. A detailed example of a transition relation is given in the next section.

3. The XMW Transition Relation

The main transition relation used in this paper is the Extended Multiple Winner (XMW) relation described below. Let $B = \{0,1\}$ and $T = \{0,1,\times\}$. Define the partial order $\leq$ on T as follows: $t_i \leq t_i$ for all $t_i \in T$, $0 \leq \times$, and $1 \leq \times$. The partial order is extended to T^m in the obvious way: $s \leq t$ iff $s_i \leq t_i$ for $1 \leq i \leq m$. We write $s < t$ when $s \leq t$ and $s \neq t$.

The value $\times$ is used to denote an unknown or intermediate value. Thus $s < t$ indicates that s has less uncertainty (more binary values) than t. The following fundamental assumption is made about the excitation function of any network N:

$$a, b \leq c, d \quad \text{implies} \quad Y(a,b) \leq Y(c,d).$$

This is a monotonicity property that is consistent with our use of the value $\times$. Basically, if the total state is more uncertain, the excitation cannot become less uncertain.

For any $a \in T^n$ and $b \in T^m$, define $U(a,b)$ to be the set of unstable vertices in b, i.e.

$$U(a,b) = \{i : 1 \leq i \leq m, \text{ and } b_i \neq Y_i(a,b)\}.$$

The XMW relation R_a on the set T^m defines the set of successors for any total state $b \in T^m$ as follows. If b is stable, i.e. if $U(a,b) = \emptyset$, then the only possible successor is b, i.e. $b R_a b$. Otherwise, let S be any nonempty subset of the set $U(a,b)$ of unstable gate variables. Further, let S_1 be an arbitrary subset of the set $\{i \in S : b_i \in B \text{ and } Y_i(a,b) = b_i'\}$. Define $b^{(S,S_1)}$ as follows:

$$\begin{aligned} b_i^{(S,S_1)} &= \times, && \text{if } i \in S_1 \\ b_i^{(S,S_1)} &= Y_i(a,b), && \text{if } i \in S - S_1 \\ b_i^{(S,S_1)} &= b_i, && \text{if } i \notin S. \end{aligned}$$

Now every state of the form $b^{(S,S_1)}$ is an immediate successor of b, i.e. $b R_a b^{(S,S_1)}$.

No other pairs are related by R_a. The reader can easily verify that R_a is a transition relation.

There are three basic ideas behind the XMW model. First, it is assumed that the input remains fixed after each change until the network has a chance to "stabilize". This corresponds to the fundamental-mode operation assumption of [15]. Second, the past history is completely ignored in the sense that all unstable vertices have the same chance of "winning" a race no matter when they entered the race. Third, any unstable vertex with a binary present value may take on the intermediate value $\times$.

The XMW analysis of the network N_1 of Fig. 2 is shown in Fig. 3. It is assumed that the network is started in the stable state $\hat{a}, b = 00, 0010$ and that the input changes to $a = 10$. Only those states that are reachable from 0010 are shown. Unstable states are shown subscripted; the subscript denotes the value of the excitation function for that vertex. For example, 0_1010 indicate that vertices 2, 3 and 4 are stable and that vertex 1 is unstable with the excitation 1.

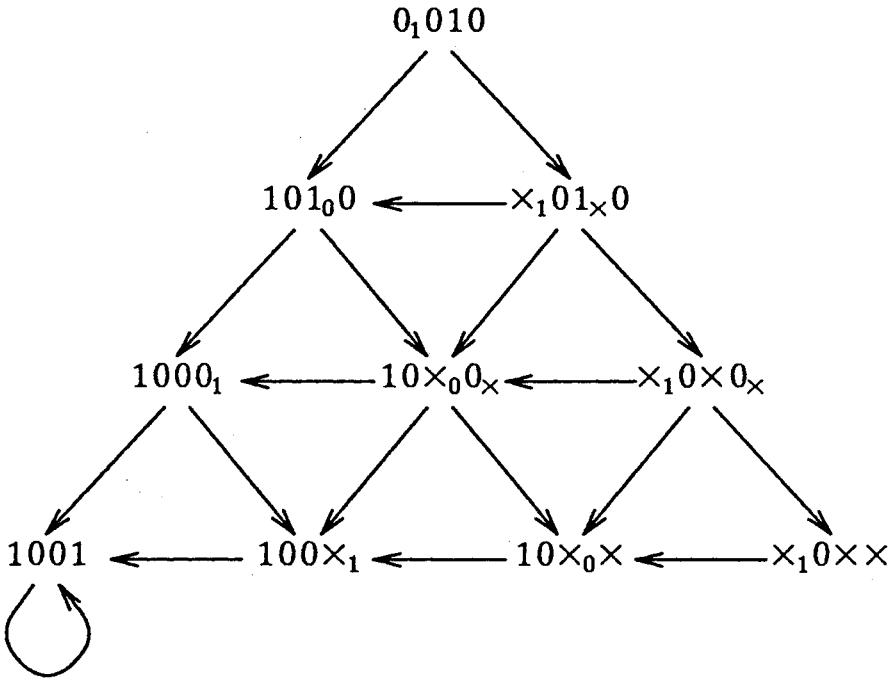


Figure 3. XMW analysis of network N_1.

The following proposition about the XMW relation is used later and can be easily verified.

Proposition 1 If $b, c \in T^m$ and $b\,R_a\,c$, then $c \leq l.u.b.(b, Y(a,b))$.

For any input vector $a \in T^n$ and any state $b \in T^m$, define the set cycl(R_a, b) to be the set of total states of N that appear in cycles in the relation R_a and are reachable from b. Note that each stable state reachable from b is in cycl(R_a, b). Formally,

$$\text{cycl}(R_a,b) = \{c \in T^m : b\, R_a^* c \text{ and } c\, R_a^+ c\},$$

where R_a^+ is the transitive closure of R_a. The notation R_a^h (used later) denotes the composition of h copies of R_a for $h \geq 1$, and R_a^0 is the identity relation.

The concept of a transient cycle is introduced in order to capture the fact that delays cannot be infinite. This concept is very similar to the definition of a transient cycle in the GMW model [7] (which will also be considered later), except for one important difference. The basic idea is to call a cycle transient if there is some vertex that is unstable in all the states of the cycle and has the same value in all these states. However, this definition is slightly too restricted as the following example shows. Suppose we have a cycle like $0_1 \times_0 \rightarrow 1_0 \times_1 \rightarrow 0_1 \times_0$ etc. Note that the vertex with the value $\times$ is unstable in all the states and has the same value in all the states. However, since the excitation of that vertex oscillates between 0 and 1 it is reasonable to assume that such a cycle can persist indefinitely. Because of this, the definition of a transient cycle in the XMW model is somewhat more complicated than the corresponding idea in the GMW model.

A cycle is called *transient* if there exists a vertex v which is unstable in all of the states in the cycle, has the same value in all these states, and either that value is binary or the excitation of v is the same in all these states. If a cycle is not transient, it is called *nontransient*. Let

$$\text{trans}(R_a,b) = \{c \in \text{cycl}(R_a,b) : c \text{ appears only in transient cycles}\}$$

and

$$\text{out}(R_a,b) = \text{cycl}(R_a,b) - \text{trans}(R_a,b).$$

The set out(R_a, b) is the *outcome* of the XMW analysis of the behavior of N when started in total state b, in the sense that it consists of all the states N can be in, under nontransient conditions.

To illustrate the concepts above consider Fig. 3. There is only one cyclic state, namely 1001. Since this state is stable, the cycle is nontransient. Thus out$(R_{10}, 0010) = \{1001\}$. A more complicated example is provided by the network N_2 specified by the following excitation functions:

$$Y_1 = x_1 \qquad Y_2 = (y_1 + y_3)' \qquad Y_3 = (y_1 + y_2)'.$$

Let $\hat{a}, b = 1, 100$ be the initial stable state and let the new input be $a = 0$. The states reachable from 100 are shown in Fig. 4. There are the following cycles:

$$(000,011),\ (0\times0,01\times),\ (00\times,0\times1),\ (010),\ (0\times\times),\ (001)$$

None of these cycles are transient, and hence the outcome is

$$\mathrm{out}(R_0,100) = \{000,001,010,011,0\times0,00\times,01\times,0\times1,0\times\times\}.$$

The XMW model permits us to predict the outcome of any input change under very general assumptions about delays in a network: In fact each vertex may have an arbitrary finite inertial delay. The model, though conceptually simple and relatively natural, is computationally intractable; in the worst case the graph of the relation R_a may have $O(3^m)$ vertices. Fortunately, there exists an efficient algorithm computing essentially the same information, as described in the next section.

4. Ternary Simulation

A ternary simulation of binary networks has been proposed by Eichelberger [10]. Algorithms A and B described below are an adaptation of his work. Let N be a network, $\hat{a}\in T^n$ be an input vector, and $b\in T^m$ be such that $\hat{a},b$ is a stable total state. Furthermore, let $a\in T^n$ be a new input vector and $\mathbf{a}=l.u.b.(\hat{a},a)$. Algorithm A is defined by:

Algorithm A
h:=0;
$\mathbf{y}^0$:=b;
repeat
  h:=h+1;
  for $i=1$ *to* m
    $\mathbf{y}_i^h = Y_i(\mathbf{a},\mathbf{y}^{h-1})$;
until $\mathbf{y}^h=\mathbf{y}^{h-1}$;

Proposition 2 Algorithm A produces a finite sequence $\mathbf{y}^0,\mathbf{y}^1,\cdots,\mathbf{y}^A$ of states, where $A\le m$, and $\mathbf{y}^h>\mathbf{y}^{h-1}$ for $1\le h\le A$.

Proof: We first prove by induction on h that $\mathbf{y}^h\ge\mathbf{y}^{h-1}$, for $h\ge1$. The basis $h=1$ follows because $b=Y(\hat{a},b)$, by the stability requirement, $\mathbf{a},b\ge\hat{a},b$, by the definition of *l.u.b.*, and $\mathbf{y}^1=Y(\mathbf{a},\mathbf{y}^0)\ge Y(\hat{a},\mathbf{y}^0)=Y(\hat{a},b)=b=\mathbf{y}^0$, by the monotonicity of Y. Assuming that $\mathbf{y}^h\ge\mathbf{y}^{h-1}$, it follows by the monotonicity of Y that $\mathbf{y}^{h+1}=Y(\mathbf{a},\mathbf{y}^h)\ge Y(\mathbf{a},\mathbf{y}^{h-1})=\mathbf{y}^h$, and the induction goes through. In each step either $\mathbf{y}^h>\mathbf{y}^{h-1}$ or the algorithm terminates. At least one new vertex becomes $\times$ if $\mathbf{y}^h>\mathbf{y}^{h-1}$; therefore $A\le m$. □

Algorithm B is defined next:

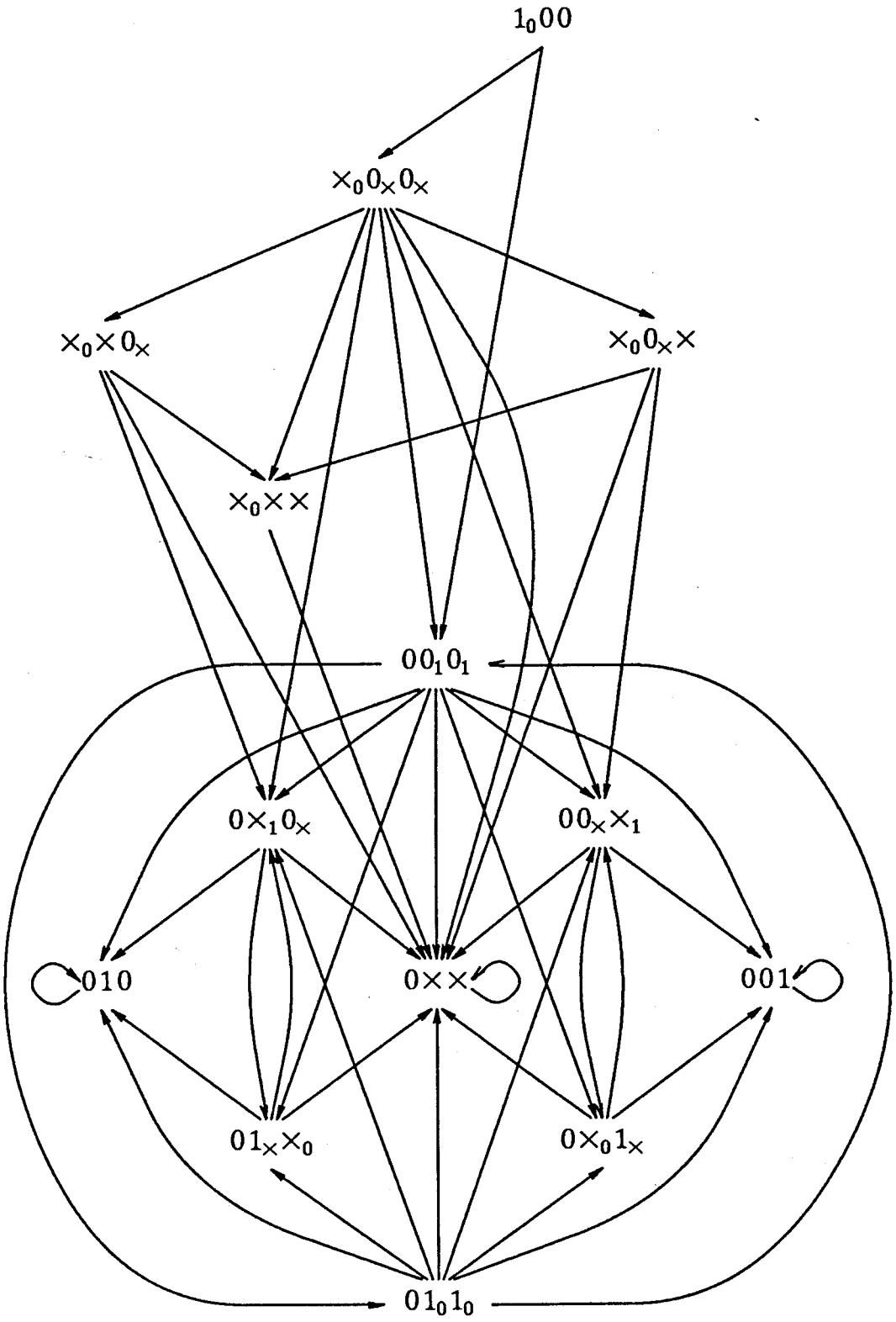


Figure 4. XMW analysis of network N_2.

Algorithm B

h:=0;
$\mathbf{z}^0$:=$\mathbf{y}^A$;
repeat
    h:=h+1;
    for $i = 1$ *to* m
        $\mathbf{z}_i^h = Y_i(a, \mathbf{z}^{h-1})$;
until $\mathbf{z}^h = \mathbf{z}^{h-1}$;

Proposition 3 Algorithm B produces a finite sequence $\mathbf{z}^0, \mathbf{z}^1, \cdots, \mathbf{z}^B$ of states, where $B \leq m$, and $\mathbf{z}^h < \mathbf{z}^{h-1}$ for $1 \leq h \leq B$.

Proof: The proof is dual to the proof of Proposition 2, with $\leq$ replaced by $\geq$, etc. □

The following results are an adaptation of the work of Lengauer and Näher [14]. They used a somewhat different model, but the main ideas are the same.

Theorem 1 The result $\mathbf{y}^A$ of Algorithm A is the least upper bound of all the states reachable from the initial state b in the XMW transition relation, i.e.

$$\mathbf{y}^A = l.u.b.\{c \in T^m : b R_a^* c\}.$$

Proof: Note that $\mathbf{y}^h R_a \mathbf{y}^{h+1}$ for $0 \leq h < A$; hence $b R_a^* \mathbf{y}^A$ and so $\mathbf{y}^A \leq l.u.b.\{c \in T^m : b R_a^* c\}$. To prove the converse, i.e. that $\mathbf{y}^A \geq l.u.b.\{c \in T^m : b R_a^* c\}$, we show the following claim by induction on h: $\mathbf{y}^A \geq l.u.b.\{c \in T^m : b R_a^h c\}$ for all $h \geq 0$.

Basis:

$h = 0$. Trivially true since $\mathbf{y}^A \geq \mathbf{y}^0 = b$.

Induction hypothesis:

Assume that $\mathbf{y}^A \geq l.u.b.\{c \in T^m : b R_a^h c\}$ for some $h \geq 0$.

Induction step:

Assume $c \in T^m$, and $b R_a^{h+1} c$. There must exist a state $d \in T^m$ such that $b R_a^h d$ and $d R_a c$. By the induction hypothesis $\mathbf{y}^A \geq d$. Furthermore, $d R_a c$ implies $c \leq l.u.b.(d, Y(a, d))$, by Proposition 1. Since Y is monotone and $\mathbf{y}^A = Y(\mathbf{a}, \mathbf{y}^A)$, it follows that

$$c \leq l.u.b.(d, Y(a, d)) \leq l.u.b.(d, Y(\mathbf{a}, d)) \leq l.u.b.(\mathbf{y}^A, Y(\mathbf{a}, \mathbf{y}^A)) = \mathbf{y}^A.$$

Thus the induction goes through and the claim follows. □

As above, let $\mathbf{z}^0, \mathbf{z}^1, \cdots, \mathbf{z}^B$ denote the intermediate values produced by Algorithm B.

Theorem 2 The result $\mathbf{z}^B$ of Algorithm B is the least upper bound of all the non-transient cyclic states reachable from b in the XMW transition relation, i.e.

$$\mathbf{z}^B = l.u.b.(\text{out}(R_a, b)).$$

Proof: Since $b = \mathbf{y}^0 R_a^* \mathbf{y}^A = \mathbf{z}^0$, and $\mathbf{z}^0 R_a^* \mathbf{z}^B$, $\mathbf{z}^B$ is reachable from b, i.e. $b R_a^* \mathbf{z}^B$. Also $a, \mathbf{z}^B$ is a stable total state; hence $\mathbf{z}^B \in \text{out}(R_a, b)$ and $\mathbf{z}^B \leq l.u.b.(\text{out}(R_a, b))$. To prove that $\mathbf{z}^B \geq l.u.b.(\text{out}(R_a, b))$ we show that $\mathbf{z}^h \geq l.u.b.(\text{out}(R_a, b))$ for $h \geq 0$, by induction on h.

Basis:

$h = 0$. Since $\mathbf{z}^0 = \mathbf{y}^A$, $\mathbf{y}^A = l.u.b.\{c \in T^m : b R_a^* c\}$ (by Theorem 1), and $\text{out}(R_a, b) \subseteq \{c \in T^m : b R_a^* c\}$, the result follows immediately.

Induction hypothesis:

Assume $\mathbf{z}^h \geq l.u.b.(\text{out}(R_a, b))$ for some $h \geq 0$.

Induction step:

Let c be an arbitrary state in $\text{out}(R_a, b)$. By the induction hypothesis $\mathbf{z}^h \geq c$ and, by the monotonicity of Y, $Y(a, \mathbf{z}^h) \geq Y(a, c)$. Furthermore, since $\mathbf{z}^{h+1} = Y(a, \mathbf{z}^h)$, we have

$$\mathbf{z}^{h+1} \geq Y(a, c). \qquad \text{(i)}$$

Now study any binary value in $\mathbf{z}^{h+1}$, say $\mathbf{z}_j^{h+1} = \alpha \in B$. By (i) it follows that $Y_j(a, c) = \alpha$. Since c is arbitrary, it follows that

$$Y_j(a, c) = \alpha \text{ for all } c \in \text{out}(R_a, b),$$

i.e. the excitation of vertex j is α in all the states in $\text{out}(R_a, b)$. We claim that this implies that $c_j = \alpha$ for all $c \in \text{out}(R_a, b)$. To show this, suppose there exists a $\hat{c} \in \text{out}(R_a, b)$, such that $\hat{c}_j = \beta \neq \alpha$. Study any cycle in $\text{out}(R_a, b)$ containing $\hat{c}$. Since the excitation is the same in all states in $\text{out}(R_a, b)$, it follows that vertex j cannot have the value β in all the states in the cycle (otherwise the cycle would be transient). Suppose it changes to γ in the state $\bar{c}$ in the cycle. If $\gamma = \alpha$, vertex j would be stable from here on, and could never change back to β. Hence, $\gamma \neq \alpha$. Altogether, both β and γ can only have values in $\{\alpha', \times\}$. Since the excitation is α in all states in $\text{out}(R_a, b)$, the only possible transition is from $\beta = \alpha'$ to $\gamma = \times$. However, there cannot be any transition between a state in which vertex j is $\times$ and a state in which it is α', since such a transition would violate the definition of R_a. Hence such a cycle in $\text{out}(R_a, b)$ containing $\hat{c}$ does not exist, and we can conclude that $c_j = \alpha$ for all $c \in \text{out}(R_a, b)$. In summary, we have shown that for any binary value α in $\mathbf{z}^{h+1}$, the corresponding vertex will have the same value α in all the states in

out(R_a, b). Therefore the induction step goes through and the claim follows. □

Note that the graph of R_a may contain cycles with the following property: The value of vertex j is $\alpha \in B$ in all states of the cycle, but the excitation has the value α' in some states of the cycle and $\times$ in the remaining states. Such cycles are transient according to the definition in Section 3. However, it is easy to verify that Theorem 2 still holds if the definition of transient is changed in such a way that these cycles belong to out(R_a, b).

The results of this section will now be illustrated by examples. The ternary simulations corresponding the XMW analyses of Fig. 3 and 4 are shown in Figs. 5 and 6 respectively. Note that in both cases we have that $\mathbf{y}^A$ is equal to the *l.u.b.* of all reachable states and that $\mathbf{z}^B$ is equal to the *l.u.b.* of the outcome of the XMW analysis.

$$\begin{aligned} \mathbf{y}^0 &= 0010 \\ \mathbf{y}^1 &= \times 010 \\ \mathbf{y}^2 &= \times 0 \times 0 \\ \mathbf{y}^3 &= \times 0 \times \times = \mathbf{y}^A \end{aligned}$$

(a)

$$\begin{aligned} \mathbf{z}^0 &= \times 0 \times \times \\ \mathbf{z}^1 &= 10 \times \times \\ \mathbf{z}^2 &= 100 \times \\ \mathbf{z}^3 &= 1001 = \mathbf{z}^B \end{aligned}$$

(b)

Figure 5. Ternary simulation of N_1: (a) Algorithm A; (b) Algorithm B.

$$\begin{aligned} \mathbf{y}^0 &= 100 \\ \mathbf{y}^1 &= \times 00 \\ \mathbf{y}^2 &= \times \times \times = \mathbf{y}^A \end{aligned}$$

(a)

$$\begin{aligned} \mathbf{z}^0 &= \times \times \times \\ \mathbf{z}^1 &= 0 \times \times = \mathbf{z}^B \end{aligned}$$

(b)

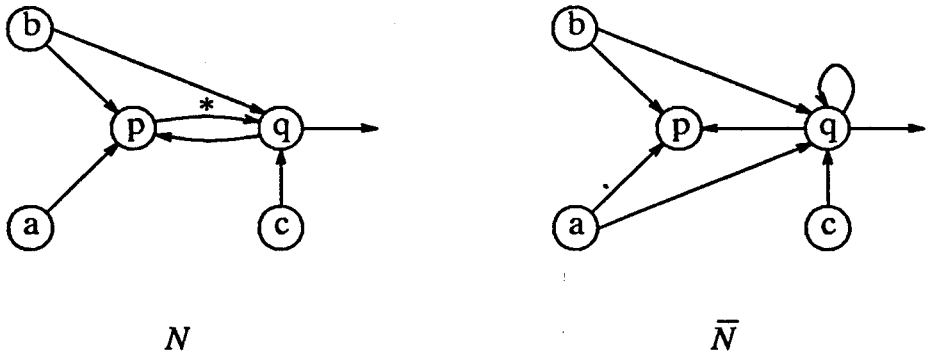
Figure 6. Ternary simulation of N_2: (a) Algorithm A; (b) Algorithm B.

5. Reduced Networks

It is proved in this section that the XMW and ternary analyses can be applied to a much smaller network and can still give the same amount of race information. It is first shown how to transform a network so that the dependence of a vertex on another vertex can be removed in certain cases.

Let $N = \langle V, E, x, y, Y \rangle$ be any network. Assume that $(p,q) \in E$, $p > n$, $p \neq q$, and that $(p,p) \notin E$, i.e. that q is a function vertex whose excitation function depends on the value of another vertex p, where p is a function vertex and does not have a self-loop. Now, let $\bar{N}$ be the network: $\bar{N} = (V, \bar{E}, x, y, \bar{Y})$, where $\bar{E} = E \cup \{(i,q) : (i,p) \in E\} - \{(p,q)\}$. Also, $\bar{Y}_j(x,y) = Y_j(x,y)$ for all $j \neq q$ and $\bar{Y}_q(x,y) = Y_q(x, (y_1, \cdots, y_{p-1}, Y_p(x,y), y_{p+1}, \cdots, y_m))$. The transformation is

performed to remove the dependence of vertex q on the value of vertex p. Note that only edges from function vertices can be removed — the reason for not removing edges from input vertices will become clear later. In Fig. 7 is shown a typical transformation to remove vertex q's dependence on the value of vertex p (i.e. we want to remove the *-marked edge). Assume that in N we have $Y_p = (y_a y_b y_q)'$ and $Y_q = y_b y_c y_p$. Therefore vertex q depends on vertex p (hence the edge from p to q). Moreover, vertex p does not have a self-loop and is also a function vertex. In this case we get $\bar{Y}_q = y_b y_c (y_a y_b y_q)'$, which can be simplified to $\bar{Y}_q = y_a' y_b y_c + y_b y_b' y_c + y_b y_c y_q'$. Note that, since the composition is performed for ternary functions, the term $y_b y_b' y_c$ in $\bar{Y}_q$ cannot be removed. Note also that the vertex q gets a self-loop by this transformation.



N $\bar{N}$

Figure 7. Removal of vertex q's dependency of vertex p.

We now prove, by a series of Lemmas, that the ternary simulation yields identical results for N and $\bar{N}$.

Lemma 1 The result $\mathbf{y}^A$ of Algorithm A for network N is equal to the result $\bar{\mathbf{y}}^{\bar{A}}$ of Algorithm A for network $\bar{N}$.

Proof: Assume N and $\bar{N}$ are started in the stable total state $\hat{a}, b$ and that the input changes to a. Let $\mathbf{a} = l.u.b.\{\hat{a}, a\}$. Now, let $\bar{\mathbf{y}}^0, \bar{\mathbf{y}}^1, \ldots, \bar{\mathbf{y}}^{\bar{A}}$ and $\mathbf{y}^0, \mathbf{y}^1, \ldots, \mathbf{y}^A$ be the results of Algorithm A for $\bar{N}$ and N respectively.

First, if the excitation of vertex p never differs from the state of vertex p, the lemma holds trivially. Otherwise, assume that the excitation of vertex p differs from the state of vertex p for the first time at step r, $r \geq 0$. From the definition and the monotonicity of Algorithm A, we can conclude that

$$Y_p(\mathbf{a}, \mathbf{y}^i) = \begin{cases} b_p & \text{if } i < r \\ \times & \text{if } i \geq r, \end{cases} \qquad \text{(i)}$$

and that

$$\mathbf{y}_p^i = \begin{cases} b_p & \text{if } i < r+1 \\ \times & \text{if } i \geq r+1. \end{cases} \qquad \text{(ii)}$$

Clearly $\mathbf{y}^i = \bar{\mathbf{y}}^i$ for $0 \leq i \leq r$. This together with (i) and the monotonicity of Algorithm A implies that

$$\bar{Y}_p(\mathbf{a}, \bar{\mathbf{y}}^i) = Y_p(\mathbf{a}, \bar{\mathbf{y}}^i) = \begin{cases} b_p & \text{if } i < r \\ \times & \text{if } i \geq r. \end{cases} \qquad \text{(iii)}$$

From (ii), and (iii) it follows that

$$\mathbf{y}_p^i \leq Y_p(\mathbf{a}, \bar{\mathbf{y}}^i) \leq \mathbf{y}_p^{i+1} \quad \text{for } i \geq 0. \qquad \text{(iv)}$$

We now prove by induction on h that $\mathbf{y}^h \leq \bar{\mathbf{y}}^h \leq \mathbf{y}^{h+1}$, for all $h \geq 0$. By the monotonicity of Algorithm A, the claim in the lemma then follows immediately.

Basis:

$h = 0$. Since N and $\bar{N}$ are started in the same state, $\mathbf{y}^0 = \bar{\mathbf{y}}^0$. By the monotonicity of Algorithm A, it follows that $\mathbf{y}^0 \leq \mathbf{y}^1$, and the claim is true for the basis.

Induction hypothesis:

Assume $\mathbf{y}^h \leq \bar{\mathbf{y}}^h \leq \mathbf{y}^{h+1}$, for some $h \geq 0$.

Induction step:

By the monotonicity of Y, the definition of Algorithm A, and the induction hypothesis we have for $i \neq q$:

$$\mathbf{y}_i^{h+1} = Y_i(\mathbf{a}, \mathbf{y}^h) \leq Y_i(\mathbf{a}, \bar{\mathbf{y}}^h) = \bar{Y}_i(\mathbf{a}, \bar{\mathbf{y}}^h) = \bar{\mathbf{y}}_i^{h+1} \leq Y_i(\mathbf{a}, \mathbf{y}^{h+1}) = \mathbf{y}_i^{h+2}.$$

Finally, since $\mathbf{y}_p^h \leq Y_p(\mathbf{a}, \bar{\mathbf{y}}^h) \leq \mathbf{y}_p^{h+1}$, by (iv), it follows that

$$\begin{aligned} \mathbf{y}_q^{h+1} &= Y_q(\mathbf{a}, (\mathbf{y}_1^h, \cdots, \mathbf{y}_{p-1}^h, \mathbf{y}_p^h, \mathbf{y}_{p+1}^h, \cdots, \mathbf{y}_m^h)) \\ &\leq Y_q(\mathbf{a}, (\bar{\mathbf{y}}_1^h, \cdots, \bar{\mathbf{y}}_{p-1}^h, Y_p(\mathbf{a}, \bar{\mathbf{y}}^h), \bar{\mathbf{y}}_{p+1}^h, \cdots, \bar{\mathbf{y}}_m^h)) = \bar{\mathbf{y}}_q^{h+1} \\ &\leq Y_q(\mathbf{a}, (\mathbf{y}_1^{h+1}, \cdots, \mathbf{y}_{p-1}^{h+1}, \mathbf{y}_p^{h+1}, \mathbf{y}_{p+1}^{h+1}, \cdots, \mathbf{y}_m^{h+1})) = \mathbf{y}_q^{h+2}. \end{aligned}$$

Hence, $\mathbf{y}^{h+1} \leq \bar{\mathbf{y}}^{h+1} \leq \mathbf{y}^{h+2}$ and the induction step goes through. □

In a similar way one can verify the following "dual" version of Lemma 1.

Lemma 1^D The result $\mathbf{y}^B$ of Algorithm B for network N is equal to the result $\bar{\mathbf{y}}^{\bar{B}}$ of Algorithm B for network $\bar{N}$.

Note that the above procedure can be repeated until each function vertex has either a self-loop or outdegree 0. The network $\dot{N}$ obtained by carrying out the above reduction procedure until only vertices with self-loops remain and then removing all vertices with outdegree 0 will be called a *reduced* network. Note that,

in general, the reduced network is not unique. For the remaining part of this section the following definition will be needed. A set of vertices F is called a *feedback vertex set* for a directed graph G iff removing the vertices in F and all their incident edges gives an acyclic graph. The following proposition is easily verified.

Proposition 4 If $\dot{N}$ is a reduced version of N, then the function vertices of $\dot{N}$ constitutes a feedback vertex set of the original network N.

Proof: The reduction process can be viewed as a two step process. First, as many edges as possible are removed by using the above reduction procedure. Secondly, when no more edges can be removed, all vertices with outdegree 0 are removed, yielding the network $\dot{N}$. It is sufficient to show that, for any cycle in N, at least one function vertex of the cycle will remain in $\dot{N}$. This follows trivially from the fact that removing an edge in a cycle using the reduction procedure will only shorten the cycle. Eventually, one of the vertices of the cycle will have a self-loop and no further reduction can be made. Since a vertex with a self-loop has outdegree ≥ 1 it will not be removed. Hence, it follows that at least one function vertex of every cycle of N remains in $\dot{N}$, thus forming a feedback vertex set. □

The converse, i.e. that, for any feedback vertex set F of N, one can find a reduced network $\dot{N}$ such that the set of function vertices is identical to F is treated in the following proposition.

Proposition 5 Given any feedback vertex set F for a network N, there exists a reduced network $\dot{N}$ such that the set of function vertices of $\dot{N}$ is equal to F.

Proof: The proof constructs the reduced network "around" the given vertices in F. The idea is to remove edges leading *into* the vertices in F. More formally, we define a sequence of reduced networks $\bar{N}^h$ recursively as follows:

Basis:

$\bar{N}^0 = N$.

Induction step:

Given $\bar{N}^h$, such that F is a proper subset of the function vertices of $\bar{N}^h$, find an edge satisfying the following conditions: i) $(i,j) \in \bar{E}^h$, ii) $i \notin F$, and iii) $j \in F$. Remove edge (i,j) using the procedure described above. If vertex i now has outdegree 0, remove it too. Let $\bar{N}^{h+1}$ be the network so obtained.

The crucial observation for this algorithm is that the set of vertices in $\bar{N}^h$ that have self-loops is a subset of F. This follows from the fact that all vertices of N that have self-loops must be in F (otherwise F would not be a feedback vertex set) and that the application of the reduction procedure can only cause a vertex in F to get a self-loop. From this it is easy to see that we will eventually get some $\bar{N}^k$ such that the set of function vertices in $\bar{N}^k$ is equal to F. □

We are now ready to state and prove the main result of this section. For simplicity, a reduced version of N will be called a feedback vertex model of N. The theorem states essentially that an XMW analysis of a feedback vertex model of N is sufficient, i.e. that it is not necessary to include all the vertices of N in a race analysis. This is a result that contrasts radically with the "classical" binary race models, where a feedback variable analysis (even if augmented with a hazard analysis) is not always correct (see for example Langdon [13]).

Theorem 3 The l.u.b. of the outcome of an *XMW* analysis of a feedback vertex model of a network N is consistent with the l.u.b. of the outcome of an *XMW* analysis of N, in the sense that both analyses produce the same values for the feedback vertices.

Proof: By Theorem 2 it follows that ternary simulation can be used instead of the XMW analysis, since the result of ternary simulation corresponds exactly to the l.u.b. of the outcome of the XMW analysis. Furthermore, Lemma 1 and 1^D together show that ternary simulation yields the same result for N and $\bar{N}$, when only one edge has been removed. It now follows, by induction on the number of times the reduction process is carried out, that ternary analysis yields identical results for N and $\dot{N}$, where $\dot{N}$ is the reduced network. (Clearly, the removal of vertices with outdegree 0 does not change the result of ternary simulation.) Finally, by Proposition 6 we know that, given a feedback vertex set, we can construct a reduced network with exactly that set as the set of function vertices. □

In order to analyze a network as fast as possible it would appear desirable to have the smallest number of vertices in the reduced network. It is well known [11] that the problem of finding a minimal feedback vertex set is NP-complete; hence the best we can hope for (assuming $P \neq NP$) is an approximation algorithm. However, it is not necessarily optimal to find a minimal feedback vertex set and reduce the network down to this set. It is true that ternary simulation (in the worst case) takes time $O(n^2)$, where n is the number of vertices, thus indicating the desirability of having as few vertices in the network as possible. Unfortunately, this result is valid under the assumption that each excitation function can be computed in constant time — an assumption that may not be true in the reduced network. In fact, with more realistic assumptions, it is not difficult to find examples in which the cost of ternary simulation of the reduced network is *higher* than the cost of ternary simulation of the original network. The opposite is also possible. For example, the new excitation functions can be sometimes drastically simplified, leading to a reduced network that is substantially more efficient to simulate than the original network. Hence, the questions whether to reduce a network or not, and how much reduction should be done, must unfortunately be decided according to some heuristic. (The answer will also depend heavily on implementation details.)

The ideas above will now be illustrated by examples. Consider network N_1 of Fig. 2. A possible feedback vertex set is {3}. Using the approach described above we get the reduced network $\dot{N}_1$ of Fig. 8 with excitation functions:

$$Y_1 = x_1 \quad Y_2 = x_2 \quad Y_3 = (y_1 + (y_2 + y_3)')'.$$

In Fig. 9 are shown both the XMW analysis and the ternary simulation for the same transition as was analyzed in Fig. 3, i.e. with $\hat{a}, b = 00, 001$ and new input $a = 10$.

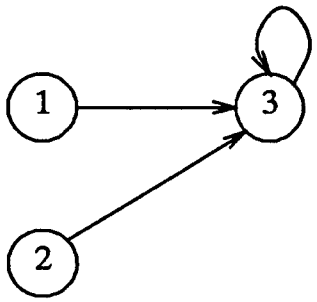


Figure 8. Network $\dot{N}_1$.

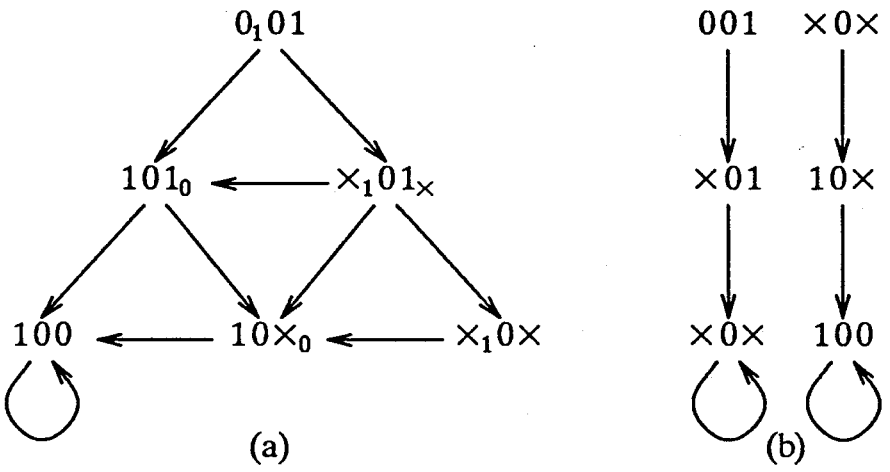


Figure 9. (a) XMW analysis of $\dot{N}_1$; (b) ternary simulation of $\dot{N}_1$.

Next consider network N_2 of Section 3. The set {2} constitutes a feedback vertex set of the network N_2. Reducing N_2 yields network $\dot{N}_2$ with excitation functions:

$$Y_1 = x_1 \quad Y_2 = (y_1 + (y_1 + y_2)')'.$$

In Fig. 10 are shown the XMW analysis and the ternary simulation for the same transition as was analyzed in Fig. 4, i.e. with $\hat{a}, b = 1, 10$ and new input $a = 0$.

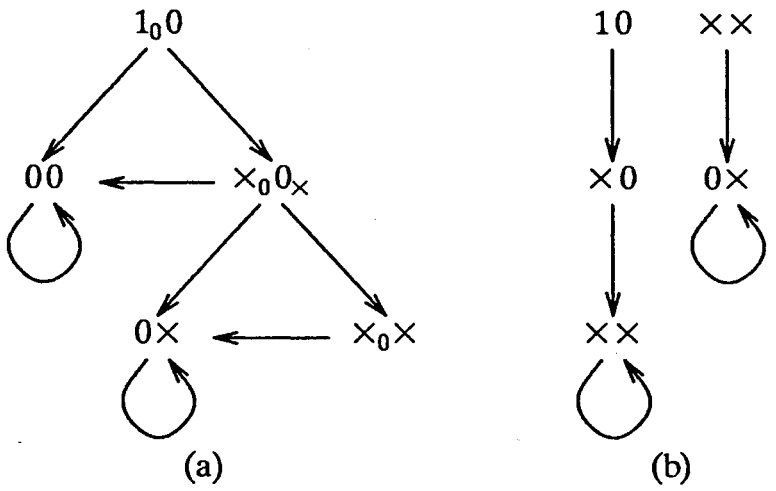


Figure 10. (a) XMW analysis of $\dot{N}_2$; (b) ternary simulation of $\dot{N}_2$.

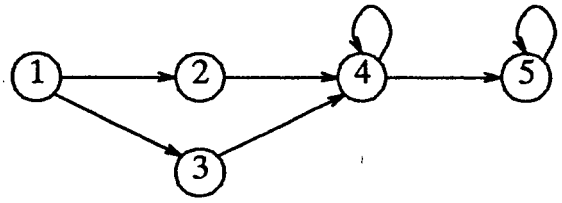


Figure 11. Network N_3.

The next example shows that a further reduction, below a minimal feedback vertex set, may be possible in some cases. Consider network N_3 of Fig. 11 with excitation functions:

$$Y_1 = x_1 \quad Y_2 = y_1 \quad Y_3 = y_1 \quad Y_4 = y_2y_3 + y_2y_4 + y_3y_4 \quad Y_5 = y_4 + y_5$$

If the edge (2, 4) is removed, we get $\bar{N}_3$ with excitation functions:

$$\bar{Y}_1 = x_1 \quad \bar{Y}_2 = y_1 \quad \bar{Y}_3 = y_1 \quad \bar{Y}_4 = y_1y_3 + y_1y_4 + y_3y_4 \quad \bar{Y}_5 = y_4 + y_5$$

Furthermore, if the edge (3, 4) is also removed, we get $\dot{N}_3$ with excitation functions:

$$\dot{Y}_1 = x_1 \quad \dot{Y}_2 = y_1 \quad \dot{Y}_3 = y_1 \quad \dot{Y}_4 = y_1y_1 + y_1y_4 + y_1y_4 \quad \dot{Y}_5 = y_4 + y_5$$

It is easy to see that we cannot remove any other edges and that vertices 4 and 5 constitute a minimal feedback vertex set. However, $\dot{Y}_4$ can be simplified to $\dot{Y}_4 = y_1$. Here the edge (4, 4), indicating that the excitation of vertex 4 depends on the value of vertex 4, can simply be erased. Note that vertex 4 *did* depend on its own value in the original network. After the edge (4, 4) has been erased, the edge (4, 5) can also be removed. If we then remove the vertices with outdegree 0, we obtain network $\ddot{N}_3$, shown in Fig. 12, with excitation functions:

$$\ddot{Y}_1 = x_1 \qquad \ddot{Y}_5 = y_1 + y_5$$

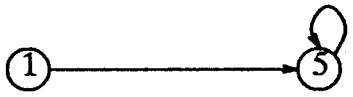


Figure 12. Network $\ddot{N}_3$.

Reductions below the feedback vertex set, such as the one described above, may be possible because some of the excitation functions may become degenerate. On the other hand, the feedback vertex set reflects only the structure of the graph. It can be easily verified that the reduction which uses functional degeneracies to remove self-loops is also valid, in the sense of Theorem 3. Unfortunately, the problem of determining whether a ternary function depends on a particular variable is NP-complete in general (by transformation from Boolean satisfiability).

6. Output Hazards

In the previous sections we were concerned only with the nontransient behavior of a given network. We concentrated on the detection of critical races and oscillations, but did not consider hazards explicitly. In fact, the XMW analysis does take into account implicitly all possible hazards associated with the vertex variables, when determining the nontransient outcome of a transition. Thus, if one is only interested in the nontransient behavior of a network, it is sufficient to find the outcome using the XMW analysis, or preferably the equivalent ternary simulation, and no further analysis is required. However, the network being analyzed is often a part of a larger system. In such cases, some subset of the vertices may be "visible" to the rest of the system. We will call such vertices *output vertices*. When output vertices are present, there is a new problem. Consider, for example, an output vertex that has the value 0 initially, and also in all the states of $\text{out}(R_a, b)$. It is quite possible that the vertex has the value 1 or $\times$ in some of the transient states during the transition. Such short glitches must be detected, since they may cause unwanted state changes in the rest of the system controlled by this output vertex. In this section we formally define the notion of output hazards, and also give methods for detecting them. We also show that the same information may be obtained from a reduced network.

Assume that a network N is started in the stable state $\hat{a}, b$ and that the input changes to a. We say that there is a *static 1-hazard* on an output vertex i for the transition $\hat{a} \rightarrow a$ iff $b_i = 1$, $c_i = 1$ for every state $c \in \text{out}(R_a, b)$, and there exists a state d such that $b\, R_a^* d$ and $d_i \neq 1$. A static 0-hazard is defined similarly.

The following theorem shows how the results of ternary simulation can be used to detect static output hazards [10, 20].

Theorem 4 Assume network N is started in the stable total state $\hat{a}, b$ and the input changes to a. Let $\mathbf{y}^A$ be the result of Algorithm A and $\mathbf{z}^B$ be the result of Algorithm B. Then output vertex i has a static 1(0)-hazard iff $b_i = \mathbf{z}_i^B = 1(0)$ and $\mathbf{y}_i^A = \times$.

Proof Suppose there is a static output hazard on output vertex i. Without loss of generality, assume it is a static 1-hazard. Then $b_i = 1$, and $c_i = 1$ for all states c in out(R_a, b). Also, there must exists a state d, reachable from b, such that $d_i \neq 1$. Since $\mathbf{z}^B = l.u.b.(\text{out}(R_a, b))$, by Theorem 2, it follows that $\mathbf{z}_i^B = 1$. Furthermore, since $b R_a^* d$ and $d_i \neq 1$, we must have $l.u.b.\{e_i : b R_a^* e\} = \times$. However $\mathbf{y}^A = l.u.b.\{e : b R_a^* e\}$, by Theorem 1; hence $\mathbf{y}_i^A = \times$, and the claim follows.

Conversely, assume, without loss of generality, that $\mathbf{z}_i^B = b_i = 1$ and that $\mathbf{y}_i^A = \times$. By Theorem 2 it follows that $c_i = 1$ for any state c in out$(R_a, b))$. By Theorem 1, there must exist a state d reachable from b, such that $d_i \neq 1$; otherwise $\mathbf{y}_i^A$ would be 1. Hence the claim goes through and the theorem holds. □

In connection with static hazards, one may ask whether all timing problems can be detected by an XMW analysis of the network. In particular, can wire delays, i.e. delays associated with the edges, create new timing problems? One can account for wire delays in our model by simply inserting a "delay vertex" in each edge; the excitation function of such a vertex is the identity function. It is easily verified that the network with wire delays yields the same results. First, by Theorem 4, ternary simulation can be used instead of an XMW analysis for the detection of static hazards. Furthermore, by Lemmas 1 and 1^D in Section 5, the dependence on a function vertex without a self-loop can be removed without affecting the ternary simulation. In particular, the dependence on a delay vertex can be so removed. By induction on the number of wire delays in the network, it follows that a network without wire delays has the same static output hazards as a network with any number of wire delays added.

As shown in Section 5, it is sometimes advantageous to reduce the original network to a feedback vertex network. However, the reduction described in Section 5 does not take care of output vertices. There are two solutions to this problem. The first alternative is to change the reduction procedure in such a way that output vertices are never removed. Unfortunately, this may lead to a substantially larger network. The other alternative is to perform the analysis on a feedback vertex model, and then recreate the output values and transitions using the results obtained from the feedback vertex model. We are going to focus on this second alternative. Note however that the first alterative can be viewed as a special case of the second approach (the reduction is simply not carried out as far as possible

and the output functions are trivial).

In the sequel, we will assume that $\dot{N}$ is a reduced version of some network N. Without loss of generality assume vertices $n+1$ to $n+o$ are output vertices in N. The idea is to add a set of *output functions*, $O_1, \cdots, O_o$, to the feedback vertex network. In general, function O_i is a function from $T^{\dot{m}}$ to T, and is computed as follows:

$O_i = y_i$;
while O_i depends on any y_k such that $k \notin \dot{V}$ **do**
    replace every occurrence of y_k in O_i with Y_k

Since $\dot{V}$ is a feedback vertex set, the procedure is guaranteed to halt with an output function that depends only on input nodes and feedback vertices. (As usual, the final function O_i can be simplified using the laws of the ternary algebra.) The following result, stating that the values of the feedback variables uniquely determine the output values, is easily verified.

Proposition 6 Assume a, b is a stable total state in N. Let $a, \dot{b}$ be the corresponding stable total state in $\dot{N}$. Then the value of output vertex i, i.e. b_{n+i}, is equal to $O_i(a, \dot{b})$.

Proof: This follows from the fact that a, b is a stable total state, i.e. $b_j = Y_j(a, b)$, and from the construction of O_i. □

Proposition 6 together with Theorem 4 gives us the following method of analyzing a network N. First, reduce the network as much as desirable to $\dot{N}$. Also, compute the output functions. Use ternary simulation to get the two state vectors $\dot{\mathbf{y}}^A$ and $\dot{\mathbf{z}}^B$. Evaluate the output functions for $\dot{b}$, $\dot{\mathbf{y}}^A$ and $\dot{\mathbf{z}}^B$. The results are interpreted as in Theorems 2 and 4. Hence, both the outcomes of the transitions and potential static output hazards are correctly computed.

7. Gate Circuits

It is shown in this section how gate circuits can be analyzed using the framework established in the previous sections. The correspondence between a gate circuit and the graph model is very natural, as described below.

Given a gate circuit with n external inputs, the graph N is formed in the following way: There is an input vertex for each input variable to the gate circuit and a function vertex for every gate. There is an edge between vertex i and vertex j iff the gate j has at least one of its inputs connected to the output of gate i if $i > n$, or to input x_i if $i \leq n$. Note that there may be more than one wire between two gates in a circuit, whereas in N we have at most one edge. This will affect the function associated with each node. Given any Boolean function $f : B^r \rightarrow B$ the *ternary extension* $\mathbf{f}$ of f is defined as follows. For any $\mathbf{z} \in T^r$,

$$\mathbf{f}(\mathbf{z}) = l.u.b.\{f(z) : z \in B^r, z \leq \mathbf{z}\}.$$

The ternary extensions of OR, AND and complement functions were shown in Fig. 1. Two steps have to be carried out to compute the ternary function associated with the function node i. First, the ternary extension of the original Boolean gate function must be computed. Second, the input variables are identified with the output vertex feeding them. Consider, for example, the gate circuit G_1 of Fig. 13 consisting of a 2-input XOR (exclusive OR) gate, with both inputs fed by a single inverter. The ternary extension of the XOR gate is $c'd + cd'$, where c and d denote the two inputs. Identifying the inputs and simplifying according to the ternary algebra gives the excitation functions:

$$Y_1 = x_1 \quad Y_2 = y_1' \quad Y_3 = y_2' y_2$$

Note that the last excitation function is *not* identical to 0.

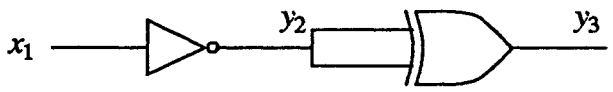


Figure 13. Gate circuit G_1.

In classical race models the basic assumption is that gates can only be in the states 0 and 1, and that transitions from 0 to 1 or from 1 to 0 are instantaneous. However, it is reasonable to assume that a transition takes a nonnegligible amount of time, and may go through an intermediate voltage. Some gates connected to the node with an intermediate voltage may interpret this voltage as a 1, whereas others may interpret it as 0. The $\times$ value introduced in the XMW model, captures this uncertainty. Also, the fact that all transitions in the XMW relation are allowed to go through $\times$ handles slowly changing gates.

It is interesting to compare the results of the previous sections with earlier results concerning gate circuits. The classical transition model, used when no assumptions are made about the delays in the circuit, is the General Multiple Winner (GMW) model [7]. The GMW transition relation is very similar to the XMW relation, except that all the states are assumed to be binary. Otherwise, the two models are identical, i.e. any nonempty subset of the unstable gates can change to their excitation values.

Let $\tilde{N}$ denote the network obtained from a network N by adding a delay element in every wire. We have the following result for gate circuits.

Theorem 5 The following analysis techniques are all equivalent for gate circuits from the point of view of nontransient state behavior and static output hazards:

1. GMW analysis of $\tilde{N}$.
2. XMW analysis of (a) $\tilde{N}$, (b) N, and (c) $\dot{N}$, any feedback vertex model of N.
3. Ternary simulation of (a) $\tilde{N}$, (b) N, and (c) $\dot{N}$.

Proof: The equivalence of 3(a), 3(b), and 1 for computing the nontransient behavior was shown in [8, Theorem 1]. Furthermore, by similar arguments as in Theorem 4, it is easy to show that this equivalence also holds for detecting static hazards. (This was not shown in [8], but all the necessary results are there.) The equivalence between 3b and 3c follows from Theorem 3. Also, by Theorems 1, 2 and 4, we immediately have the equivalence of 2a and 3a, 2b and 3b, and 2c and 3c. □

8. Switch-level Circuits

Switch-level models are frequently used for simulating MOS VLSI circuits. In switch-level models each transistor is viewed as a switch that can be turned on or off by the signal on the gate of the transistor. A number of different switch-level models are described in this section. We will focus on switch-level models for CMOS circuits, although some of the models can be easily modified to handle NMOS circuits as well. The notation used is modeled after [9].

An S-graph (S for switch) is a finite, undirected, labeled graph with:

1) *Supply nodes* shown as black dots and labeled by 0 or 1.
2) *Internal nodes* shown as white dots.
3) *Key nodes*, which are special internal nodes, each of which is labeled by a different *key letter* $Q_1, \cdots, Q_k$.
4) *Input letters* $A_1, \cdots, A_n$.
5) *Edges*, each of which is labeled with a label of the form Q^N or Q^P, where Q is either an input letter or a key letter. Each key letter appears as some edge label.

The edges represent the transistors in the circuit, whereas the nodes represent the connection points. A superscript P on an edge label indicates that the transistor is P-type; a superscript N denotes an N-type transistor. Following [3], we define a *channel-connected subnetwork* to be a subgraph $\bar{S}$ of the S-graph such that there is at least one path that does not go through a supply node between any two internal nodes in $\bar{S}$. Except for the supply nodes, the channel-connected subnetworks are disjoint.

A (ternary) *input-key state* is an assignment of 0, 1, or × to each input letter and each key letter. Similarly, a *total state* is an assignment of 0, 1, or × to each input letter and to each internal node. A path in the graph consisting of only

P-transistors is called a *P-path*. An *N-path* is defined similarly. A path that consists of both *P* and *N* transistors is called an *M-path* (M for mixed). A path is said to be *definite* if the values of the gates of all the P-transistors in the path are 0, and the values of the gates of all *N* transistors in the path are 1. A path is said to be *indefinite* if it is not definite, and the values of the gates of all the P-transistors in the path are either 0 or ×, and the values of the gates of all the *N* transistors in the path are either 1 or ×.

A property of CMOS circuits that many switch-level models fail to capture is the fact that a P-transistor conducts a 1-signal well, but a 0-signal rather poorly. Similarly, an N-transistor conducts a 0 well, and a 1 poorly. For this reason, we must distinguish between P-, N-, and M-paths. We do this by introducing two sets of *path functions*. For an internal node i we define the ternary path functions:

$$g_{i1} = \begin{cases} 1 & \text{if there is a definite P-path to 1} \\ \times & \text{if there is no definite P-path but there is an indefinite P-Path to 1} \\ 0 & \text{if there is no definite nor indefinite P-path to 1} \end{cases}$$

$$t_{i1} = \begin{cases} 1 & \text{if there is a definite M-path to 1} \\ \times & \text{if there is no definite M-path but there is an indefinite M-Path to 1} \\ 0 & \text{if there is no definite nor indefinite M-path to 1} \end{cases}$$

The corresponding path functions g_{i0} and t_{i0} are defined in a similar way. (Note that certain types of paths, e.g. "self-dependent" paths [9], may be disregarded to model the circuit behavior more accurately.)

The path functions can be computed in a number of different ways. One possibility is to enumerate all possible paths and then identify the different *P*-, *N*-, and *M*-paths. However, a more efficient method is to use a "signal flow graph" approach, similar to the procedure to convert a finite automaton to a regular expression [5]. This approach can be viewed as solving a set of linear equations with a somewhat modified version of Gaussian elimination. For more details, the reader is referred to [4] where this approach is analyzed for different graphs and is applied to switch-level simulation.

Consider the CMOS circuit C_1 of Fig. 14(a). In Fig. 14(b) is shown the corresponding S-graph. (The input and key node naming convention has been ignored in order to simplify notation.) Note that C_1 consists of two channel-connected subnetworks. It is easy to verify that for the internal node a, we have: $g_{a1} = t_{a1} = A'$, and $g_{a0} = t_{a0} = A$. (Note that we are using the ternary OR, AND and complement as defined in Section 2.) Furthermore, the path functions for node c are:

$$g_{c1} = Q'B' \quad t_{c1} = g_{c1} + AB' \quad g_{c0} = Q + AB \quad t_{c0} = g_{c0} + Q'B.$$

In general, t_{i1} must always be of the form $g_{i1} + f_{i1}$ for some function f_{i1}. The same holds for t_{i0}.

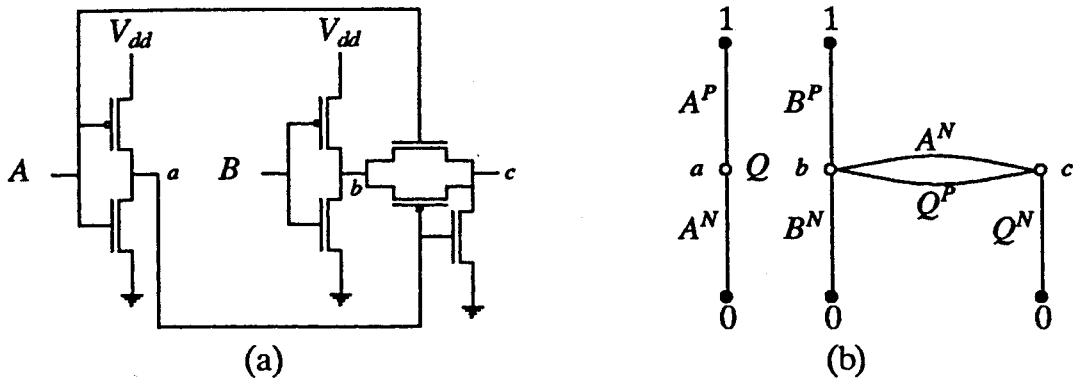


Figure 14. (a) CMOS circuit C_1; (b) corresponding S-graph S_1.

Let Y_i denote the node excitation function of the internal node i. There are four very natural ways to use the ternary path functions to compute this function. However, before we describe these models, we prove the following proposition for later use.

Proposition 7 If $a \in T$ and $b \in T$ then

$$a + (a+b)'\times = a + b'\times.$$

Proof: There are three cases. If $a = 1$ both the LHS and the RHS are 1, since $1+0 = 1+1 = 1+\times = 1$. Secondly, if $a = \times$, both the LHS and the RHS are $\times$, since the second terms in the LHS and the RHS can only contribute an $\times$ or a 0, and $\times + \times = \times + 0 = \times$. Finally, if $a = 0$ then the LHS is equal to $0 + (0+b)'\times = b'\times$, which is equal to the RHS. □

All four node excitation models defined below use the same basic concept. The node excitation function Y_i will always be of the form: $f_1 + (f_1 + f_0)'\times$. Intuitively, f_1 denotes the conditions under which Y_i should be 1, and f_0 gives the conditions under which Y_i should be 0. In view of Proposition 7, we will immediately simplify these expressions to the form $f_1 + f_0'\times$. The four models are:

Model 1 $\quad Y_i = g_{i1}t_{i0}' + (g_{i0}t_{i1}')'\times$

Model 2 $\quad Y_i = g_{i1}g_{i0}' + (g_{i0}g_{i1}')'\times$

Model 3 $\quad Y_i = t_{i1}t_{i0}' + (t_{i0}t_{i1}')'\times$

Model 4 $\quad Y_i = g_{i1}g_{i0}' + t_{i1}t_{i0}' + (g_{i0}g_{i1}' + t_{i0}t_{i1}')'\times$

Model 1, introduced in [9], yields a binary output iff there is a "good", i.e. definite, P(N)-path to 1(0) and no path whatsoever to 0(1). Furthermore, whenever there is a "fight", i.e. both a path to 1 and a path to 0, whenever there is

only a weak path to 0 or 1, or whenever the node is isolated, i.e. there is no path to 1 and no path to 0, the node excitation is $\times$. This is a very restrictive model and captures stringent design rules for combinational static CMOS circuits. The second model captures the fact that a P-path to 1 is "stronger" than a P-path to 0. We get $Y = 1$ (0) as long as there is at least one good path to 1 (0), but no good path to 0 (1). Hence, a "fight" between a strong path and a weak path is resolved in favor of the strong path. This is a more liberal rule, but seems to be necessary in order to explain certain very "tricky" exclusive OR-gate designs [9]. The third model is more traditional and corresponds to a special case of the model in [3]. Here there is no distinction at all between P- and N-transistors, and we have $Y = 1$ (0) iff there is some definite path to 1 (0) but no path to 0 (1). The final model is a mixture of models 2 and 3. Here the rules are: strong paths override weak paths; if there are no strong paths, the weak paths determine the output.

Consider the CMOS circuit C_1 of Fig. 14. Using the path functions for node a derived earlier, it is easy to verify that all four models yield the node excitation function $Y_a = A'$. A more complicated example is node c. Model 1 and 2 yield the following node excitation functions:

$$\begin{aligned}
\text{Model 1} \quad Y_c &= g_{c1}t'_{c0} + (g_{c0}t'_{c1})'\times = \\
&= Q'B'(Q + AB + Q'B)' + ((Q + AB)(Q'B' + AB')')'\times = \\
&= \cdots = Q'B' + (Q'A' + AB')\times \\
\text{Model 2} \quad Y_c &= g_{c1}g'_{c0} + (g_{c0}g'_{c1})'\times = \\
&= Q'B'(Q + AB)' + ((Q + AB)(Q'B')')'\times = \\
&= \cdots = Q'B' + Q'A'\times.
\end{aligned}$$

Note that a substantial amount of reduction has been performed above in order to get the final expressions. However, all simplifications consist of sequences of fairly simple steps, and can be carried out by a program. Also, even if the cost of doing the simplification is substantial, this is a preprocessing step which is done only once. Hopefully, the efficiency gained in the simulation phase will justify the preprocessing.

One can question whether models 2-4 are necessary in "well-designed" CMOS circuits. Unfortunately, model 1 is sometimes too restrictive to explain commonly used circuits [9]. An interesting combination would be to use two different node excitation functions: a rather liberal model for the transient analysis, and a more restrictive model for a steady-state analysis. For example, model 2 could be used for the transient analysis and model 1 for the steady state analysis.

The above node excitation functions all fail to capture the fact that there is a certain amount of capacitance in MOS circuits. In particular, the key nodes can have a fairly large capacitance associated with them; hence there is a certain amount of "memory" in each key node. The case when there is only one key node in every channel-connected subnetwork, can be handled in a very

straightforward way. The basic assumption is that a 1 (0) stored on a key node can only determine the excitation of that node, if the node is completely isolated from 0 (1). One can verify that the following models do take this into account.

$$\text{Model } 1^M \quad Y_i = (g_{i1}+y_i)t'_{i0}+((g_{i0}+y'_i)t'_{i1})'\times$$
$$\text{Model } 2^M \quad Y_i = g_{i1}g'_{i0}+y_it'_{i0}+(g_{i0}g'_{i1}+y'_it'_{i1})'\times$$
$$\text{Model } 3^M \quad Y_i = t_{i1}t'_{i0}+y_it'_{i0}+(t_{i0}t'_{i1}+y'_it'_{i1})'\times$$
$$\text{Model } 4^M \quad Y_i = g_{i1}g'_{i0}+t_{i1}t'_{i0}+y_it'_{i0}+(g_{i0}g'_{i1}+t_{i0}t'_{i1}+y'_it'_{i1})'\times$$

For example, using model 1^M, we get Y to be 1 if there is a good path to 1 and no path to 0, or if the previous value was 1 and there is no path to 0. A dual situation holds for $Y=0$.

Consider again the CMOS circuit C_1 of Fig. 14. Using model 1^M and assuming node c is a key node, we get the node excitation functions for nodes a and c:

$$\begin{aligned}
Y_a &= (g_{a1}+y_a)t'_{a0}+((g_{a0}+y'_a)t'_{a1})'\times = \\
&= (A'+y_a)A'+((A+y'_a)(A')')'\times = \\
&= \cdots = A' \\
Y_c &= (g_{c1}+y_c)t'_{c0}+((g_{c0}+y'_c)t'_{c1})'\times = \\
&= (Q'B'+y_c)(Q+AB+Q'B)'+(((Q+AB)+y'_c)(Q'B'+AB')')'\times = \\
&= \cdots = Q'B'+(Q'A'y_c+AB')\times
\end{aligned}$$

When there is more than one key node in each channel-connected subnetwork, a more complicated node excitation function must be used. This is because the charges stored on the nodes may interact with each other, and charge can "spill over" from one node to another causing the excitation to become undefined. To handle this charge sharing we need path functions for the paths between any two key nodes i and j in the channel-connected subnetwork. We call such a path function t_{ij} for consistency. It is defined like t_{i0} and t_{i1}, i.e. $t_{ij}=1$ if there is a definite M-path from node i to node j, $t_{ij}=0$ if there is no definite nor indefinite M-path from i to j, and $t_{ij}=\times$ otherwise, i.e. if there is no definite M-path but there is an indefinite M-path from node i to node j. It is easy to verify that one can disregard all paths that go *through* the nodes 1 or 0, simply because such paths will show up in t_{i0} and t_{i1} and "override" the charge sharing path. Note that this is essentially what Bryant [3] calls blocking. Assume, for notational convenience, that key nodes $2,3,\cdots,i-1,i$ are in a channel-connected subgraph, and that we are interested in the node excitation function of node i. We assume that charge sharing comes into effect only when the nodes are isolated from the supply nodes. Model 1, extended to handle the multiple key node case, is as follows:

$$\begin{aligned}
\text{Model } 1^C \quad Y_i = {} & g_{i1}t'_{i0}+y_i(y_2+t'_{i2})(y_3+t'_{i3})\cdots(y_{i-1}+t'_{i,i-1})t'_{i0}+ \\
& +(g_{i0}t'_{i1}+y'_i(y'_2+t'_{i2})(y'_3+t'_{i3})\cdots(y'_{i-1}+t'_{i,i-1})t'_{i1})'\times
\end{aligned}$$

It is easy to derive similar node excitation functions for the other basic CMOS

node excitation functions.

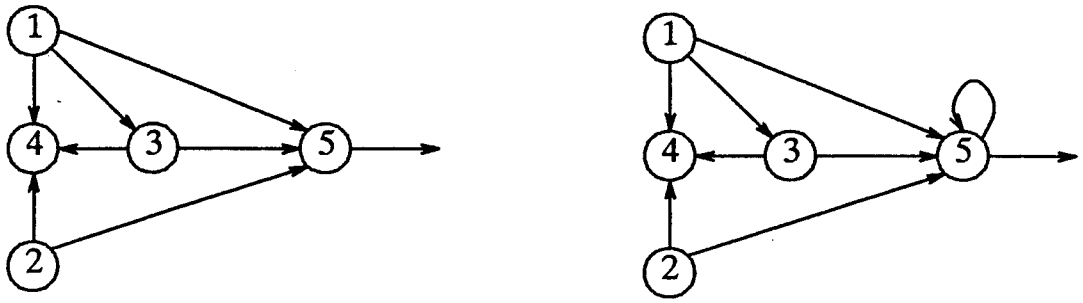
In the above discussion only the key nodes were assumed to have memory. Furthermore, it was assumed that all key nodes have the same "strength". A common technique in MOS circuits is the use of pre-charged lines. In such circuits, certain nodes are designed with a substantially higher capacitance. This can be modeled as if these nodes were "stronger" than normal nodes. We will not derive node excitation functions for this case, but the interested reader is referred to Bryant's work [4]. Here it suffices to say that, using the MOSSIM model [3] and the procedure described in [4], one can derive ternary functions for the node excitations. Using this approach, not only CMOS, but also NMOS circuits can be handled.

9. Network Model for MOS Circuits

It is shown in this section how the different switch-level models described in the previous section can all be handled by the framework established earlier. There is one input vertex for every input letter in the S-graph. Furthermore, there is one function vertex for each internal node in the S-graph. The excitation function associated with a function vertex is simply one of the node excitation functions described in Section 8. (It is easy to verify that all node excitation functions of Section 8 satisfy the monotonicity property of Section 3.) There is an edge between vertex i and vertex j if the excitation function of vertex j depends on the input letter A_i if $i \leq n$, or on the key node y_i if $i > n$.

Consider for example the CMOS circuit C_1 of Fig. 14. The corresponding network N_5, when the node excitation model 1 is used, is shown in Fig. 15. The excitation functions are:

$$Y_1 = A,\ Y_2 = B,\ Y_3 = y_1',\ Y_4 = y_2'(y_3' + y_1'y_3 + \times),\ Y_5 = y_2'y_3' + (y_1'y_3' + y_1y_2')\times$$



N_5 N_6

Figure 15. (a) Network N_5; (b) network N_6.

On the other hand, if the node excitation function 1^M is used, we get the network N_6 of Fig. 15(b), with excitation functions:

$$Y_1 = A,\ Y_2 = B,\ Y_3 = y_1',\ Y_4 = y_2'(y_3' + y_1'y_3 + \times),\ Y_5 = y_2'y_3' + (y_1'y_3'y_5 + y_1y_2')\times$$

In the model above, only the input letters and the internal nodes have state variables associated with them. There are several natural ways to modify this model. First, we can associate a state variable with each internal node and with each transistor. The excitation function for a transistor is trivial: For an N-transistor i, controlled by node y_j, the excitation is $Y_i = y_j$; similarly, for a P-transistor i, controlled by y_j, $Y_i = y_j'$. We will call such a network $\tilde{N}$. Second, it is possible to apply the reduction procedure of Section 6, and obtain a reduced network $\dot{N}$. Naturally, combinations of the these ideas are also possible. The following theorem summarizes the results concerning simulation of switch-level circuits.

Theorem 6 The following analysis techniques are all equivalent for switch-level circuits, using anyone of the node excitation models described in Section 8, from the point of view of nontransient state behavior and static output hazards:

1. XMW analysis of: (a) $\tilde{N}$, (b) N, and (c) $\dot{N}$.
2. Ternary simulation of: (a) $\tilde{N}$, (b) N, and (c) $\dot{N}$.

Proof: The equivalence between an XMW analysis and ternary simulation of the same network follows from Theorems 1, 2 and 4. Hence, we will only discuss the equivalence of the different networks for ternary simulation. Since no function vertex in $\tilde{N}$ representing a transistor can have a self loop, Lemma 1 and 1^D applies. Hence, without affecting the ternary simulation we can remove all the transistor function vertices, yielding network N. This, together with Theorem 4, demonstrates the equivalence between ternary simulation of N and $\tilde{N}$. Using the same arguments it follows that ternary simulation of N and $\dot{N}$ are equivalent. □

Returning to our examples, we apply the reduction method to remove the function vertex 3 from networks N_5 and N_6. We obtain the networks $\dot{N}_5$ and $\dot{N}_6$ of Fig. 16, with excitation functions:

$$Y_1 = A \qquad Y_2 = B \qquad Y_5 = y_1y_2' + y_1y_1'$$

and

$$Y_1 = A \qquad Y_2 = B \qquad Y_5 = y_1y_2' + y_1y_1'y_5.$$

respectively.

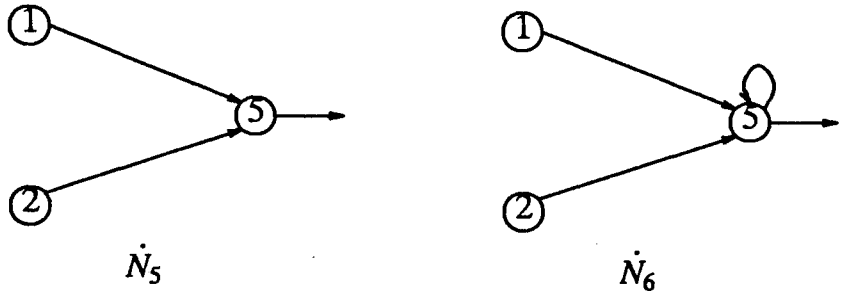


Figure 16. (a) Network $\dot{N}_5$; (b) network $\dot{N}_6$.

10. Conclusions

In this paper we have developed a general framework for the analysis of asynchronous circuits. We have shown how both gate circuits and switch-level circuits fit into the framework. (The interested reader can easily verify that the framework can also handle relay circuits.) Using the framework, we defined a transition relation based on the assumptions that delays are inertial, unknown, but finite. Contrary to classical binary race models, the XMW analysis gives the same results even if only a feedback vertex set of state variables is used.

The XMW analysis is exponential in the number of state variables. However, ternary simulation can be used to efficiently compute the outcome of the XMW analysis. Hence, race detection and hazard detection in the XMW model can be done in polynomial time.

Unfortunately, the XMW transition relation is "pessimistic", in the sense that it will detect timing problems that are highly unlikely to occur in practice. One example of a more optimistic model is the almost-equal-delay (AED) model [6, 18]. This model rejects certain outcomes of the GMW model as very unlikely; in fact, it is probably too optimistic for most applications. Nevertheless, we feel that this work represents a step in the right direction, and that more realistic models can be developed. The results of the present paper constitute a consistent framework in which such problems can be studied.

References

[1] M. A. Breuer, "A Note on Three-Valued Logic Simulation," *IEEE Transactions on Computers*, Vol. C-21, pp. 399-401, Apr. 1972.

[2] R. E. Bryant, "Race Detection in MOS Circuits by Ternary Simulation," in *VLSI '83*, F. Anceau and E. J. Aas, Eds., Elsevier Science Publishers B. V. (North Holland), 1983, pp. 85-95.

[3] R. E. Bryant, "A Switch-Level Model and Simulator for MOS Digital Systems," *IEEE Transactions on Computers*, Vol. C-33, No. 2, pp. 160-177, Feb. 1984.

[4] R. E. Bryant, "A Collection of Papers about a Symbolic Analyzer for MOS Circuits," Technical Report CMU-CS-86-114, Department of Computer Science, Carnegie-Mellon University, Mar. 1986.

[5] J. A. Brzozowski and E. J. McCluskey, Jr., "Signal Flow Graph Techniques for Sequential Circuit State Diagrams," *IEEE Transactions on Electronic Computers*, Vol. EC-12, No. 2, pp. 67-76, Apr. 1963.

[6] J. A. Brzozowski and M. Yoeli, *Digital Networks*, Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[7] J. A. Brzozowski and M. Yoeli, "On a Ternary Model of Gate Networks," *IEEE Transactions on Computers*, Vol. C-28, No. 3, pp. 178-183, Mar. 1979.

[8] J. A. Brzozowski and C-J. Seger, "Correspondence between Ternary Simulation and Binary Race Analysis in Gate Networks," in *Automata, Languages and Programming, Lecture Notes in Computer Science, Vol. 226*, L. Kott, Ed., Springer Verlag, 1986, pp. 67-78.

[9] J. A. Brzozowski and M. Yoeli, "Combinational Static CMOS Networks," in *VLSI Algorithms and Architectures, Lecture Notes in Computer Science, Vol. 227*, F. Makedon, K. Melhorn, T. Papatheodorou and P. Spirakis, Eds., Springer Verlag, 1986, pp. 271-282.

[10] E. B. Eichelberger, "Hazard Detection in Combinational and Sequential Switching Circuits," *IBM Journal of Research and Development*, Vol. 9, pp. 90-99, Mar. 1965.

[11] M. R. Garey and D. S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., New York, 1979.

[12] D. A. Huffman, "The Synthesis of Sequential Switching Circuits," in *Sequential Machines: Selected Papers*, E. F. Moore, Ed., Addison-Wesley, Reading, Massachusetts, 1964, pp. 3-62, First appeared in the *J. Franklin Inst.* , 257, Nos. 3-4, 1954.

[13] G. G. Langdon, "Analysis of Asynchronous Circuits Under Different Delay Assumptions," *IEEE Transactions on Computers*, Vol. C-17, pp. 1131-1143, Dec. 1968.

[14] T. Lengauer and S. Näher, "Delay-Independent Switch Level Simulation of Digital MOS Circuits," in *VLSI: Algorithms and Architectures*, P. Bertolazzi and F. Luccio, Eds., Elsevier Science Publishers, 1984, pp. 315-326.

[15] E. J. McCluskey, Jr., "Transients in Combinational Logic Circuits," in *Redundancy Techniques for Computing Systems*, Wilcox, R. H. and W. C. Mann, Eds., Spartan Books, Washington, 1962, pp. 9-46.

[16] R. E. Miller, *Switching Theory, Vol. 2*, Wiley, New York, 1965.

[17] D. E. Muller and W. S. Bartky, "A Theory of Asynchronous Circuits," *Proceedings of an International Symposium on the Theory of Switching*, Vol. 29 of the *Annals of the Computation Laboratory of Harvard University*, pp. 204-243, Harvard University Press, 1959.

[18] C-J. Seger and J. A. Brzozowski, "An Optimistic Ternary Simulation of Gate Races," in *Canadian Conference on VLSI*, pp. 67-72, Oct. 1986, also in Technical Report CS-86-22, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.

[19] S. T. Unger, *Asynchronous Sequential Switching Circuits*, Wiley-Interscience, New York, 1969.

[20] M. Yoeli and S. Rinon, "Application of Ternary Algebra to the Study of Static Hazards," *Journal of the Association for Computing Machinery*, Vol. 11, No. 1, pp. 84-97, Jan. 1964.