CIRCUIT SIMULATION USING A HAZARD ALGEBRA

by

Mihaela Gheorghiu

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Master of Mathematics

in

Computer Science

Waterloo, Ontario, Canada, 2001

I hereby declare that I am the sole author of this thesis.

I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Waterloo to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

Hazards are unwanted signal changes that may affect the correctness of computations in asynchronous circuits. For this reason, it is very important to have accurate and efficient methods of hazard detection. Recently, Brzozowski and Ésik proposed a new infinite-valued algebra for the detection of hazards, and a simulation algorithm based on this algebra. Their simulation method seems to be able not only to detect hazards, but also to count all the signal changes that may occur in gate circuits in the worst case. In this thesis we study the correctness of the new simulation method, by comparing it to binary analysis. Binary analysis is a model of behavior for asynchronous circuits that considers all possible behaviors under different delay distributions. We prove that, for any gate circuit started in any state, the result of the binary analysis is covered by the result of the simulation, in the sense that all signal changes occurring in the binary analysis occur also in the simulation. Conversely, we prove that, for feedback-free circuits composed of one- and two-input gates and started in a stable state, the result of the simulation is covered by the result of the binary analysis, in the sense that all the changes predicted by the simulation occur in the binary analysis, if input, wire, and fork delays are taken into consideration.

Acknowledgements

I am thankful to God for giving me the gifts that made this work possible.

One of these gifts is my supervisor, Prof. Janusz Brzozowski. He generously accepted me as his student, and offered me financial support. Moreover, he entrusted me an interesting problem to solve, and stood by my side all the way, helping me every time I had difficulties. My work has benefitted greatly from his numerous ideas and suggestions, and especially from his meticulous proofreading of the uncountably many and often unreadable samples of my writing. Working with him has constantly been challenging me to refine my way of thinking and writing. I am most grateful to Prof. Brzozowski for his understanding, his patience, and his encouragements.

I have also been blessed to have Florin Bobaru as my husband. I know that only his endless love and trust could have made him accept to sacrifice two years of our marriage for my independent growth. I am very grateful to him for that, and for his constant support and confidence.

I am also grateful to the readers of my thesis, Prof. Nancy Day, Prof. Mark Aagaard, and Prof. Zoltan Ésik, for their useful comments and suggestions. My special thanks go to Zoltan Ésik, who also read and commented on earlier versions of my work.

My life in Waterloo would have been far less enjoyable, if I had not met the wonderful people with whom I shared many precious moments. I am especially thankful to Corina, for her sisterly care and closeness. Relu and Valentina have also been a source of warmth and good advice, and their presence gave me a sense of family. Andrei has been a reliable biking, movie going, and politically correct and not-so-correct discussions partner. Dinners with John and Maria Brzozowski have been agreeable and refreshing intermezzos in my routinely unsettling graduate student life. Discussions with Daniel Berry have always been comforting, interesting, and fun. Everybody else who has shared some time with me, be it lunch, dinner, movie, party, tennis game, bike ride, little chit-chat, or anything else, is hereby acknowledged, because these little things made my life so much more bearable.

My family back home has permanently been in my mind during my work. I thank them for accepting my absence, and for their love and support.

The financial support from the Natural Sciences and Engineering Research Council under Grant No. OGP0000871 is gratefully acknowledged.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Asynchronous Circuits

Asynchronous circuits, in contrast to *synchronous circuits*, operate without a global synchronization signal called *clock*. Although synchronous circuits are by far better understood and more widely used than asynchronous ones, interest in asynchronous circuits has grown considerably in recent years, in both academia and industry. Several successful asynchronous microprocessors have been designed and implemented by different academic research groups ([4, 19, 27]). Other projects have been carried out in an industrial setting ([13, 24, 33]). Important contributions have been made to the theory, verification, performance and timing analysis, testing, and synthesis of asynchronous circuits (see, as examples, [2, 7, 15, 25, 32]).

The growing interest in asynchronous circuits is motivated by the potential of these circuits to overcome many of the limitations that are becoming critical as modern synchronous systems become more sophisticated. We list some of these limitations below (for more detailed discussions of these issues, see Chapter 15 of [8], or see [14]).

- **Clock distribution.** As synchronous circuits become more complex, distributing the clock evenly to all circuit components becomes more difficult. Asynchronous circuits are not exposed to this kind or problem, since they do not need a clock.

- **Power consumption.** In synchronous circuits the clock contributes significantly to power consumption, since the clock is distributed throughout the circuit, even to some parts that are not involved in computation. In asynchronous circuits power consumption is local to the computation sites.

- **Speed.** The clock must be slow enough to accommodate the slowest component of a system. For this reason, synchronous circuits exhibit worst-case computation time. In contrast, asynchronous circuits have average-case performance, since each component operates at its own speed.

- **Metastability.** Synchronous circuits may be affected by metastability [12], which is an unstable equilibrium that can last for an unbounded amount of time. Since these circuits require bounded response time, if the metastability is not resolved within one clock cycle, erroneous behavior may result. Asynchronous circuits are more robust in such situations, since components can wait until metastability is resolved.

- **Modularity.** Because of the global synchronization need, synchronous circuits are not easily composable, and do not naturally support modular design. Asynchronous circuits, on the other hand, allow easier modular design.

Despite their advantages over synchronous circuits, asynchronous circuits are subject to problems that can be avoided in synchronous systems. *Hazards* are an example of such a problem. Hazards are unwanted signal changes that may affect the correctness of computations. For instance, a signal that should be 1 at all times during a computation but changes from 1 to 0 and then to 1 has a *static hazard*. A signal that is supposed to change once from 0 to 1, but instead changes from 0 to 1, to 0, and then to 1 has a *dynamic hazard*. Since the components of an asynchronous circuit react to all signal changes on their inputs, incorrect signals due to hazards may propagate through the circuit, causing it to malfunction. Synchronous circuits can be protected from malfunctioning in the presence of hazards by a careful design that ensures all signals have the correct values at each edge of the clock. Even if they do not affect correctness, however, hazards may increase computation time and power consumption in synchronous circuits.

Because the problem of hazards is very important for the operation of asynchronous circuits, much of the research in the field has been dedicated to finding efficient methods of hazard detection. These efforts have been made difficult by the fact that the behavior of asynchronous circuits, as opposed to that of synchronous circuits, is significantly more difficult to analyze.

Our work addresses the general problem of asynchronous circuit analysis, with application to hazard detection in particular. In the next section we survey some of the most important contributions made in the area, and show how our work relates to them.

1.2 Previous Work on Hazards

A widely used model for the behavior of asynchronous circuits was introduced by Muller and Bartky [31]. The model assumes binary signal values and it exhaustively generates all the states of a circuit that are possible as a result of a change in the inputs. While the model captures all possible behaviors under different delay distributions, it is computationally inefficient, since the state space has exponential worst-case complexity. That renders it impractical, for instance for hazard detection. The model is also known as the *General Multiple Winner* (GMW) model ([5, 8]), but we refer to it as the *binary analysis*. We present it in Chapter 2.

Researchers have considered other, more efficient methods of circuit analysis for hazard detection. One of the earliest treatments of hazards in digital circuits is due to Huffman

[23], who introduces informal definitions of the static and dynamic hazards and discusses some characterizations of these hazards. His work was later formalized by McCluskey [28] and generalized by Unger [36]. Their approach uses Boolean algebra and Karnaugh maps to manipulate the Boolean expressions describing circuits, in order to find so-called *lift-* and *drop-sets* that are used to characterize hazards. The procedures are rather complicated and depend very much on the structure of the circuits. Also, the time/space usage of the procedures increases dramatically with the size of the circuits.

As a more efficient alternative to the binary methods mentioned above, several multi-valued algebras have been proposed over the years for the characterization and detection of hazards. We briefly mention some of them here; for a detailed survey we refer the reader to [10].

One of the most successful algebras used for hazard detection is the three-valued algebra that is the basis of the *ternary simulation* algorithm introduced by Eichelberger [16]. This simulation algorithm provides a simple and a very efficient (linear time) method of detecting static hazards and oscillations, but is not capable of detecting dynamic hazards. A complete characterization of the ternary simulation in terms of binary analysis is given in [6]. The characterization states that the simulation provides a *least upper bound* of the result of binary analysis, under the assumption that both gates and wires have arbitrary, but finite delays. As a corollary, it is shown that static hazards and oscillations are correctly detected by simulation. The algorithm, originally defined for stable initial states, is generalized in [35] to handle any initial state.

None of the other multi-valued algebras proposed for hazard analysis provides a simulation algorithm as well defined and well characterized as the ternary simulation. For instance, Metze [29] introduces a four-valued algebra aimed at detecting both static and dynamic hazards, but his simulation method using that algebra is described only by examples, and no algorithm is defined. A five-valued enhancement of the algebra of Metze is proposed by Lewis [26]. This time a simulation algorithm is defined, but in a complicated way, and a rather informal characterization is given.

Hayes [20] makes a more rigorous study of several multi-valued algebras for hazard representation. He shows that larger algebras can be constructed from smaller ones and provides the rules for such constructions. By applying these rules, Hayes obtains, from the two-valued Boolean algebra and the five-valued algebra, an eight-valued algebra capable of representing static and dynamic hazards explicitly. No simulation algorithm is given in [20] for this algebra, but a simulation using an identical algebra is presented in [3]. This simulation is limited to combinational circuits, and no discussion of its results is provided. An extension of this simulation to sequential circuits is reported in [1], but its results are assessed only experimentally. Hayes also constructs a 13-valued algebra, from the ternary and five-valued algebras. This algebra is a more complete version of the eight-valued algebra, since in addition to static and dynamic hazards it explicitly represents unknown behavior. A similar algebra is presented in [11], together with an informal description of a corresponding 13-valued simulation.

Other multi-valued algebras have been proposed as well, such as a nine-valued algebra [18], a 16-valued algebra [21], and a 27-valued algebra [3], but they suffer from serious flaws (see [10]).

In a recent paper [9], Brzozowski and Ésik introduce a general, infinite-valued algebra, the *change-counting* algebra C, that generalizes all the successful multi-valued algebras proposed previously for hazard analysis. On the basis of this algebra, they define a simple polynomial time simulation algorithm that generalizes the ternary simulation. Their simulation seems to be able not only to detect static and dynamic hazards, and oscillations, but also to provide more information about the behavior of circuits. In particular, the algorithm seems capable of counting all the signal changes in gate circuits in the worst case; this could provide an estimation of power consumption.

The purpose of our work is to characterize the simulation algorithm of Brzozowski and Ésik in order to prove that it is correct with respect to binary analysis, in the sense that it accurately predicts all the signal changes that occur in the binary analysis. We prove that all the changes that occur in the binary analysis, also occur in the simulation; in particular, if any hazardous change takes place, the simulation detects it. This result applies to all gate circuits, with any initial state. Conversely, we prove that, for feedback-free circuits started in a stable state, all the changes predicted by the simulation occur in the binary analysis, if input, wire, and fork delays are taken into consideration. The latter result is restricted to circuits of one- and two-input gates. These results make a step toward a full characterization of the new general simulation in the spirit of the one previously given for the ternary simulation.

1.3 Thesis Overview

The thesis is structured as follows.

In the remainder of this chapter we introduce some of the notational conventions that are used throughout the text.

In Chapter 2 we define the *network model* of gate circuits on which our theory is based; we also present the binary analysis, and the algebra C of *transients* introduced by Brzozowski and Ésik.

Chapter 3 introduces the simulation method based on algebra C. We define a more general algorithm than that of [9], in the sense that it does not require the initial state to be stable. Our algorithm is called Algorithm A. We also present Algorithm Ã, that is the original definition of the simulation, with initial stable state; we show that Algorithm Ã is equivalent to Algorithm A, under models containing input delays. We also prove some important properties of these algorithms, namely that Algorithm A is monotonic, and Algorithm Ã is insensitive to wire delays, for feedback-free circuits.

In Chapter 4 we establish one side of the correspondence between binary analysis and simulation by showing that all signal changes occurring in the binary analysis appear also in the result of Algorithm A, for any gate circuit. This result implies that the simulation detects all hazardous signal changes.

The subsequent three chapters are dedicated to proving a converse of the result in Chapter 4, namely that all changes predicted by the simulation take place in binary analysis. This is important in order to ensure that the simulation does not produce false negatives, in the sense that it predicts hazards that never occur. In Chapter 5 we introduce *delay automata* that model the *hazard-preserving* behavior of delays in binary analysis. In Chapter 6 we introduce *gate automata* that characterize the *worst-case* behavior of gates in binary analysis. We prove an important lemma that relates delay automata and gate automata, and is essential to our main proof. The main proof is completed in Chapter 7, where we show by an inductive construction that there exist state sequences in binary analysis that exhibit the behavior predicted by simulation. This result is restricted to Algorithm Ã, applied to feedback-free circuits with one- and two-input gates; it also requires that input, wire, and fork delays be considered in the binary analysis. The reader might find it useful to consult Chapter 7 prior to reading Chapters 5 and 6, in order to better understand the goal of the concepts and results presented in these two chapters.

Chapter 8 concludes the thesis, with a summary of the results and open problems.

1.4 Notational Conventions

We use $[n]$ to denote the set $\{1, \ldots, n\}$, for an integer $n > 0$. The Boolean operations AND, OR, NOT, and XOR are denoted $\wedge$, $\vee$, $\overline{}$, and $\veebar$, respectively.

For a word w, $w\!\downarrow_A$ is its subword determined by the characters belonging to alphabet A, *e.g.*, if $w = cacdabd$, and $A = \{a, b, e\}$, then $w\!\downarrow_A = aab$. We denote by $|w|_a$ the number of occurrences of character a in word w. The empty word is denoted by ϵ. We sometimes write w^n to denote the word obtained by concatenating n copies of word w, for a positive integer n; when $n = 0$, $w^n = \epsilon$. A word u is called a *factor* of a word w, if there exist words s, t such that $w = sut$.

We denote by $\mathcal{L}(\mathcal{A})$ the language recognized by an automaton $\mathcal{A}$. When writing regular expressions, we use $+$ for union, juxtaposition for concatenation, and $*$ for Kleene's star operator.

Whenever possible, we write the steps of our proofs in the form

$$\begin{array}{ll} & P \\ \mathcal{R} & \{\ \mathcal{F}\ \} \\ & Q, \end{array}$$

where P, Q are statements or expressions, $\mathcal{R}$ is a relation such as $=,\ \leq,\ \geq,\ \Rightarrow$, or $\Leftrightarrow$, and $\mathcal{F}$ is a series of facts. These deduction steps should be read as: P is in relation $\mathcal{R}$ with Q, by the facts in $\mathcal{F}$.

Chapter 2

Preliminaries

In this chapter we introduce some basic concepts that underlie our theory. Most of the ideas presented here are taken from [8] and [9].

2.1 Circuit Model

This section defines the mathematical model of circuits that constitutes the basis of our theory. Our model is a variation of the model introduced by Brzozowski and Seger in [8].

We model asynchronous circuits at the gate level of abstraction. We restrict ourselves to gate circuits composed of single-output gates, with no wired-AND, or wired-OR connections (we consider AND or OR gates, respectively, in place of such connections).

Given a gate circuit with n inputs and m gates, we associate a variable to each input, and call it an *input variable* or simply an *input*; we denote the input variables by $X_1, \ldots, X_n$. We also associate a variable with the output of each gate, and call it a *state variable*; we denote the state variables by $s_1, \ldots, s_m$. Input and state variables take values in the binary domain $\mathcal{D} = \{0, 1\}$. Each state variable s_i has an *excitation* S_i, that is the Boolean function implemented by the corresponding gate.

Example 2.1.1 *For the circuit of Figure 2.1, the inputs are X_1 and X_2. The state variables are s_1, s_2, s_3, and s_4, with excitations*

$$S_1 = \overline{X_2}, \quad S_2 = X_2 \wedge s_1, \quad S_3 = s_2 \vee s_3, \quad S_4 = X_1 \veebar s_3.$$

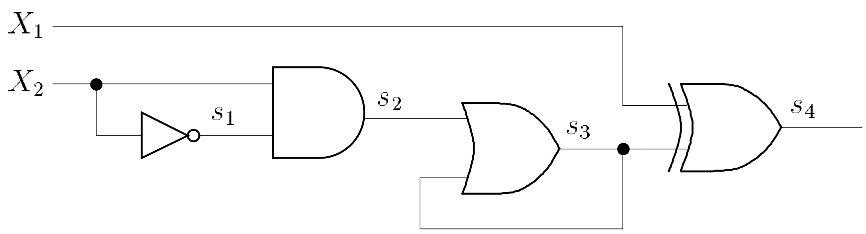


Figure 2.1: Sample gate circuit.

We assume that each excitation depends functionally on all its arguments, in the sense that we define next.

Definition 2.1.1 (Functional dependence) *Let $k > 0$ be an integer. A Boolean function $f : \{0,1\}^k \longrightarrow \{0,1\}$ depends on its ith argument if there exist $b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_k$, $b_j \in \{0,1\}$, for all $j \in [k], j \neq i$, such that*

$$f(b_1, \ldots, b_{i-1}, 0, b_{i+1}, \ldots, b_k) \neq f(b_1, \ldots, b_{i-1}, 1, b_{i+1}, \ldots, b_k).$$

A Boolean function that does not depend on all its arguments can always be rewritten so that it does, *e.g.*, $f(x, y) = x$ is replaced with $f'(x)$, where $f'(x) = f(x, y)$, for all y. That means we remove the wires on the inputs of a gate on which its excitation does not depend, and the behavior of the gate is not changed. For a gate implementing a constant function $f = b$, where $b \in \{0,1\}$, we remove any input wires it might have, and replace the gate with a non-inverting buffer which takes its input from a circuit input X_j that has the constant value b. The excitation of the gate is not constant any more, it is the identity function instead, but the behavior of the gate is unchanged. Note that a state variable with constant excitation can influence the behavior of the circuit when the variable is initially unstable, that is when its current state differs from its current excitation. The variable can change once, and its change can influence other signals in the circuit. This is why we do not eliminate gates with constant excitations.

We also assume that none of the gates has forked inputs, that is multiple inputs coming from the same fork. The excitation of such a gate can always be rewritten in terms of its distinct arguments. For instance, for a three-input gate, with two of its inputs being the same, say x, we replace its excitation $f(x, x, y)$ with $f'(x, y)$, where $f'(x, y) = f(x, x, y)$. This means that we merge the two identical inputs into one input x. The behavior of the gate is unchanged.[1]

Definition 2.1.2 (Network model) *We model a given gate circuit by a tuple*

$$N = \langle \mathcal{D}, \mathcal{X}, \mathcal{S}, \mathcal{E} \rangle,$$

where: $\mathcal{D}$ is the domain of values for the inputs, state variables, and excitations of the circuit; $\mathcal{X} = \{X_1, \ldots, X_n\}$ is the set of inputs; $\mathcal{S} = \{s_1, \ldots, s_m\}$ is the set of state variables with associated excitations $S_1, \ldots, S_m$; $\mathcal{E} \subseteq (\mathcal{X} \times \mathcal{S}) \cup (\mathcal{S} \times \mathcal{S})$ is a set of directed edges, where there is an edge between x and y if and only if the excitation of y depends functionally on x. We call N a network. *Note that N defines a graph $(\mathcal{X} \cup \mathcal{S}, \mathcal{E})$ called the* network graph *(that is a dependency or data flow graph of the circuit).*

Example 2.1.2 *For the circuit of Figure 2.1 the network graph is shown in Figure 2.2.*

Note that a domain different from the binary domain may be chosen. The ternary domain $\{0, \Phi, 1\}$ has been used in [8]. In this thesis we often use the domain $\mathbf{T}$ of *transients* defined in Section 2.5. In general, any multi-valued domain [10] can be used, as long as one can extend each gate function from the binary domain to the multi-valued domain. The excitations in a network with a multi-valued domain are the extensions of the Boolean excitations. Examples of such networks are given in Chapters 3 and 4.

[1] Our assumption is that any wire that is supposed to have a delay will be represented as a gate, *i.e.*, as a non-inverting buffer, and will never be removed. Otherwise, it is safe to remove it.

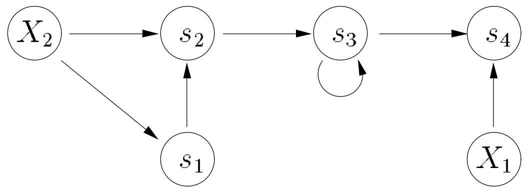


Figure 2.2: Network graph for circuit of Figure 2.1.

2.1.1 Complete Network

In order to take into account wire delays that may be present in a circuit, we also use an augmented network, called a *complete network*, to model the given gate circuit. The complete network is obtained in the following way. We model the input ports and the forks of the circuit as gates, for mathematical convenience. The reasons for doing so will become clear in Chapter 7. An input port becomes a gate with a single input and a single output; we call it an *input gate*. It operates by transferring the input signal to its output. We use the usual symbol for a (non-inverting) buffer (shown in Figure 2.3) as a symbol for an input gate. A k-way fork becomes a gate with a single input and k outputs, as shown

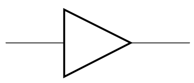


Figure 2.3: Symbol for non-inverting buffer.

in Figure 2.4; we call it a *fork gate*. Note the new symbol for a fork gate. The operation of a fork gate consists of repeating the signal from its input to each of its outputs. Input

(a) Usual fork. (b) Fork gate.

Figure 2.4: A k-way fork.

gates are inherited from the original model introduced by Brzozowski and Seger.[2] Fork gates are new in our model.[3] To obtain the complete network, we keep the state variables of the initial circuit and assign additional state variables to the outputs of input gates and fork gates, and to the wires of the augmented circuit.

Example 2.1.3 *The augmented version of the circuit of Figure 2.1 is shown in Figure 2.5. We label input-gate, fork-gate and wire variables with subscripted i, f, and w labels, respectively.*

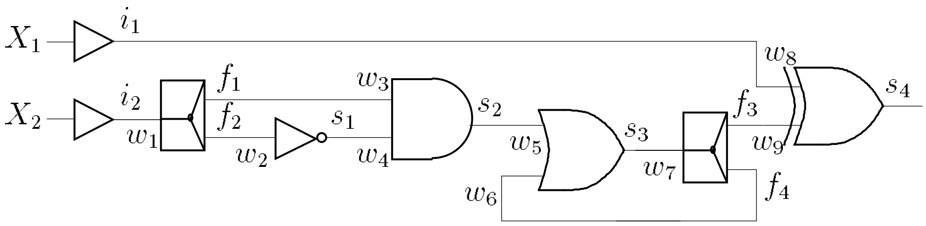
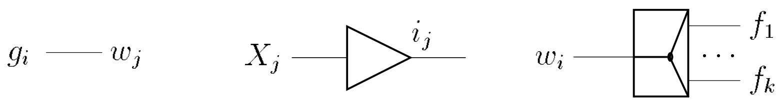


Figure 2.5: Augmented version of circuit of Figure 2.1.

The excitations of the added state variables are identity functions. Figure 2.6 shows generic wire, input-gate, and fork-gate variables with their input variables. The corre-



(a) Wire. (b) Input gate. (c) Fork gate.

Figure 2.6: Variables with identity functions.

sponding excitations are:

$$W_j = g_i, \quad I_j = X_j, \quad F_1 = \ldots = F_k = w_i,$$

[2]In [8] input gates are called "input delays"; we find the name "input gates" more appropriate for our approach.

[3]The use of fork gates was suggested by J. A. Brzozowski.

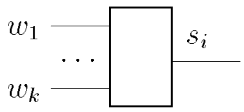
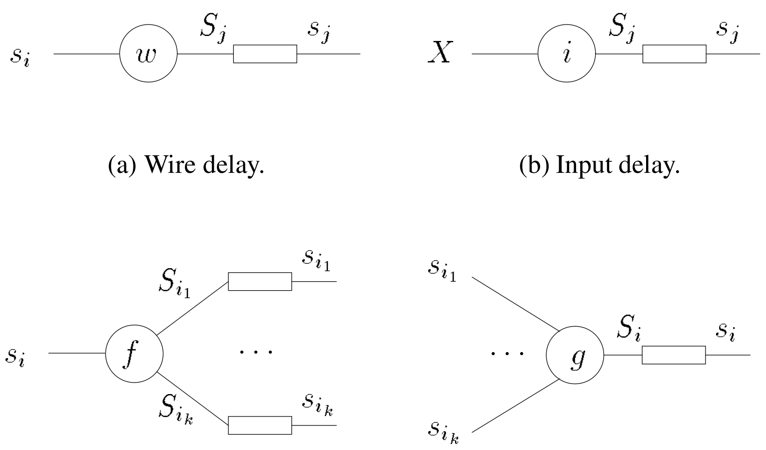


Figure 2.7: Gate variable.

where g_i represents an input-gate, fork-gate or any other gate variable. For a gate variable other than that of an input gate or a fork gate, the excitation is the Boolean function implemented by that gate applied to the new wire variables on its input wires. Figure 2.7 shows a generic gate with its input wire variables $w_1, \ldots, w_k$. Assuming the gate implements function g, its excitation is $S_i = g(w_1, \ldots, w_k)$. The initial and added state variables together with their excitations determine the complete network.

2.2 Delay Model

It is important to notice that, by differentiating between a state variable and its excitation, we associate a delay with that state variable. This interpretation of state variables is depicted in Figure 2.8. Each component is seen as consisting of two parts: a delay-free gate realizing the excitation function and a delay element. The value of a state variable is the output of the delay to which its excitation is the input. We assume the delays are *inertial*, *i.e.*, very short pulses of the excitation may not be transferred into the state. This is why we cannot consider the excitation as the next-state function. Figure 2.9 illustrates the difference between inertial and *ideal* delays. An ideal delay does not alter the shape of its input waveforms, only their timing, whereas an inertial delay tends to smooth out the signals, by ignoring very short pulses. The inertial delay model is more realistic, since physical delays do have an inertial behavior.



(a) Wire delay. (b) Input delay.

(c) Fork delays. (d) Gate delay.

Figure 2.8: Delays for state variables.

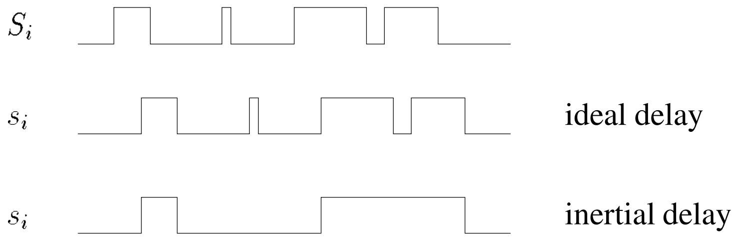


Figure 2.9: Inertial versus ideal delay.

2.3 Terminology Related to Networks

In the following we introduce, for later use, more terminology related to networks. Let N be a network. For any state variable $s_i \in \mathcal{S}$, we define its *fan-in set* $\phi(s_i)$ as

$$\phi(s_i) = \{s_j \mid s_j \in \mathcal{S}, (s_j, s_i) \in \mathcal{E}\},$$

and its *fan-out set* $\psi(s_i)$ as

$$\psi(s_i) = \{s_j \mid s_j \in \mathcal{S}, (s_i, s_j) \in \mathcal{E}\}.$$

The variables in $\phi(s_i)$ are called the *fan-in variables* of s_i and those in $\psi(s_i)$, the *fan-out variables* of s_i.

For any two state variables s_i and s_j we say that s_i *precedes* s_j if $s_i\, \mathcal{E}^+\, s_j$, where $\mathcal{E}^+$ is the transitive closure of $\mathcal{E}$, with $\mathcal{E}$ seen as a binary relation. We then say that s_i is a *predecessor* of s_j and s_j a *successor* of s_i. If s_i does not precede s_j, and s_j does not precede s_i, then we say that s_i and s_j are *unrelated by the precedence relation*.

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* of N is an $(n + m)$-tuple $c = a \cdot b$ of values from $\mathcal{D}$, the first n values (the n-tuple a) being the values of the inputs, and the remaining m values (the m-tuple b) being the values of the state variables. The dot " $\cdot$ " is used for convenience, to separate inputs from state variables.

Each excitation S_i is a function of some inputs $X_{j_1}, \ldots, X_{j_l} \in \mathcal{X}$, and some state variables $s_{i_1}, \ldots, s_{i_k} \in \mathcal{S}$, *i.e.*,

$$S_i = f(X_{j_1}, \ldots, X_{j_l}, s_{i_1}, \ldots, s_{i_k}),$$

where $f : \mathcal{D}^{l+k} \longrightarrow \mathcal{D}$. It is often convenient to treat S_i as a function from $\mathcal{D}^{n+m}$ into $\mathcal{D}$. Thus we define $\tilde{S}_i : \mathcal{D}^{n+m} \longrightarrow \mathcal{D}$ by

$$\tilde{S}_i(a \cdot b) = f(a_{j_1}, \ldots, a_{j_l}, b_{i_1}, \ldots, b_{i_k}),$$

for any total state $a \cdot b$. From now on we write S_i for $\tilde{S}_i$; it is clear from the context which definition we use.

For any $i \in [m]$, the value of excitation S_i in total state $a \cdot b$ is denoted $S_i(a \cdot b)$. The values $S_1(a \cdot b), \ldots, S_m(a \cdot b)$ of all excitations in any total state $a \cdot b$ constitute an m-tuple denoted $S(a \cdot b)$. For any total state $a \cdot b$, we define the set $U(a \cdot b)$ of unstable state variables as those variables for which the current state differs from the current excitation. Formally,

$$U(a \cdot b) = \{s_i \mid b_i \neq S_i(a \cdot b)\}.$$

Therefore, state $a \cdot b$ is *stable* if and only if $U(a \cdot b) = \emptyset$, *i.e.*, $S(a \cdot b) = b$.

2.4 Analysis of Binary Networks

In response to changes of its inputs, a circuit passes through a sequence of states as its internal signals change. Analyzing the behavior of a circuit means exploring all sequences of states that are possible as a result of a change in the inputs. This section describes a formal model of analysis called the *General Multiple Winner* (GMW) model. This model was originally introduced in [31] and later used in [5, 8, 30] and elsewhere. Our presentation of the model follows that of [8]. We refer to the GMW model as *binary analysis*.

A binary network is a network $N = \langle \mathcal{D}, \mathcal{X}, \mathcal{S}, \mathcal{E} \rangle$ having the binary domain, *i.e.*, $\mathcal{D} = \{0, 1\}$. A total state $a \cdot b$ of network N is an $(n + m)$-tuple of binary values, that is $a \cdot b \in \{0, 1\}^{n+m}$.

We describe how the behavior of a network evolves when it is started in a given initial state and the input is kept constant at the 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 $b \in \{0, 1\}^m$,

$bR_a b$, if $U(a \cdot b) = \emptyset$, *i.e.*, total state $a \cdot b$ is stable,

$bR_a b^K$, if $U(a \cdot b) \neq \emptyset$, and K is any nonempty subset of $U(a \cdot b)$,

where by b^K we mean b with all the variables in K complemented. No other pairs of states are related by R_a. The relation R_a is called the *General Multiple Winner* (GMW) relation (see [8] for more details).

We associate a directed graph $G_a = (V_a, E_a)$ to the R_a relation, where

$V_a = \{s \mid s = s_1 \ldots s_m, (s_1, \ldots, s_m) \in \{0, 1\}^m\}$,

i.e., the elements of V_a are the word equivalents of binary state tuples, and

$E_a = \{(s, t) \mid s \in V_a, t \in V_a, (s_1, \ldots, s_m) R_a (t_1, \ldots, t_m)\}$.

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 $reach(R_a(b))$. We refer to graph $G_a(b)$ as the *result of binary analysis* for N started in total state $a \cdot b$.

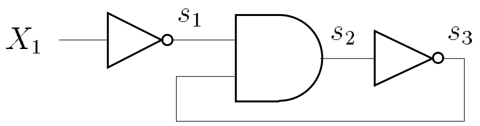


Figure 2.10: Sample circuit for binary analysis.

Example 2.4.1 *For the circuit in Figure 2.10 the excitation functions are:*

$$S_1 = \overline{X_1}, \quad S_2 = s_1 \wedge s_3, \quad S_3 = \overline{s_2}.$$

Suppose we start in total state $a \cdot b$, where $a = X_1 = 0$ and $b = 000$. The corresponding $G_0(000)$ graph is shown in Figure 2.11(a). The unstable variables are underlined in each state. Note that the graph contains no stable states. Another graph, $G_1(111)$, is shown in Figure 2.11(b). This time the graph has one stable state.

While binary analysis is an exhaustive analysis of the behavior of a circuit, it is computationally inefficient, since the state space is exponential in the worst case. Simulation using multi-valued domains is a more efficient alternative, if not all the information from binary analysis is needed.

2.5 Transients

Transients were introduced in [9] to represent signal waveforms in an algebraic setting. The set $\mathbf{T}$ of transients is defined as the set of all nonempty binary words over $\{0, 1\}$ in which no two consecutive symbols are the same, *i.e.*,

$$\mathbf{T} = 0(10)^* \cup 1(01)^* \cup 0(10)^*1 \cup 1(01)^*0.$$

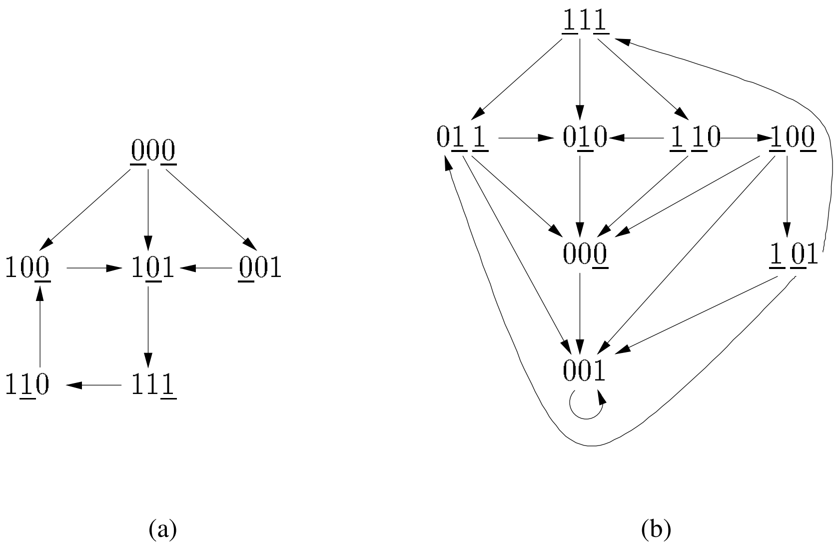


Figure 2.11: Sample $G_a(b)$ graphs for circuit of Figure 2.10.

Transients are associated to waveforms in the natural way, as shown in the examples of Figure 2.12.

Figure 2.12: Transients as words for waveforms.

We use boldface symbols to denote transients, tuples of transients, and functions of transients. For any transient $\mathbf{t}$ we denote by $\alpha(\mathbf{t})$ and $\omega(\mathbf{t})$ its first and last characters, respectively. A transient can be obtained from any nonempty binary word by *contraction*, *i.e.*, the elimination of all duplicates immediately following a symbol (*e.g.*, the contraction of 000110010011 is 010101). For a binary word s we denote by $\hat{s}$ the result of its contraction. For any $\mathbf{t}, \mathbf{t}' \in \mathbf{T}$, we denote by $\mathbf{t}\mathbf{t}'$ the concatenation of $\mathbf{t}$ and $\mathbf{t}'$.

Definition 2.5.1 (Prefix and suffix) *Two partial orders are considered on* $\mathbf{T}$*: the* prefix *order denoted* $\leq$*, and the* suffix *order denoted* $\preceq$*. For any two transients* $\mathbf{t}$ *and* $\mathbf{t}'$*,* $\mathbf{t}$ *is a* *prefix of* $\mathbf{t}'$ *if* $\mathbf{t} = \mathbf{t}'$*, or there exists a transient* $\mathbf{t}''$ *such that* $\mathbf{t}' = \mathbf{t}\mathbf{t}''$*;* $\mathbf{t}$ *is a suffix of* $\mathbf{t}'$ *if* $\mathbf{t} = \mathbf{t}'$*, or there exists a transient* $\mathbf{t}''$ *such that* $\mathbf{t}' = \mathbf{t}''\mathbf{t}$*.*

In this thesis we use only the prefix order, described by the following inequalities, together with the reflexive and transitive laws:

$$0 \le 01 \le 010 \le 0101 \le 01010 \le \ldots,$$

$$1 \le 10 \le 101 \le 1010 \le 10101 \le \ldots.$$

We extend the prefix order to tuples of transients.

Definition 2.5.2 (Prefix on tuples) *For* $\mathbf{u} = (\mathbf{u}_1, \ldots, \mathbf{u}_m), \mathbf{v} = (\mathbf{v}_1, \ldots, \mathbf{v}_m) \in \mathbf{T}^m$*, we say that* $\mathbf{u}$ *is a prefix of* $\mathbf{v}$ *and write* $\mathbf{u} \le \mathbf{v}$*, if* $\mathbf{u}_i \le \mathbf{v}_i$*, for all* $i \in [m]$*.*

Extensions of Boolean functions to functions of transients are defined in [9]. Any Boolean function $f : B^n \to B$ is extended to a function $\mathbf{f} : \mathbf{T}^n \to \mathbf{T}$ so that, for any tuple $(\mathbf{t}_1, \ldots, \mathbf{t}_n)$ of transients, $\mathbf{f}$ produces the longest transient when $\mathbf{t}_1, \ldots, \mathbf{t}_n$ are applied to the inputs of a gate performing the Boolean function f. We give an example of extended Boolean function next.

Example 2.5.1 *We take* f *to be the two-input Boolean OR and we want to find its extension* $\mathbf{f}$*. Suppose we want to compute* $\mathbf{f}(01, 010)$*. We construct a directed graph* $D(01, 010)$ *in which the nodes consist of all the pairs* $(\mathbf{t}, \mathbf{t}')$ *of transients such that* $(\mathbf{t}, \mathbf{t}') \le (01, 010)$*, and there is an edge between any two pairs* $\mathbf{p}, \mathbf{p}'$ *such that* $\mathbf{p}'$*, compared to* $\mathbf{p}$*, has one additional character in one coordinate, all other coordinates being the same. The resulting graph is shown in Figure 2.13(a). For each node* $(\mathbf{t}, \mathbf{t}')$ *in the graph we consider as its label the value* $f(\omega(\mathbf{t}), \omega(\mathbf{t}'))$*. This results in a graph of labels, as shown in Figure 2.13(b). The value of* $\mathbf{f}(01, 010)$ *is the contraction of the label sequence of those paths in the graph of labels that have the largest number of alternations between* 0 *and* 1*. Therefore,* $\mathbf{f}(01, 010) = 0101$*.*

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

(a) (b)

Figure 2.13: Graph $D(01, 010)$ with labels.

In general, to evaluate $\mathbf{f}(\mathbf{t}_1, \ldots, \mathbf{t}_n)$ we construct graph $D(\mathbf{t}_1, \ldots, \mathbf{t}_n)$ whose nodes are tuples $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ such that $(\mathbf{x}_1, \ldots, \mathbf{x}_n) \leq (\mathbf{t}_1, \ldots, \mathbf{t}_n)$ and there is an edge between any two tuples $\mathbf{x}, \mathbf{y}$ such that $\mathbf{y}$, compared to $\mathbf{x}$, has one additional character in one coordinate, all other coordinates being the same. To any node $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ we associate the label $f(\omega(\mathbf{x}_1), \ldots, \omega(\mathbf{x}_n))$ and obtain a graph of labels. The value of $\mathbf{f}(\mathbf{t}_1, \ldots, \mathbf{t}_n)$ is the contraction of the label sequence of those paths in the graph of labels that have the largest number of alternations between 0 and 1.

Let $z(\mathbf{t})$ and $u(\mathbf{t})$ denote the number of 0s and the number of 1s in a transient $\mathbf{t}$, respectively. We denote by $\otimes$ and $\oplus$ the extensions, defined as above, of the Boolean AND and OR operations, respectively. It is shown in [9] that for any $\mathbf{w}, \mathbf{w}' \in \mathbf{T}$ of length > 1, $\mathbf{w} \otimes \mathbf{w}' = \mathbf{t}$, where $\mathbf{t} \in \mathbf{T}$ is such that

$$\alpha(\mathbf{t}) = \alpha(\mathbf{w}) \wedge \alpha(\mathbf{w}'),$$
$$\omega(\mathbf{t}) = \omega(\mathbf{w}) \wedge \omega(\mathbf{w}'), \text{ and}$$
$$u(\mathbf{t}) = u(\mathbf{w}) + u(\mathbf{w}') - 1.$$

Similarly, $\mathbf{w} \oplus \mathbf{w}' = \mathbf{t}$, where $\mathbf{t} \in \mathbf{T}$ is such that

$$\alpha(\mathbf{t}) = \alpha(\mathbf{w}) \vee \alpha(\mathbf{w}'),$$
$$\omega(\mathbf{t}) = \omega(\mathbf{w}) \vee \omega(\mathbf{w}'), \text{ and}$$
$$z(\mathbf{t}) = z(\mathbf{w}) + z(\mathbf{w}') - 1.$$

If one of the arguments is 0 or 1 (*i.e.*, a transient of length 1) the following rules apply for any $\mathbf{t} \in \mathbf{T}$:

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

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

A complement operation is also defined; the complement $\overline{\mathbf{t}}$ of any $\mathbf{t} \in \mathbf{T}$ is obtained by complementing each character of $\mathbf{t}$. For example, $\overline{1010} = 0101$.

The set $\mathbf{T}$, together with the operations $\otimes$, $\oplus$ and $^{-}$, and the constants 0 and 1 constitute an algebra $C = (\mathbf{T}, \otimes, \oplus, ^{-}, 0, 1)$, called the *change-counting algebra*, which is a commutative de Morgan bisemigroup [9].

For $\mathbf{t}, \mathbf{t}' \in \mathbf{T}$, we denote by $\mathbf{t} \circ \mathbf{t}'$ the concatenation followed by contraction, *i.e.*, $\mathbf{t} \circ \mathbf{t}' = \widehat{\mathbf{t}\mathbf{t}'}$.

Remark 2.5.1 *The operation $\circ$ satisfies the following properties, for all $\mathbf{t}, \mathbf{t}', \mathbf{t}'' \in \mathbf{T}$ and $b \in \{0, 1\}$:*

1. *associativity, i.e., $(\mathbf{t} \circ \mathbf{t}') \circ \mathbf{t}'' = \mathbf{t} \circ (\mathbf{t}' \circ \mathbf{t}'')$*
   (from now on we omit parentheses and write $\mathbf{t} \circ \mathbf{t}' \circ \mathbf{t}''$);
2. *if $\mathbf{t} \leq \mathbf{t}'$ then $b \circ \mathbf{t} \leq b \circ \mathbf{t}'$;*
3. *$\mathbf{t}_1 \circ \ldots \circ \mathbf{t}_n = \widehat{\mathbf{t}_1 \ldots \mathbf{t}_n}$.*

2.6 Conclusions

In this chapter we have presented some concepts that are fundamental to the theory we develop in subsequent chapters. We have introduced network models of gate circuits, with inputs, state variables, and excitations, in which state variables may be assigned either only to gates, or to gates, input ports, forks, and wires. We have also introduced our delay model that assumes inertial delays associated to state variables. We have described the binary analysis method that exhaustively explores all state sequences that are possible from a given initial state of a network. We have also described the algebra of transients, that is an infinite-value algebra for representing signal waveforms, and provides extensions of the Boolean functions. Most of the notation introduced here is summarized in the glossary on page 112.

Chapter 3

Simulation with Algebra C

A simulation algorithm using algebra C has been proposed in [9]; it generalizes ternary simulation [8]. The algorithm is shown in [9] to have polynomial time complexity in the number of inputs and state variables, for feedback-free gate circuits. Hence, this simulation provides a more efficient method of analysis of circuits, compared to binary analysis. The simulation, however, does not provide all the information that is provided by binary analysis about the behavior of a circuit.

In this section we give a more general version of the simulation algorithm proposed in [9] and show how it relates to the original version. This extension parallels the extension of ternary the simulation from stable initial state to any initial state, given in [8, 35]. In subsequent chapters we study the correctness of the simulation by comparing it to binary analysis. The aim is to show that the simulation accurately predicts the changes that happen to state variables in binary analysis.

Given any circuit, we use two networks to model it: a binary network

$$N = \langle \{0,1\}, \mathcal{X}, \mathcal{S}, \mathcal{E} \rangle$$

and its counterpart

$$\mathbf{N} = \langle \mathbf{T}, \mathcal{X}, \mathcal{S}, \mathcal{E} \rangle$$

having set $\mathbf{T}$ of transients as the domain. We call $\mathbf{N}$ a *transient network*. The two networks have the same inputs and the same state variables. A state of network $\mathbf{N}$ is a tuple of transients; the value of the excitation of a variable is also a transient. Excitations in $\mathbf{N}$ are the extensions of the Boolean excitations in N, as defined for algebra C. It is shown in [9] that an extended Boolean function depends on one of its arguments if and only if the corresponding Boolean function depends on that argument. Therefore N and $\mathbf{N}$ have the same set of edges.

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

Example 3.0.1 *For the circuit of Figure 2.1 on page 10 the transient network has inputs* $\mathbf{X}_1$ *and* $\mathbf{X}_2$ *and state variables* $\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4$*, with excitations*

$$\mathbf{S}_1 = \overline{\mathbf{X}_2}, \quad \mathbf{S}_2 = \mathbf{X}_2 \otimes \mathbf{s}_1, \quad \mathbf{S}_3 = \mathbf{s}_2 \oplus \mathbf{s}_3, \quad \mathbf{S}_4 = \mathbf{X}_1 \oslash \mathbf{s}_3,$$

where we denote by $\oslash$ *the extension to transients of the Boolean XOR operation* $\veebar$.

3.1 General Simulation: Algorithm A

Our simulation method using algebra C is a generalization of Algorithm A of [9] that, in turn, generalizes Algorithm A of ternary simulation [8]. In our Algorithm A we start with a binary total state of a network $\mathbf{N}$ (*i.e.*, each variable is either 0 or 1), and at each step we change all the unstable variables to follow their excitations. For example, suppose we have a variable with current state 0 and excitation 1; in this case we want to set the new value of this variable to 01, the transient that shows this variable changes once from 0 to 1. The changes in the excitation also accumulate in a transient that can only become longer, as we will prove shortly. If the excitation for the same variable next becomes 101, for instance, we want to set the variable to 0101, to mirror the changes in the excitation.

In general, we want to record in the value of a variable all the changes in that variable since the start of the simulation, as dictated by its excitation. For variables that are stable initially, since the initial state agrees with the initial excitation, the state transient and the excitation transient will be the same, so at each step we just copy the excitation into the variable. For example, with initial state 0 and excitation 0, if the excitation becomes next 01, we set the variable to 01, and so on. For variables that are initially unstable, we first record the initial state, and then the excitation. This is just the case we outlined in the previous paragraph, with initial state 0 and excitation 1. In this case, at each step we append the excitation to the initial value. The operator that gives us the desired result in both cases is $\circ$; thus we have *new_value* $=$ *initial_value* $\circ$ *excitation*.

Let $a \cdot b$ be a (binary) total state of $\mathbf{N}$. The general algorithm of the simulation is defined as follows.

Definition 3.1.1 (Algorithm A)

Algorithm A

begin

$\mathbf{s}^0 := b;$

$h := 1;$

$\mathbf{s}^h := b \circ \mathbf{S}(a \cdot \mathbf{s}^0);$

while $(\mathbf{s}^h <> \mathbf{s}^{h-1})$ **do**

$h := h + 1;$

$\mathbf{s}^h := b \circ \mathbf{S}(a \cdot \mathbf{s}^{h-1});$

od

end

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

Algorithm A produces a sequence

$$\mathbf{s}^0, \mathbf{s}^1, \ldots, \mathbf{s}^h, \ldots$$

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

It is shown in [9] that any extended Boolean function $\mathbf{f} : \mathbf{T}^m \longrightarrow \mathbf{T}$ is *monotonic* with respect to the prefix order, *i.e.*, for any $\mathbf{x}, \mathbf{y} \in \mathbf{T}^m$, if $\mathbf{x} \leq \mathbf{y}$, then $\mathbf{f}(\mathbf{x}) \leq \mathbf{f}(\mathbf{y})$.

Proposition 3.1.1 *The sequence resulting from Algorithm A is monotonic or nondecreasing with respect to the prefix order, that is, for all $h \geq 0$, $\mathbf{s}^h \leq \mathbf{s}^{h+1}$.*

Proof: Since extended Boolean functions are monotonic with respect to the prefix order, it follows that excitations in $\mathbf{N}$ are monotonic with respect to the prefix order. We prove the proposition by induction on h.

Basis, $h = 0$. For any $i \in [m]$,

$\mathbf{s}_i^0$

= { definition of Algorithm A}

b_i

$\leq$ { Remark 2.5.1 }

$b_i \circ \mathbf{S}_i(a \cdot \mathbf{s}^0)$

= { definition of Algorithm A }

$\mathbf{s}_i^1$.

Induction hypothesis: $\mathbf{s}^{h-1} \leq \mathbf{s}^h$ for some $h \geq 1$.

Induction step. For any $i \in [m]$,

$\mathbf{s}_i^h$

= { definition of Algorithm A}

$b_i \circ \mathbf{S}_i(a \cdot \mathbf{s}^{h-1})$

$\leq$ { induction hypothesis, monotonicity of excitation, and Remark 2.5.1 }

$b_i \circ \mathbf{S}_i(a \cdot \mathbf{s}^{h})$

= { definition of Algorithm A }

$\mathbf{s}_i^{h+1}$.

Thus, the claim holds for all $h \geq 0$. □

We refer to the property stated in Proposition 3.1.1 as the *monotonicity* of Algorithm A.

For feedback-free circuits, the sequence resulting from Algorithm A is finite. We can easily see this if we order the state variables using levels as follows: level 0 consists of all state variables that depend only on external inputs; level 1 is comprised of all state variables whose fan-in variables belong to level 0, and in general level l consists of all state variables whose fan-in variables belong to levels $< l$, and which have at least one fan-in variable in level $l-1$. Such an ordering is possible since there is no feedback (*i.e.*, there are no loops). Since the inputs do not change during simulation, level-0 variables change at most once, in the first step of Algorithm A. Level-1 variables change at most twice, in the first and second step of the simulation, because after that none of the inputs or the variables in level 0 changes. In general, level-i variables change at most $i+1$ times. Since the number of levels is finite, our claim follows. This analysis also shows that the running time of the algorithm for feedback-free circuits is polynomial in the number of state variables.

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

Example 3.1.1 *Consider the feedback-free circuit in Figure 3.1. The excitations are:*

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

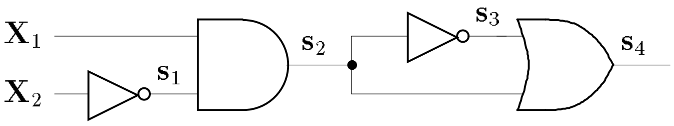


Figure 3.1: Circuit with finite simulation.

For the initial state $a \cdot b = 11 \cdot 1011$*, Algorithm A results in Table 3.1.*

Table 3.1: Result of Algorithm A.

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

For circuits with feedback the simulation sequence may be infinite because, in a loop, a change in a variable can eventually trigger another change in the same variable, and so on.

Example 3.1.2 *Consider the circuit with feedback in Figure 3.2. The excitation functions are:*

$$\mathbf{S}_1 = \mathbf{X}_1 \otimes \mathbf{s}_2, \quad \mathbf{S}_2 = \overline{\mathbf{s}_1}.$$

We run Algorithm A for this network started in state $a \cdot b = 1 \cdot 10$*; the resulting sequence of states, which is infinite, is illustrated in Table 3.2.*

3.2 Simulation with Stable Initial State: Algorithm Ã

Algorithm A above makes no assumptions about the starting state $a \cdot b$. If the network starts in a stable total state and the inputs change, then we have a slightly simpler formulation of

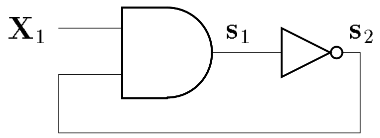


Figure 3.2: Circuit with infinite simulation.

Table 3.2: Infinite simulation.

$\mathbf{X}_1$	$\mathbf{s}_1$	$\mathbf{s}_2$	state
1	1	0	$\mathbf{s}^0$
1	10	0	$\mathbf{s}^1$
1	10	01	$\mathbf{s}^2$
1	101	01	$\mathbf{s}^3$
1	101	010	$\mathbf{s}^4$
1	1010	010	$\mathbf{s}^5$
1	1010	0101	$\mathbf{s}^6$
$\dots$	$\dots$	$\dots$	$\dots$

Algorithm A, in which, at each step, each variable takes the value of its excitation; we call this version Algorithm Ã; this is the version given in [9]. Assume $\mathbf{N}$ is started in stable total state $\tilde{a} \cdot b$ and the input tuple changes to a. Algorithm Ã is defined in Definition 3.2.1. The algorithm results in a sequence

$$\tilde{\mathbf{s}}^0, \tilde{\mathbf{s}}^1, \dots, \tilde{\mathbf{s}}^h, \dots,$$

of states, where $\tilde{\mathbf{s}}^h = (\tilde{\mathbf{s}}_1^h, \tilde{\mathbf{s}}_2^h, \dots, \tilde{\mathbf{s}}_m^h) \in \mathbf{T}^m$, for all $h \geq 0$.

Example 3.2.1 *We illustrate Algorithm Ã with the network in Figure 3.1, started in (stable) state* $\tilde{a} \cdot b = 11 \cdot 0011$*, with the input changing to* 10*. The result is shown in Table 3.3.*

Definition 3.2.1 (Algorithm Ã)

Algorithm Ã

<u>**begin**</u>

$\mathbf{a} = \tilde{a} \circ a;$

$\tilde{\mathbf{s}}^0 := b;$

$h := 1;$

$\tilde{\mathbf{s}}^h := \mathbf{S}(\mathbf{a} \cdot \tilde{\mathbf{s}}^0);$

<u>**while**</u> $(\tilde{\mathbf{s}}^h <> \tilde{\mathbf{s}}^{h-1})$ <u>**do**</u>

$h := h + 1;$

$\tilde{\mathbf{s}}^h := \mathbf{S}(\mathbf{a} \cdot \tilde{\mathbf{s}}^{h-1});$

<u>**od**</u>

<u>**end**</u>

where $\circ$ *is applied component-wise.*

Table 3.3: Simulation using Algorithm Ã.

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

It is shown in [9] that the sequence of states resulting from Algorithm Ã is nondecreasing with respect to the prefix order, *i.e.*, Algorithm Ã is monotonic.

The following result shows that Algorithms A and Ã are equivalent for any network $\mathbf{N}$ started in a stable state, provided that $\mathbf{N}$ contains input-gate variables.

Proposition 3.2.1 *Let* N *be a transient network containing input-gate variables. Let*

$$\tilde{\mathbf{s}}^0, \tilde{\mathbf{s}}^1, \ldots, \tilde{\mathbf{s}}^h, \ldots$$

be the sequence of states produced by Algorithm $\tilde{\mathrm{A}}$ *for this network started in the stable (binary) total state* $\tilde{a} \cdot b$ *with the input tuple changing to* a*. Then, for all* $h \geq 0$*,*

$$\tilde{\mathbf{s}}^h = \mathbf{s}^h,$$

where

$$\mathbf{s}^0, \mathbf{s}^1, \ldots, \mathbf{s}^h, \ldots$$

is the sequence of states produced by Algorithm A for the same network started in total state $a \cdot b$*.*

Proof: We prove the proposition by induction on h.

Basis, $h = 0$. Since $\mathbf{s}^0 = b = \tilde{\mathbf{s}}^0$, the basis holds.

First step, $h = 1$. In states $\tilde{\mathbf{s}}^0$ and $\mathbf{s}^0$ only input-gate variables can be unstable; therefore only they can change in the first step of Algorithm $\tilde{\mathrm{A}}$, and in the first step of Algorithm A. Let $\mathbf{s}_i$ be any input-gate variable; suppose $\mathbf{S}_i = \mathbf{X}_j$. Then the value of $\mathbf{s}_i$ after the first step of Algorithm $\tilde{\mathrm{A}}$ is $\tilde{\mathbf{s}}_i^1 = \mathbf{S}_i(\mathbf{a} \cdot b) = \mathbf{a}_j = \tilde{a}_j \circ a_j$. The value of $\mathbf{s}_i$ after the first step of Algorithm A is $\mathbf{s}_i^1 = b_i \circ \mathbf{S}_i(a \cdot b) = \tilde{a}_j \circ a_j$, where $b_i = \tilde{a}_j$ by the stability of $\tilde{a} \cdot b$. Therefore $\tilde{\mathbf{s}}_i^1 = \mathbf{s}_i^1$, for any input-gate variable $\mathbf{s}_i$. For any other state variable $\mathbf{s}_j$, since $\mathbf{s}_j$ does not change, $\tilde{\mathbf{s}}_j^1 = \tilde{\mathbf{s}}_j^0 = \mathbf{s}_j^0 = \mathbf{s}_j^1$, by the basis. Hence $\tilde{\mathbf{s}}^1 = \mathbf{s}^1$.

Induction hypothesis: $\tilde{\mathbf{s}}^{h-1} = \mathbf{s}^{h-1}$ for some $h > 1$.

Induction step. For any $i \in [m]$, if $\mathbf{s}_i$ is an input-gate variable then $\mathbf{s}_i^h = \mathbf{s}_i^{h-1}$ and $\tilde{\mathbf{s}}_i^h = \tilde{\mathbf{s}}_i^{h-1}$, because in both algorithms the input-gate variables do not change after the first step, since the input is constant. By the induction hypothesis, we have $\mathbf{s}_i^h = \tilde{\mathbf{s}}_i^h$. If $\mathbf{s}_i$ is not an input-gate variable, then it is initially stable in both algorithms, and its excitation does not depend on the input tuple, *i.e.*, $\mathbf{S}_i(a \cdot \mathbf{x}) = \mathbf{S}_i(\mathbf{a} \cdot \mathbf{x})$, for any (internal) state tuple $\mathbf{x}$.

Then

$$\begin{array}{ll} & \mathbf{s}_i^h \\ = & \{ \text{ definition of Algorithm A, and stability of } \mathbf{s}_i \ \} \\ & \mathbf{S}_i(a \cdot \mathbf{s}^{h-1}) \\ = & \{ \text{ induction hypothesis and observation above } \} \\ & \mathbf{S}_i(\mathbf{a} \cdot \tilde{\mathbf{s}}^{h-1}) \\ = & \{ \text{ definition of Algorithm } \tilde{\text{A}} \ \} \\ & \tilde{\mathbf{s}}_i^h. \end{array}$$

Hence $\tilde{\mathbf{s}}_i^h = \mathbf{s}_i^h$, for all $i \in [m]$. □

3.3 Simulation of Feedback-Free Circuits without Wire Delays

In later chapters we often use Algorithm Ã with the complete network modeling a circuit, for mathematical convenience. Recall that a complete network contains input-gate, fork-gate, and wire variables that are added to the initial set of state variables whose behavior we need to study. We show that the result of Algorithm Ã with respect to the initial set of state variables is the same if we use the complete network. We prove this by showing that Algorithm Ã is insensitive (in a sense that we define formally) to the removal of state variables having identity excitation functions. We restrict our attention to feedback-free circuits.

The different network notations $\dot{\mathbf{N}}, \tilde{\mathbf{N}}$ defined here are local to this section, and must not be remembered later.

Let $\mathbf{N} = \langle \mathbf{T}, \mathcal{X}, \mathcal{S}, \mathcal{E} \rangle$ be a feedback-free transient network with m state variables and n inputs. We assume that $\mathbf{N}$ contains at least one state variable whose excitation is the identity function. Without loss of generality, we assume the last state variable, $\mathbf{s}_m$, is such a variable. We can always renumber the variables in such a way that the assumption holds. Let $\mathbf{S}_m = \mathbf{x}$, where $\mathbf{x} \in \mathcal{X} \cup \mathcal{S} \setminus \{\mathbf{s}_m\}$. Since $\mathbf{N}$ is feedback-free, $\mathbf{S}_m$ does not depend on $\mathbf{s}_m$. Hence, for any total state $\mathbf{c} \in \mathbf{T}^{n+m}$ of $\mathbf{N}$ and for all $i \in [m]$, we have $\mathbf{S}_i(\mathbf{c}) = \mathbf{S}_i(\mathbf{c}_1, \ldots, \mathbf{c}_{n+m-1}, \mathbf{t})$, where $\mathbf{t}$ is any transient.

Let $\dot{\mathbf{N}} = \langle \mathbf{T}, \mathcal{X}, \dot{\mathcal{S}}, \dot{\mathcal{E}} \rangle$ be the network obtained from $\mathbf{N}$ by removing state variable $\mathbf{s}_m$ as described below. The set $\dot{\mathcal{S}}$ of state variables is $\mathcal{S} \setminus \{\mathbf{s}_m\}$, with labels $\dot{\mathbf{s}}_1, \ldots, \dot{\mathbf{s}}_{m-1}$ and excitations $\dot{\mathbf{S}}_1, \ldots, \dot{\mathbf{S}}_{m-1}$, respectively. The excitations in $\dot{\mathbf{N}}$ are defined as follows, for all $i \in [m-1]$ and any total state $\mathbf{d} \in \mathbf{T}^{n+m-1}$ of $\dot{\mathbf{N}}$:

$$\dot{\mathbf{S}}_i(\mathbf{d}) = \mathbf{S}_i(\mathbf{d}, \mathbf{S}_m(\mathbf{d}, \mathbf{t})),$$

where $\mathbf{t}$ is any transient.

Example 3.3.1 *Consider the network* $\mathbf{N}$ *whose graph is in Figure 3.3(a).*

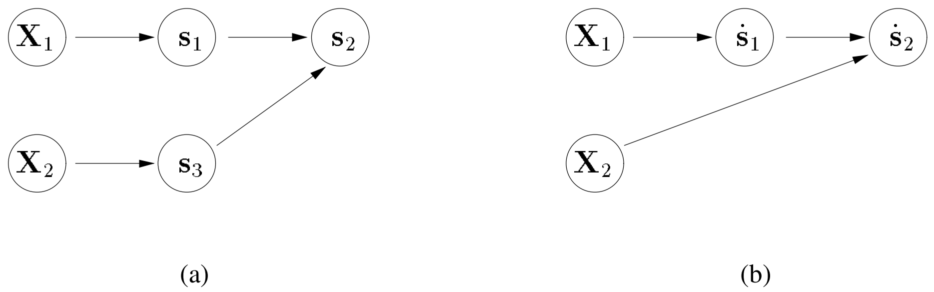


Figure 3.3: Illustration of state variable removal.

The excitations in $\mathbf{N}$ *are*

$$\mathbf{S}_1 = \mathbf{X}_1, \quad \mathbf{S}_2 = \mathbf{s}_1 \oplus \mathbf{s}_3, \quad \mathbf{S}_3 = \mathbf{X}_2.$$

In Figure 3.3(b) we show the network $\dot{\mathbf{N}}$ *obtained from* $\mathbf{N}$ *by removing* $\mathbf{s}_3$*. The new excitations are*

$$\dot{\mathbf{S}}_1 = \mathbf{X}_1, \quad \dot{\mathbf{S}}_2 = \dot{\mathbf{s}_1} \oplus \mathbf{X}_2.$$

Proposition 3.3.1 *Let* $\tilde{a} \cdot b$ *be a binary total state of* $\mathbf{N}$*, and let* $\tilde{a} \cdot \dot{b}$ *be a binary total state of* $\dot{\mathbf{N}}$*, where* $\dot{b}_i = b_i$*, for all* $i \in [m-1]$*. If* $\tilde{a} \cdot b$ *is stable, then* $\tilde{a} \cdot \dot{b}$ *is also stable.*

Proof: For any $i \in [m-1]$,

$\dot{b}_i$

= { hypothesis }

b_i

= { stability of $\tilde{a} \cdot b$ }

$\mathbf{S}_i(\tilde{a} \cdot b)$

= { $b_i = \dot{b}_i$, for all $i \in [m-1]$ }

$\mathbf{S}_i(\tilde{a} \cdot (\dot{b}, b_m))$

= { stability of $\mathbf{s}_m$ in state $\tilde{a} \cdot b$}

$\mathbf{S}_i(\tilde{a} \cdot (\dot{b}, \mathbf{S}_m(\tilde{a} \cdot b)))$

= { $b_i = \dot{b}_i$, for all $i \in [m-1]$ }

$\mathbf{S}_i(\tilde{a} \cdot (\dot{b}, \mathbf{S}_m(\tilde{a} \cdot (\dot{b}, b_m))))$

= { definition of $\dot{\mathbf{S}}_i$ }

$\dot{\mathbf{S}}_i(\tilde{a} \cdot \dot{b})$.

Thus, state $\tilde{a} \cdot \dot{b}$ is stable. □

Proposition 3.3.2 *Let* $\mathbf{s}^H$ *be the result of Algorithm Ã for* $\mathbf{N}$ *started in binary stable state* $\tilde{a} \cdot b$*, with the input changing to* a*. Let* $\dot{\mathbf{s}}^G$ *be the result of Algorithm Ã for* $\dot{\mathbf{N}}$ *started in binary state* $\tilde{a} \cdot \dot{b}$*, with the input changing to* a*, where* $\dot{b}_i = b_i$ *for all* $i \in [m-1]$*. Then, for all* $i \in [m-1]$*,*

$$\mathbf{s}_i^H = \dot{\mathbf{s}}_i^G.$$

Proof: By Proposition 3.3.1, state $\tilde{a} \cdot \dot{b}$ is stable, so that we can run Algorithm Ã. Hence the claim of the proposition is well defined.

Let the sequence resulting from Algorithm Ã for $\mathbf{N}$ be $\mathbf{s}^0, \ldots, \mathbf{s}^H$, and that resulting from Algorithm Ã for $\dot{\mathbf{N}}$ be $\dot{\mathbf{s}}^0, \ldots, \dot{\mathbf{s}}^G$. We consider both sequences infinite, with $\mathbf{s}^h = \mathbf{s}^H$, for all $h > H$, and $\dot{\mathbf{s}}^h = \dot{\mathbf{s}}^G$, for all $h > G$.

It is enough if we show that, for each $h \geq 0$ there exists $j > 0$ such that, for all $i \in [m-1]$,

$$\mathbf{s}_i^h \leq \dot{\mathbf{s}}_i^h \leq \mathbf{s}_i^{h+j}. \tag{3.1}$$

Since both sequences resulting from Algorithm Ã for $\mathbf{N}$ and $\dot{\mathbf{N}}$ converge, from (3.1) the claim of the proposition follows immediately.

We prove (3.1) by induction on $h \geq 0$.

Basis, $h = 0$. By the definition of Algorithm Ã, and the hypothesis of the proposition, we have $\mathbf{s}_i^0 = b_i = \dot{\mathbf{s}}_i^0$, for all $i \in [m-1]$. By the monotonicity of Algorithm Ã, we know $\mathbf{s}^0 \leq \mathbf{s}^1$. Then $\mathbf{s}_i^0 \leq \dot{\mathbf{s}}_i^0 \leq \mathbf{s}_i^1$, for all $i \in [m-1]$. So the basis holds, with $j = 1$.

Induction hypothesis. Suppose (3.1) holds for some $h \geq 0$.

Induction step. We show that there exists $l > 0$ such that, for all $i \in [m-1]$,

$$\mathbf{s}_i^{h+1} \leq \dot{\mathbf{s}}_i^{h+1} \leq \mathbf{s}_i^{h+1+l}.$$

For any $i \in [m-1]$ the following argument applies:

$$\begin{array}{cl}
& \mathbf{s}_i^{h+1} \\
= & \{ \text{ definition of Algorithm Ã } \} \\
& \mathbf{S}_i(\mathbf{a} \cdot \mathbf{s}^h) \\
\leq & \{ \text{ induction hypothesis, and monotonicity of excitations } \} \\
& \mathbf{S}_i(\mathbf{a} \cdot (\dot{\mathbf{s}}^h, \mathbf{s}_m^h)) \\
= & \{ \text{ definition of Algorithm Ã } \} \\
& \mathbf{S}_i(\mathbf{a} \cdot (\dot{\mathbf{s}}^h, \mathbf{S}_m(\mathbf{a} \cdot \mathbf{s}^{h-1}))) \\
\leq & \{ \text{ induction hypothesis, and monotonicity of excitations } \} \\
& \mathbf{S}_i(\mathbf{a} \cdot (\dot{\mathbf{s}}^h, \mathbf{S}_m(\mathbf{a} \cdot (\dot{\mathbf{s}}^{h-1}, \mathbf{s}_m^{h-1})))) \\
\leq & \{ \text{ monotonicity of Algorithm Ã, and monotonicity of excitations } \} \\
& \mathbf{S}_i(\mathbf{a} \cdot (\dot{\mathbf{s}}^h, \mathbf{S}_m(\mathbf{a} \cdot (\dot{\mathbf{s}}^h, \mathbf{s}_m^h)))).
\end{array}$$

Hence

$$\mathbf{s}_i^{h+1} \leq \mathbf{S}_i(\mathbf{a} \cdot (\dot{\mathbf{s}}^h, \mathbf{S}_m(\mathbf{a} \cdot (\dot{\mathbf{s}}^h, \mathbf{s}_m^h)))). \tag{3.2}$$

Next, we note that

$$\mathbf{S}_i(\mathbf{a} \cdot (\dot{\mathbf{s}}^h, \mathbf{S}_m(\mathbf{a} \cdot (\dot{\mathbf{s}}^h, \mathbf{s}_m^h)))) = \dot{\mathbf{s}}_i^{h+1}, \tag{3.3}$$

since

$$\mathbf{S}_i(\mathbf{a} \cdot (\dot{\mathbf{s}}^h, \mathbf{S}_m(\mathbf{a} \cdot (\dot{\mathbf{s}}^h, \mathbf{s}_m^h)))) = \dot{\mathbf{S}}_i(\mathbf{a} \cdot \dot{\mathbf{s}}^h) = \dot{\mathbf{s}}_i^{h+1},$$

by the definitions of $\dot{\mathbf{S}}_i$ and Algorithm Ã.

We have

$$\begin{array}{ll}
 & \mathbf{S}_i(\mathbf{a} \cdot (\dot{\mathbf{s}}^h, \mathbf{S}_m(\mathbf{a} \cdot (\dot{\mathbf{s}}^h, \mathbf{s}_m^h)))) \\
\leq & \{ \text{ induction hypothesis, monotonicity of Algorithm Ã and of excitations } \} \\
 & \mathbf{S}_i(\mathbf{a} \cdot (\mathbf{s}_1^{h+j}, \ldots, \mathbf{s}_{m-1}^{h+j}, \mathbf{S}_m(\mathbf{a} \cdot \mathbf{s}^{h+j}))) \\
= & \{ \text{ definition of Algorithm Ã } \} \\
 & \mathbf{S}_i(\mathbf{a} \cdot (\mathbf{s}_1^{h+j}, \ldots, \mathbf{s}_{m-1}^{h+j}, \mathbf{s}_m^{h+j+1})) \\
\leq & \{ \text{ monotonicity of Algorithm Ã and of excitations } \} \\
 & \mathbf{S}_i(\mathbf{a} \cdot \mathbf{s}^{h+j+1}) \\
= & \{ \text{ definition of Algorithm Ã } \} \\
 & \mathbf{s}_i^{h+j+2}.
\end{array}$$

Thus,

$$\mathbf{S}_i(\mathbf{a} \cdot (\dot{\mathbf{s}}^h, \mathbf{S}_m(\mathbf{a} \cdot (\dot{\mathbf{s}}^h, \mathbf{s}_m^h)))) \leq \mathbf{s}_i^{h+j+2}. \tag{3.4}$$

By (3.2), (3.3), and (3.4), the claim of the induction step follows, with $l = j + 1$. □

Let $\tilde{\mathbf{N}} = \langle \mathbf{T}, \mathcal{X}, \tilde{\mathcal{S}}, \tilde{\mathcal{E}} \rangle$ be the complete network corresponding to a feedback-free network $\mathbf{N} = \langle \mathbf{T}, \mathcal{X}, \mathcal{S}, \mathcal{E} \rangle$. All variables in $\tilde{\mathcal{S}} \setminus \mathcal{S}$ are input-gate, fork-gate, or wire variables with identity excitation functions. We run Algorithm Ã for $\tilde{\mathbf{N}}$ started in (binary) stable total state $\tilde{a} \cdot \tilde{b}$, with the input changing to a. We also run Algorithm Ã for $\mathbf{N}$ started in state $\tilde{a} \cdot b$, with the input changing to a, where $b_i = \tilde{b}_i$, for all $s_i \in \mathcal{S}$. Applying Proposition 3.3.2 repeatedly, we eliminate the variables in $\tilde{\mathcal{S}} \setminus \mathcal{S}$ one by one, and in the end conclude that the result $\mathbf{s}^H$ of Algorithm Ã for $\tilde{\mathbf{N}}$ is the same as the result $\mathbf{s}^G$ of Algorithm Ã for $\mathbf{N}$, with respect to the variables in $\mathcal{S}$.

3.4 Conclusions

We have defined a general simulation algorithm, Algorithm A, using algebra C of transients. We have shown that this algorithm is monotonic with respect to the prefix order; we have also shown that the version of the simulation with initial stable state, Algorithm Ã, is a particular case of the general algorithm, for networks with input-gate variables. Algorithm Ã is also insensitive to the removal of state variables having identity excitations functions. Therefore, the result of Algorithm Ã for a given feedback-free network is the same, with respect to the initial variables, as the result of the same algorithm for the complete version of the given network.

The simulation produces transients corresponding to the state variables. We claim that those transients represent the changes of the state variables during binary analysis. In the remainder of the thesis we prove our claim by comparing the transients resulting from simulation against binary analysis.

Chapter 4

From Binary Analysis to Simulation

4.1 Preliminaries

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

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

Example 4.1.1 *Consider the binary counterpart of the transient network in Figure 3.1 on page 29. The excitations are:*

$$S_1 = \overline{X_2}, \quad S_2 = X_1 \wedge s_1, \quad S_3 = \overline{s_2}, \quad S_4 = s_2 \veebar s_3.$$

In $G_{11}(1011)$, with the same initial total state as that for the simulation in Example 3.1.1 *on page 28, we find, for instance, path*

$$\pi = 1011, 1111, 0111, 0001$$

of length $h = 3$. *If we follow state variable* s_3, *for example, it changes from* 1 *to* 0 *along this path, so the corresponding transient is* 10. *Observe that we can compute this transient by contracting the sequence of values of the variable along the path, i.e.,* $\widehat{1110} = 10$. *The value of* s_3 *in the third step of Algorithm A, as shown in Table 3.1, is* $\mathbf{s}_3^3 = 101$, *which has* 10 *as a prefix. This holds for all components, since* $(10, 010, 10, 1) \leq (10, 010, 101, 1010)$.

Definition 4.1.1 (History) *Let* $\pi = s^0, \ldots, s^h$ *be a path of length* $h \geq 0$ *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* $\widehat{s_i^0 \ldots s_i^h}$, *which shows the changes of the* i*-th state variable along path* π. *We refer to it as the* history *of variable* s_i *along the path. We also define* $\boldsymbol{\Sigma}_i^\pi$ *to be* $\widehat{E_i}$, *where* $E_i = S_i(a \cdot s^0) S_i(a \cdot s^1) \ldots S_i(a \cdot s^h)$, *and we call it the* excitation history *of variable* s_i *along path* π. *The histories of all variables along* π *constitute tuple* $\boldsymbol{\sigma}^\pi = (\boldsymbol{\sigma}_1^\pi, \ldots, \boldsymbol{\sigma}_m^\pi)$. *The histories of all excitations along* π *form tuple* $\boldsymbol{\Sigma}^\pi = (\boldsymbol{\Sigma}_1^\pi, \ldots, \boldsymbol{\Sigma}_m^\pi)$.

Note that $\boldsymbol{\sigma}_i^\pi$ and $\boldsymbol{\Sigma}_i^\pi$ are not always the same. If s_i is unstable initially, they are obviously different, since their first characters are different, that is $s_i^0 \neq S_i(a \cdot s_i^0)$. Even if the variable is stable initially, $\boldsymbol{\sigma}_i^\pi$ and $\boldsymbol{\Sigma}_i^\pi$ can still be different, due to the inertial nature of the delay associated to s_i; the delay might lose changes, so that the variable changes less than its excitation.

Example 4.1.2 *An example of a path in graph* $G_{11}(1011)$ *of the previous example, on which a variable changes less than its excitation is path*

$$\pi = 1011, 0111, 0011.$$

On this path we have $\boldsymbol{\sigma}_3^\pi = 1$, *whereas* $\boldsymbol{\Sigma}_3^\pi = 101$.

The exact relation between $\boldsymbol{\sigma}_i^\pi$ and $\boldsymbol{\Sigma}_i^\pi$ is proved in the next section.

4.2 Covering of Binary Analysis by Simulation

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

Proposition 4.2.1 *Let* $\pi = s^0, \ldots, s^h, s^{h+1}$ *be a path in* $G_a(b)$*, and let* $\pi' = s^0, \ldots, s^h$*. Then, for all* $i \in [m]$*,*

$$\boldsymbol{\sigma}_i^\pi \leq \boldsymbol{\sigma}_i^{\pi'} \circ S_i(a \cdot s^h).$$

Proof: For any variable s_i we have one of the following cases.

Case I, s_i changes during the transition from s^h to s^{h+1}. Then s_i must be unstable in state s^h, *i.e.*, $S_i(a \cdot s^h) \neq s_i^h$, and $s_i^{h+1} = S_i(a \cdot s^h)$. Hence

$$
\begin{array}{ll}
 & \boldsymbol{\sigma}_i^\pi \\
= & \{ \text{ definition } \} \\
 & s_i^0 \ldots \widehat{s_i^h s_i^{h+1}} \\
= & \{ \text{ definition of } \circ \} \\
 & \widehat{s_i^0 \ldots s_i^h} \circ s_i^{h+1} \\
= & \{ \text{ definition of } \boldsymbol{\sigma}_i^{\pi'} \text{ and } s_i^{h+1} = S_i(a \cdot s^h) \text{ from above } \} \\
 & \boldsymbol{\sigma}_i^{\pi'} \circ S_i(a \cdot s^h) \,.
\end{array}
$$

Case II, s_i does not change during the transition from s^h to s^{h+1}. Then $s_i^{h+1} = s_i^h$. Thus

$$
\begin{array}{ll}
 & \boldsymbol{\sigma}_i^\pi \\
= & \{ \text{ definition } \} \\
 & s_i^0 \ldots \widehat{s_i^h s_i^{h+1}} \\
= & \{ \text{ definition of contraction and } s_i^{h+1} = s_i^h \text{ from above } \} \\
 & \widehat{s_i^0 \ldots s_i^h} \\
= & \{ \text{ definition of } \boldsymbol{\sigma}_i^{\pi'} \}
\end{array}
$$

$\boldsymbol{\sigma}_i^{\pi'}$

$\leq$ { definition of $\circ$ }

$\boldsymbol{\sigma}_i^{\pi'} \circ S_i(a \cdot s^h)$.

Thus our claim holds in any case. □

Corollary 4.2.1 *For any path $\pi = s^0, \ldots, s^h, s^{h+1}$ in $G_a(b)$, with $\pi' = s^0, \ldots, s^h$ we have, for all $i \in [m]$,*

$$\boldsymbol{\sigma}_i^{\pi} \leq s_i^0 \circ \boldsymbol{\Sigma}_i^{\pi'}.$$

Proof: The following argument applies for any $i \in [m]$:

$\boldsymbol{\sigma}_i^{\pi}$

$\leq$ { Proposition 4.2.1 }

$\boldsymbol{\sigma}_i^{\pi'} \circ S_i(a \cdot s^h)$

$\leq$ { Proposition 4.2.1 applied recursively }

$(\ldots((s_i^0 \circ S_i(a \cdot s^0)) \circ S_i(a \cdot s^1)) \circ \ldots) \circ S_i(a \cdot s^h)$

= { associativity of $\circ$ }

$s_i^0 \circ (S_i(a \cdot s^0) \circ S_i(a \cdot s^1) \circ \ldots \circ S_i(a \cdot s^h))$

= { definition of $\boldsymbol{\Sigma}_i^{\pi'}$ }

$s_i^0 \circ \boldsymbol{\Sigma}_i^{\pi'}$. □

Corollary 4.2.2 *Let $\pi = s^0, \ldots, s^h$ be a path in $G_a(b)$. For all variables s_i that are stable in s^0,*

$$\boldsymbol{\sigma}_i^{\pi} \leq \boldsymbol{\Sigma}_i^{\pi}. \tag{4.1}$$

Proof: If $h = 0$ then $\boldsymbol{\sigma}_i^{\pi} = s_i^0$ and $\boldsymbol{\Sigma}_i^{\pi} = S_i(a \cdot s^0)$ for all variables s_i. For s_i initially stable we have $s_i^0 = S_i(a \cdot s^0)$ by the definition of stability. The claim follows immediately.

For $h > 0$ the following argument applies for any s_i initially stable:

$$
\begin{array}{ll}
 & \boldsymbol{\sigma}_i^{\pi} \\
\leq & \{ \text{ Corollary 4.2.1 with } \pi' = s^0, \ldots, s^{h-1} \} \\
 & s_i^0 \circ \boldsymbol{\Sigma}_i^{\pi'} \\
= & \{ \text{ stability of } s_i \text{ in } s^0, \text{ and } \alpha(\boldsymbol{\Sigma}_i^{\pi'}) = S_i(a \cdot s^0) \} \\
 & \boldsymbol{\Sigma}_i^{\pi'} \\
\leq & \{ \text{ definition of } \boldsymbol{\Sigma}_i^{\pi} \text{ as } \boldsymbol{\Sigma}_i^{\pi} = \boldsymbol{\Sigma}_i^{\pi'} \circ S_i(a \cdot s^h) \} \\
 & \boldsymbol{\Sigma}_i^{\pi}.
\end{array}
$$

□

In Chapter 5 we study paths for which equality holds in (4.1); we call those *hazard-preserving paths.*

Proposition 4.2.2 *For any path $\pi = s^0, \ldots, s^h$ in $G_a(b)$, and for all $i \in [m]$,*

$$\boldsymbol{\Sigma}_i^{\pi} \leq \mathbf{S}_i(a \cdot \boldsymbol{\sigma}^{\pi}).$$

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

Corollary 4.2.3 *Let path π be as in Proposition 4.2.2 and let s_i, with $i \in [m]$, be a gate variable whose excitation computes Boolean function $f : \{0,1\}^k \longrightarrow \{0,1\}$ that depends* *only on state variables. Let* $\{s_{i_1}, \ldots, s_{i_k}\}$ *be the fan-in set of* s_i*. Then*

$$\mathbf{\Sigma}_i^\pi \leq \mathbf{f}(\boldsymbol{\sigma}_{i_1}^\pi, \ldots, \boldsymbol{\sigma}_{i_k}^\pi). \tag{4.2}$$

Proof: Since the excitation S_i computes function f that depends only on state variables, and the fan-in set of s_i is $\{s_{i_1}, \ldots, s_{i_k}\}$, we have $S_i = f(s_{i_1}, \ldots, s_{i_k})$. From the definition of excitations in $\mathbf{N}$ we have $\mathbf{S}_i(a \cdot \boldsymbol{\sigma}^\pi) = \mathbf{f}(\boldsymbol{\sigma}_{i_1}^\pi, \ldots, \boldsymbol{\sigma}_{i_k}^\pi)$. Applying Proposition 4.2.2, the claim follows immediately. □

In Chapter 6 we study paths for which equality holds in (4.2); we call those *worst-case paths*.

Theorem 4.2.1 *For all paths* $\pi = s^0, \ldots, s^h$ *in* $G_a(b)$*, with* $s^0 = b$*,*

$$\boldsymbol{\sigma}^\pi \leq \mathbf{s}^h,$$

where $\mathbf{s}^h$ *is the* $(h+1)$*st state in the sequence resulting from Algorithm A.*

Proof: We prove the proposition by induction on $h \geq 0$.

Basis, $h = 0$. We have $\pi = s^0 = b = \mathbf{s}^0$; hence $\boldsymbol{\sigma}^\pi = s^0 = \mathbf{s}^0$, so the claim holds.

Induction hypothesis. The claim holds for some $h \geq 0$, *i.e.*, for all paths π of length h from b in $G_a(b)$, we have $\boldsymbol{\sigma}^\pi \leq \mathbf{s}^h$.

Induction step. Let $\gamma = s^0, \ldots, s^h, s^{h+1}$ be a path of length $h+1$ from b in $G_a(b)$. Then $\pi = s^0, \ldots, s^h$ is a path of length h. For all $i \in [m]$,

$$\begin{array}{ll} & \boldsymbol{\sigma}_i^\gamma \\ \leq & \{ \text{ Corollary 4.2.1 } \} \\ & s_i^0 \circ \mathbf{\Sigma}_i^\pi \\ \leq & \{ \ s^0 = b \text{ and Proposition 4.2.2 } \} \\ & b_i \circ \mathbf{S}_i(a \cdot \boldsymbol{\sigma}^\pi) \\ \leq & \{ \text{ induction hypothesis, monotonicity of excitations, and Remark 2.5.1 } \} \\ & b_i \circ \mathbf{S}_i(a \cdot \mathbf{s}^h) \end{array}$$

= { definition of Algorithm A }

$\mathbf{s}_i^{h+1}$. □

Corollary 4.2.4 *If Algorithm A terminates with state* $\mathbf{s}^H$*, then for any path* π *from* b *in* $G_a(b)$ *we have* $\boldsymbol{\sigma}^\pi \leq \mathbf{s}^H$*.*

Proof: We prove the claim by contradiction.

Suppose there exists a path π from b in $G_a(b)$ that satisfies $\boldsymbol{\sigma}_i^\pi > \mathbf{s}_i^H$, for some $i \in [m]$. Let h be the length of π.

If $h \leq H$, Theorem 4.2.1 shows that $\boldsymbol{\sigma}^\pi \leq \mathbf{s}^h$. We also have $\mathbf{s}^h \leq \mathbf{s}^H$, by Proposition 3.1.1. So $\boldsymbol{\sigma}^\pi \leq \mathbf{s}^H$, and in particular $\boldsymbol{\sigma}_i^\pi \leq \mathbf{s}_i^H$, which contradicts our supposition.

If $h > H$, then Theorem 4.2.1 states that $\boldsymbol{\sigma}^\pi \leq \mathbf{s}^h$. By our convention, $\mathbf{s}^h = \mathbf{s}^H$. So, again we have $\boldsymbol{\sigma}_i^\pi \leq \mathbf{s}_i^H$, which is a contradiction. □

4.3 Conclusions

In conclusion, in this chapter we have proved that all the changes that occur in any state variable in binary analysis are shown by the result of the simulation. In particular, if any hazardous change takes place, the simulation detects it. The next question that arises is whether all the changes predicted by the simulation occur in binary analysis. We address this question in the following chapters.

Chapter 5

Characterization of Hazard-Preserving Paths

5.1 Preliminaries

In Chapter 4 we have shown that, for any given circuit, the result of binary analysis is covered by the result of simulation. From now on we concentrate on showing the converse, namely that the result of simulation is covered by the result of binary analysis. That would conclude the equivalence of the two analysis methods. Because the simulation may not terminate for circuits with feedback, we only consider feedback-free circuits from now on.

For the rest of this chapter, we work with the following premises:

- a given *feedback-free circuit*;
- the *complete network* N of the given circuit;
- the *graph* $G_a(b)$ resulting from the binary analysis of N started in *total state* $a \cdot b$, in which *the only variables that are unstable are input-gate variables*.

Note that the simulation computes the longest transients at each step, by virtue of the extended excitations. We want to show that there exist paths in $G_a(b)$ whose histories match those transients. Those are paths on which no changes are lost, that is the history of a variable is the same as its excitation history, for variables that are initially stable; hence the paths we are looking for are *hazard-preserving*, as defined next (a formal argument for the necessity of hazard-preserving paths is presented later in Chapter 7).

Definition 5.1.1 (Hazard-preserving path) *Let π be a path from b in $G_a(b)$, and s_i a state variable that is initially stable in $G_a(b)$. We call π* hazard-preserving with respect to s_i *if $\boldsymbol{\sigma}_i^\pi = \boldsymbol{\Sigma}_i^\pi$. For a set $V \subseteq \mathcal{S}$, path π is* hazard-preserving with respect to V *if it is hazard-preserving with respect to each $s_i \in V$.*

Example 5.1.1 *Consider the complete binary network in Figure 5.1 with its $G_a(b)$ graph, where $a = 01$ and $b = 10101$. A hazard-preserving path with respect to s_3, for instance, is $\pi = 10101, 00101, 00001$, since $\boldsymbol{\sigma}_3^\pi = 10$ and $\boldsymbol{\Sigma}_3^\pi = 10$. In contrast, path $\pi' = 10101, 00101$ is not hazard-preserving with respect to s_3, since $\boldsymbol{\sigma}_3^{\pi'} = 1$ but $\boldsymbol{\Sigma}_3^{\pi'} = 10$.*

Another example of a non-hazard-preserving path was given earlier in Example 4.1.2 on page 40.

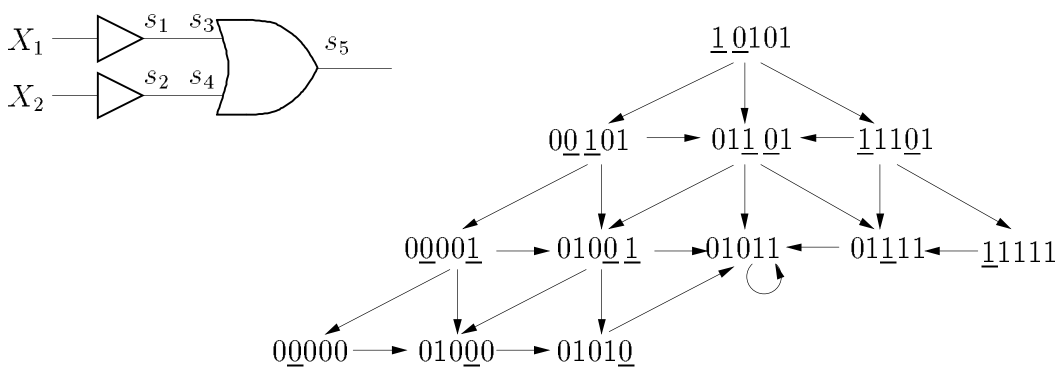


Figure 5.1: Simple circuit with $G_a(b)$ graph.

In this chapter we give a characterization of hazard-preserving paths using a special kind of finite-state machine named *delay automaton* (delay automata are due to J. A. Brzozowski[1]). This characterization is used in Chapter 7 to find hazard-preserving paths in the binary analysis.

Let $G'_a(b)$ be the subgraph of $G_a(b)$ containing all paths on which exactly one variable changes at each step. Note that $G_a(b)$ and $G'_a(b)$ share the initial state b.

Definition 5.1.2 (Equivalent paths) *We call two paths π and π' in $G_a(b)$* equivalent *if they have the same history, i.e., if $\boldsymbol{\sigma}^{\pi} = \boldsymbol{\sigma}^{\pi'}$.*

Proposition 5.1.1 *For any path π in $G_a(b)$ with simultaneous changes, i.e., π not belonging to $G'_a(b)$, there exists an equivalent path π' without simultaneous changes, i.e., π' belonging to $G'_a(b)$.*

Proof: Take any path $\pi = s^0, \ldots, s^{p-1}, s^p, \ldots, s^h$, with $0 < p \leq h$. Let $s_1, \ldots, s_k$ be the variables that change in step p. Since the circuit has no feedback, the graph of N is acyclic, so there exists a topological ordering of its state variables. Let $s_k, \ldots, s_1$ be the order of the k variables in the topological order, *i.e.*, if $s_i \in \phi(s_j)$, then $i > j$, for $i, j \in [k]$. It follows that for any $i \in [k]$, $s_i \notin \phi(s_j)$, for all j such that $i \leq j \leq k$. This means that changing s_i in a state in which $s_k, \ldots, s_i$ are all unstable, leads to a state in which $s_k, \ldots, s_{i+1}$ are still unstable, where $i \in [k-1]$. This allows us to replace step p of π with a sequence $r^0, \ldots, r^k$ of k steps, where $r^0 = s^{p-1}$, $r^k = s^p$, and for any $i \in [k]$ only s_i changes in step i. The new path $\pi' = s^0, \ldots, s^{p-1}, r^1, \ldots, r^{k-1}, s^p, \ldots, s^h$ has the same history as π, *i.e.*, $\boldsymbol{\sigma}^{\pi} = \boldsymbol{\sigma}^{\pi'}$. In a similar way we eliminate all steps with simultaneous changes, and in the end obtain an equivalent path with no simultaneous changes. □

From here on we restrict our attention to $G'_a(b)$. The result above guarantees that by doing so we do not lose any information, in terms of path histories.

[1]Personal communication.

Remark 5.1.1 *Since N is feedback-free, no state variable is related to itself by the precedence relation.*

Proposition 5.1.2 *For any state variables s_i, s_j (not necessarily distinct), if s_i does not precede s_j then s_i and S_j do not change simultaneously in $G'_a(b)$.*

Proof: Since s_i does not precede s_j we have $s_i \notin \phi(s_j)$. Therefore, if s_i and S_j changed in the same step it would mean that a variable in $\phi(s_j)$, and thus different from s_i, changed in the same step; but that contradicts the definition of $G'_a(b)$. □

Corollary 5.1.1 *In $G'_a(b)$ no state variable changes simultaneously with its excitation.*

Proof: Immediate from Remark 5.1.1 and Proposition 5.1.2. □

Proposition 5.1.3 *Any two state variables with distinct excitations have disjoint fan-in sets.*

Proof: Since N is complete, all state variables except wire variables that are inputs of fork gates have zero or one fan-out variable; hence each such variable belongs to at most one fan-in set. Therefore, all state variables except fork-gate variables have distinct excitations and disjoint fan-in sets. □

Proposition 5.1.4 *For any two state variables s_i, s_j with distinct excitations, at most one of S_i, S_j changes at each step in $G'_a(b)$.*

Proof: If S_i and S_j changed at the same step in $G'_a(b)$, it would mean that the single state variable that changes in that step belonged to both $\phi(s_i)$ and $\phi(s_j)$. But that contradicts Proposition 5.1.3. □

5.2 Delay Automata

From now on, for any set $V \subseteq \mathcal{S}$ we denote by $\Xi(V)$ the set of excitation labels of the state variables in V, that is

$$\Xi(V) = \{S_i \mid s_i \in V\}.$$

Consider a set V of state variables in $\mathcal{S}$ that are initially stable in $G'_a(b)$. For simplicity of notation, suppose $V = \{s_1, \ldots, s_k\}$. Suppose these variables are unrelated to each other by the precedence relation and their excitations $S_1, \ldots, S_k$ are pairwise distinct. Recall from Section 2.1 that every state variable represents a delay. The delays associated with the variables in V are depicted in Figure 5.2. We describe the hazard-preserving behavior

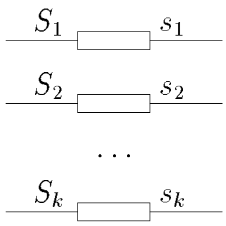


Figure 5.2: Delays for variables in V.

of these delays in $G'_a(b)$, that is we are interested in those paths on which the k delays do not any changes.

Proposition 5.2.1 *In $G'_a(b)$ at most one of $s_1, \ldots, s_k, S_1, \ldots, S_k$ changes at each step.*

Proof: By the definition of $G'_a(b)$, no two variables s_i, s_j, with $i, j \in [k]$ can change in the same step. Since $s_1, \ldots, s_k$ are unrelated to themselves or to each other by the precedence relation, by Proposition 5.1.2 and Corollary 5.1.1 no s_i and S_j change simultaneously in $G'_a(b)$, where $i, j \in [k]$. Since $s_1, \ldots, s_k$ have distinct excitations, by Proposition 5.1.4, no two excitations S_i, S_j change simultaneously in $G'_a(b)$. Hence the claim is true. □

Consider any variable $s_i \in V$. In any state of $G'_a(b)$ s_i can be either stable or unstable; in the initial state b s_i is stable. In any state of $G'_a(b)$, regardless of s_i being stable or unstable, if any of the s_j, or S_j, with $j \in [k], j \neq i$, changes, the (in)stability of s_i is not altered, due to Proposition 5.2.1. When s_i is stable, it cannot change. If S_i changes when s_i is stable, s_i becomes unstable. Since on any hazard-preserving path a variable changes as many times as its excitation, it follows that s_i must change each time it is unstable. Thus, the excitation S_i cannot change when s_i is unstable, but first s_i must change and become stable.

Therefore, we can describe the paths in $G'_a(b)$ that are hazard-preserving with respect to s_i by the automaton depicted in Figure 5.3; we call this automaton 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 automaton state shows whether s_i is stable (label is $\emptyset$) or unstable (label is $\{s_i\}$) in that state.

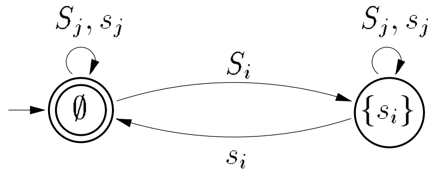


Figure 5.3: Delay automaton for variable s_i.

Definition 5.2.1 (Delay automaton for one variable) *Let* $\Delta_V = V \cup \Xi(V)$. *We denote by* $\mathcal{D}^i_V$ *the delay automaton for variable* s_i. *Automaton* $\mathcal{D}^i_V$ *is formally defined as* $\mathcal{D}^i_V = (Q^i_V, \Delta_V, q^i_0, F^i, \delta^i_V)$, *where:* $Q^i_V = 2^{\{s_i\}} = \{\emptyset, \{s_i\}\}$ *is the set of states,* Δ_V *is the alphabet,* $q^i_0 = \emptyset$ *is the initial state,* $F^i = \{q^i_0\}$ *is the set of final states, and the transition function* $\delta^i_V : Q^i_V \times \Delta_V \longrightarrow Q^i_V$ *is defined as*

$$\delta^i_V(\emptyset, S_i) = \{s_i\},\ \delta^i_V(\{s_i\}, s_i) = \emptyset,$$

and for all $q \in Q^i_V, x \in \Delta_V \setminus \{S_i, s_i\}$,

$$\delta^i_V(q, x) = q.$$

Transitions $\delta^i_V(\emptyset, s_i)$ *and* $\delta^i_V(\{s_i\}, S_i)$ *are undefined.*

Each delay automaton $\mathcal{D}_V^i$, for $i \in [k]$, describes the histories of s_i and its excitation S_i along paths that are hazard-preserving with respect to s_i. We want to describe paths that are hazard-preserving with respect to all variables $s_i \in V$ at the same time, so we need an automaton that, for each $i \in [k]$, behaves like $\mathcal{D}_V^i$. Thus, we need to take the *direct product* [17] of $\mathcal{D}_V^1, \ldots \mathcal{D}_V^k$.

By definition, the direct product of $\mathcal{D}_V^1, \ldots \mathcal{D}_V^k$ is an automaton

$$\mathcal{D}_V = (Q_V, \Delta_V, q_0, F, \delta_V)$$

with $Q_V = Q_V^1 \times \ldots \times Q_V^k,\ q_0 = (q_0^1, \ldots, q_0^k),\ F = F^1 \times \ldots \times F^k$, and transition function $\delta_V : Q_V \times \Delta_V \longrightarrow Q_V$ defined for any $(q_1, \ldots, q_k) \in \mathcal{Q}_V$ and $x \in \Delta_V$ as

$$\delta_V((q_1, \ldots, q_k), x) = (q_1', \ldots, q_k') \text{ iff } \delta_V^i(q_j, x) = q_j',\ \text{ for all } j \in [k].$$

Definition 5.2.2 (Delay automaton) *We call $\mathcal{D}_V$ the* delay automaton of set V.

Example 5.2.1 *With $V = \{s_1, s_2\}$, the delay automata for s_1 and s_2 are shown in Figure 5.4, and their direct product is shown in Figure 5.5.*

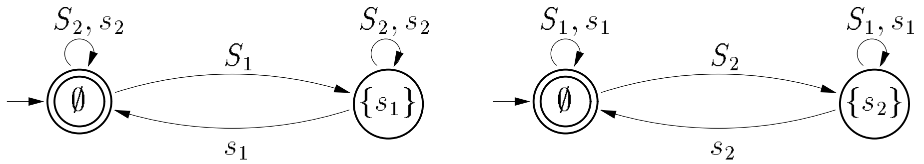


Figure 5.4: Delay automata $\mathcal{D}_V^1$ and $\mathcal{D}_V^2$.

Note that the nonempty components of an automaton state show the variables that are unstable in that state.

Definition 5.2.3 (Balanced word) *A word $w \in \Delta_V^*$ is called* balanced with respect to V *if* $w \in \mathcal{L}(\mathcal{D}_V)$.

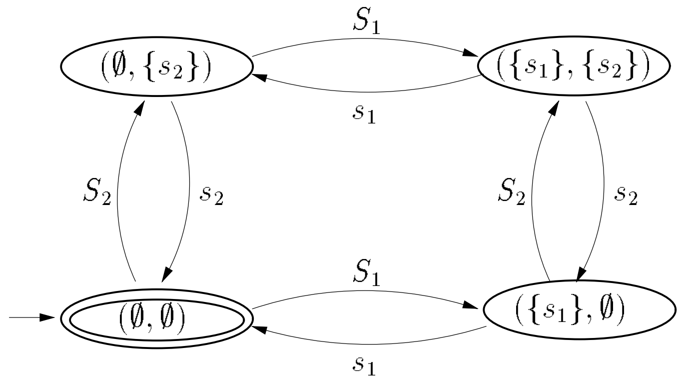


Figure 5.5: Delay automaton of set V.

By the definition of the direct product, we have $\mathcal{L}(\mathcal{D}_V) = \mathcal{L}(\mathcal{D}_V^1) \cap \ldots \cap \mathcal{L}(\mathcal{D}_V^k)$. From the definition of each $\mathcal{D}_V^i$, it is easy to see that a word $w \in \Delta_V^*$ belongs to $\mathcal{L}(\mathcal{D}_V^i)$ iff $w{\downarrow_{\{S_i, s_i\}}}= (S_i s_i)^{c_i}$, for some integer $c_i \geq 0$. Then, the following fact holds.

Remark 5.2.1 *A word $w \in \Delta_V^*$ is balanced with respect to V iff, for all $s_i \in V$,*

$$w{\downarrow_{\{S_i, s_i\}}}= (S_i s_i)^{c_i},$$

for some integer $c_i \geq 0$.

The language $\mathcal{L}(\mathcal{D}_V)$ is a regular subset of the *Dyck language* D_k [22, 34] that is the context-free language of well-balanced parentheses of k types. Our language can be defined as the subset of D_k containing all words u over Δ_V such that, for each prefix v of u and each $i \in [k]$, the number of occurrences of S_i in v equals or is one more than the number of occurrences of s_i in v, and these numbers are equal if v is u (the connection between our language and the Dyck language, and this particular description have been suggested by Z. Ésik).

5.2.1 Delay Automata and Hazard-Preserving Paths

In the following we establish the exact relation between hazard-preserving paths and delay automata. Let V be a set of state variables defined as before. By choosing V, we limit our

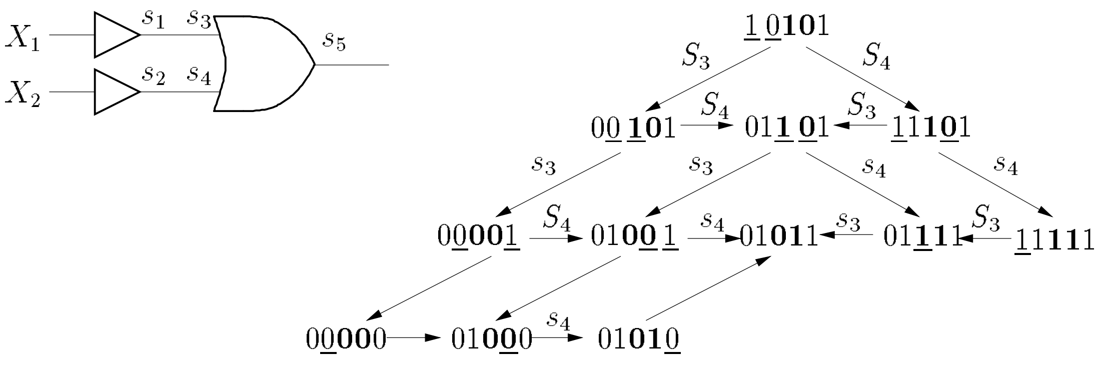


Figure 5.6: Sample circuit with $G'_a(b)$ graph.

interest to paths that are hazard-preserving with respect to V. To any path π in $G'_a(b)$ we associate a word $w^\pi \in \Delta_V^*$ called the *path-word with respect to* V in the following way: we label each step of the path with S_i iff excitation S_i changes, and with s_i iff unstable variable s_i changes in that step, for $S_i \in \Xi(V), s_i \in V$. Other steps are labeled by the empty word. By Proposition 5.2.1, it is guaranteed that each step has a single label. The path-word w^π is the concatenation of the labels along path π.

Example 5.2.2 *Consider the circuit and the $G_a(b)$ graph from Example 5.1.1. We choose $V = \{s_3, s_4\}$ and show the labeled subgraph $G'_a(b)$ in Figure 5.6 beside the circuit. The values of s_3, s_4 are shown in boldface in each state. If we take*

$$\pi = 10101, 00101, 01101, 01111, 01011$$

for instance, then the path-word of π with respect to V is $w^\pi = S_3 S_4 s_4 s_3$.

We denote by $\mathcal{H}_V$ the set of all paths from b in $G'_a(b)$ that are hazard-preserving with respect to V, that is

$$\mathcal{H}_V = \{\pi \mid \pi \text{ is a path from } b \text{ in } G'_a(b), \text{ and for all } s_j \in V,\ \boldsymbol{\sigma}_j^\pi = \boldsymbol{\Sigma}_j^\pi\}.$$

We denote by $\mathcal{W}_V$ the set of path-words of all hazard-preserving paths in $\mathcal{H}_V$, that is

$$\mathcal{W}_V = \{w^\pi \mid \pi \in \mathcal{H}_V\}.$$

Note that the delay automaton is quite general, and it 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 a particular $G'_a(b)$, not all words accepted by the delay automaton correspond to paths in $G'_a(b)$. We find a necessary and sufficient condition for a balanced word to be a path-word; that condition is *relevance*, as defined next.

Definition 5.2.4 (Relevant word) *A balanced word $w \in \mathcal{L}(\mathcal{D}_V)$ is* relevant *to $G'_a(b)$ if 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 5.2.3 *For the circuit and the $G'_a(b)$ of Example 5.2.2, with $V = \{s_3, s_4\}$, the delay automaton $\mathcal{D}_V$ is that of Figure 5.5, with S_3, s_3, S_4, s_4 taking the roles of S_1, s_1, S_2, s_2, respectively. Sample relevant words are in this case $S_4S_3s_4s_3$ and $S_3s_3S_4s_4$, and sample irrelevant words are $S_3s_3S_3S_4s_3s_4$ and $S_3S_4s_4S_4s_3s_4$.*

The following proposition states that a balanced word is the path-word of a hazard-preserving path if and only if it is relevant. This result reduces the problem of finding a hazard-preserving path to finding a relevant balanced word. In Chapter 7 we find relevant balanced words and then apply this result to find corresponding hazard-preserving paths.

Proposition 5.2.2 $\mathcal{W}_V = \mathcal{L}(\mathcal{D}_V){\downarrow_{G'_a(b)}}$.

The proof of this proposition is straightforward but tedious, and is given in the next section.

5.2.2 Proof of Proposition 5.2.2

We first prove $\mathcal{W}_V \subseteq \mathcal{L}(\mathcal{D}_V){\downarrow_{G'_a(b)}}$.

Let π be any path in $\mathcal{H}_V$, and w^π its path-word. Let c_i be the number of times excitation S_i changes along π, for each $i \in [k]$.

For all $i \in [k]$, we have

$$
\begin{aligned}
& |w^\pi|_{S_i} = c_i \\
\Rightarrow \quad & \{\ \boldsymbol{\sigma}_i^\pi = \boldsymbol{\Sigma}_i^\pi \} \\
& |w^\pi|_{s_i} = c_i \\
\Rightarrow \quad & \{ \text{ discussion in beginning of Section 5.2 } \} \\
& w^\pi{\downarrow_{\{S_i,s_i\}}} = (S_i s_i)^{c_i}, \text{ for some integer } c_i \geq 0.
\end{aligned}
$$

Hence, for all $i \in [k]$, $w^\pi{\downarrow_{\{S_i,s_i\}}} = (S_i s_i)^{c_i}$, with $c_i \geq 0$. By Remark 5.2.1, it follows that w^π is in $\mathcal{L}(\mathcal{D}_V)$. Word w^π is relevant, since it is the path-word of π. Hence $w^\pi \in \mathcal{L}(\mathcal{D}_V){\downarrow_{G'_a(b)}}$.

Next we prove the converse, that is $\mathcal{L}(\mathcal{D}_V){\downarrow_{G'_a(b)}} \subseteq \mathcal{W}_V$. In other words, we show that if w is a balanced word relevant to $G'_a(b)$, then there exists a hazard-preserving path γ such that $w^\gamma = w$.

Let *pred*(V) denote the set of all predecessors of the variables in V, and *succ*(V) the set of all successors of the variables in V. Since the variables in V are unrelated to each other by the precedence relation, it follows that *pred*$(V) \cap V = \emptyset$, and *succ*$(V) \cap V = \emptyset$. Also, since N is feedback-free, we have *pred*$(V) \cap$ *succ*$(V) = \emptyset$. Thus, *pred*$(V), V$, and *succ*(V) are disjoint. Therefore, there exists a partition (*left*$(V), V,$ *right*(V)) of $\mathcal{S}$, such that *pred*$(V) \subseteq$ *left*(V), *succ*$(V) \subseteq$ *right*(V), and *right*(V) also includes any state variable that has the same excitation as some variable in V. The other variables that are unrelated to variables in V by the precedence relation are partitioned arbitrarily between *left*(V) and *right*(V). We consider any such partition. The variables of *left*(V) are not influenced by the variables in V and those in *right*(V). We now write a state tuple s as $s = ($*left*$(V), V,$ *right*$(V))$, where the sets are seen as tuples.

Let $\Delta_0 = \Xi(V) = \{S_1, \ldots, S_k\}$, and $\Delta_i = \{S_1, \ldots, S_k, s_1, \ldots, s_i\}$, for all $i \in [k]$.

Let w be a relevant balanced word, that is $w \in \mathcal{L}(\mathcal{D}_V){\downarrow_{G'_a(b)}}$. By definition, there exists path π in $G'_a(b)$ such that $w^\pi{\downarrow_{\Delta_0}} = w{\downarrow_{\Delta_0}}$. Let $\pi = s^0, \ldots, s^h$. Suppose π is not hazard-preserving (if it is, then we are done). Without loss of generality, we can assume

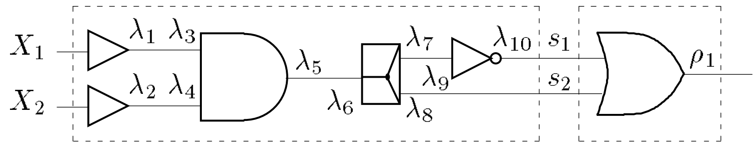


Figure 5.7: Sample circuit for proof illustration.

that $(V^0, \textit{right}^0(V)) = \ldots = (V^h, \textit{right}^h(V))$, that is variables in V and those in *right*(V) retain their initial values along π. To justify this assumption, observe that the histories of the excitations in $\Xi(V)$ that determine the word $w^\pi\!\downarrow_{\Delta_0}$ depend only on *left*(V), and are not influenced by the variables of V, nor by the ones in *right*(V). Thus, given a path with certain histories for the variables of *left*(V), we can always find an equivalent path that satisfies our assumption. With this assumption, since the variables in V do not change, $w^\pi\!\downarrow_{\Delta_0}= w^\pi = w\!\downarrow_{\Delta_0}$.

The proof is by induction, through a series of constructions. The constructions are illustrated with an example using the complete binary network in Figure 5.7.

Example 5.2.4 *The network is started in state $a \cdot b$ with $a = X_1X_2 = 10$ and*

$$b = 0101000001101,$$

where all variables are stable except the input-gate variables. Let $V = \{s_1, s_2\}$. The partition is shown in the figure. Variables in left(V) are labeled with subscripted λ labels, and variables in right(V) are labeled with subscripted ρ labels. We choose word $w = S_1S_2s_1s_2S_2S_1s_1s_2$ in $\mathcal{L}(\mathcal{D}_V)$, and path π in $G'_a(b)$ as shown in Table 5.1. The path-word w^π is in this case $S_1S_2S_2S_1$. Word w is relevant to $G'_a(b)$ since $w\!\downarrow_{\{S_1,S_2\}}= S_1S_2S_2S_1 = w^\pi$. Note that π is not hazard-preserving with respect to V, since variables s_1 and s_2 become unstable, but never change along π.

We prove our claim by induction on i, where $0 \le i \le k$.

Basis, $i = 0$. We show that there exists a path π_0 from b in $G'_a(b)$ such that $w^{\pi_0} = w\!\downarrow_{\Delta_0}$. This is obvious, with $\pi_0 = \pi$.

Table 5.1: Sample path π.

state	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8	λ_9	λ_{10}	s_1	s_2	ρ_1	label
s^0	$\underline{0}$	$\underline{1}$	0	1	0	0	0	0	0	1	1	0	1	
s^1	1	$\underline{1}$	$\underline{0}$	1	0	0	0	0	0	1	1	0	1	
s^2	1	0	$\underline{0}$	$\underline{1}$	0	0	0	0	0	1	1	0	1	
s^3	1	0	1	$\underline{1}$	$\underline{0}$	0	0	0	0	1	1	0	1	
s^4	1	0	1	$\underline{1}$	1	$\underline{0}$	0	0	0	1	1	0	1	
s^5	1	0	1	$\underline{1}$	1	1	$\underline{0}$	$\underline{0}$	0	1	1	0	1	
s^6	1	0	1	$\underline{1}$	1	1	1	$\underline{0}$	$\underline{0}$	1	1	0	1	
s^7	1	0	1	$\underline{1}$	1	1	1	$\underline{0}$	1	$\underline{1}$	1	0	1	
														S_1
s^8	1	0	1	$\underline{1}$	1	1	1	$\underline{0}$	1	0	$\underline{1}$	0	1	
														S_2
s^9	1	0	1	$\underline{1}$	1	1	1	1	1	0	$\underline{1}$	$\underline{0}$	1	
s^{10}	1	0	1	0	$\underline{1}$	1	1	1	1	0	$\underline{1}$	$\underline{0}$	1	
s^{11}	1	0	1	0	0	$\underline{1}$	1	1	1	0	$\underline{1}$	$\underline{0}$	1	
s^{12}	1	0	1	0	0	0	$\underline{1}$	$\underline{1}$	1	0	$\underline{1}$	$\underline{0}$	1	
														S_2
s^{13}	1	0	1	0	0	0	$\underline{1}$	0	1	0	$\underline{1}$	0	1	
s^{14}	1	0	1	0	0	0	0	0	$\underline{1}$	0	$\underline{1}$	0	1	
s^{15}	1	0	1	0	0	0	0	0	0	$\underline{0}$	$\underline{1}$	0	1	
														S_1
s^{16}	1	0	1	0	0	0	0	0	0	1	1	0	1	

Induction hypothesis. Suppose there exists a path π_i from b in $G'_a(b)$ with $w^{\pi_i} = w{\downarrow_{\Delta_i}}$ for some i such that $0 \leq i < k$.

Induction step. We prove there exists a path π_{i+1} from b in $G'_a(b)$, with $w^{\pi_{i+1}} = w{\downarrow_{\Delta_{i+1}}}$.

If w contains no occurrences of s_{i+1}, then $w{\downarrow_{\Delta_{i+1}}} = w{\downarrow_{\Delta_i}}$, so we can take $\pi_{i+1} = \pi_i$. Otherwise, we construct path π_{i+1} from π_i, guided by $w{\downarrow_{\Delta_{i+1}}}$. Let $u = w{\downarrow_{\Delta_{i+1}}}$. Suppose $u = u_0 \ldots u_l$. Word u is the same as word w^{π_i}, except that it also contains occurrences of s_{i+1}. Hence, it is true that $u{\downarrow_{\Delta_i}} = w{\downarrow_{\Delta_i}} = w^{\pi_i}$. We insert in π_i steps labeled with s_{i+1}, in the places indicated by word u. We now describe the procedure.

We illustrate the procedure with our example, by showing how we construct π_1 from π_0, where $\pi_0 = \pi$. In this case $u = u_0 \dots u_5 = S_1 S_2 s_1 S_2 S_1 s_1$.

- Let u_j, with $0 < j \leq l$ be the first occurrence of s_{i+1} in u. Then $u_0 \dots u_{j-1}$ contains only letters from Δ_i. Since $u = w{\downarrow_{\Delta_{i+1}}}$, $u{\downarrow_{\{S_{i+1}, s_{i+1}\}}} = w{\downarrow_{\{S_{i+1}, s_{i+1}\}}}$. Word w is balanced with respect to V, so, from Remark 5.2.1, $w{\downarrow_{\{S_{i+1}, s_{i+1}\}}} = (S_{i+1} s_{i+1})^{c_{i+1}}$, for some integer $c_{i+1} \geq 0$. Hence $u{\downarrow_{\{S_{i+1}, s_{i+1}\}}} = (S_{i+1} s_{i+1})^{c_{i+1}}$. Since u_j is the first occurrence of s_{i+1} in u, it follows that

  $$(u_0 \dots u_{j-1}){\downarrow_{\{S_{i+1}, s_{i+1}\}}} = S_{i+1},$$

  i.e., $u_0 \dots u_{j-1}$ contains only one occurrence of S_{i+1}.

  In our example $j = 2$ and $u_0 u_1 = S_1 S_2$.

  Since $u_0 \dots u_{j-1}$ is a prefix of $u{\downarrow_{\Delta_i}}$ which is the same as w^{π_i}, there must be a prefix of π_i whose path-word is $u_0 \dots u_{j-1}$. We call that a *matching prefix*. Let $\pi^p = s^0, \dots, s^p$, with $0 < p \leq h$, be the shortest such prefix (in fact, any such prefix works). Since s_{i+1} is stable in s^0, the single occurrence of S_{i+1} corresponds to a step in which variable s_{i+1} becomes unstable. With no consequent occurrence of S_{i+1} or s_{i+1}, s_{i+1} is still unstable in state s^p.

  In the example, we take $p = 9$, so $\pi^9 = s^0 \dots s^9$, with path-word $S_1 S_2$, and s_1 is indeed unstable in state s^9.

  We can change s_{i+1} from state s^p, and thus create another state r^p that differs from s^p only in the value of s_{i+1}. Note that s_{i+1} is stable in state r^p.

  In the example, $r^9 = 1011111110001$.

  We insert the new state in π_i right after s^p. If $p = h$ then we are done. Otherwise, we replace $s^{p+1}, \dots, s^h$ with $q^{p+1}, \dots, q^h$, where each q^l, with $p + 1 \leq l \leq h$, differs from s^l only in the value of s_{i+1}. This is possible since the variables in *left*(V) are not influenced by s_{i+1}.

The new path for our example is in the first 18 rows of Table 5.2 (above the horizontal line).

We then repeat the steps above taking $r^p, q^{p+1}, \ldots, q^h$ for the role of π_i and $u_{j+1} \ldots u_l$ in place of u. This is possible, since s_{i+1} is stable in state r^p and word $u_{j+1} \ldots u_l$ is the same as the path-word of $r^p, q^{p+1}, \ldots, q^h$, except for some occurrences of s_{i+1}. We stop when we have exhausted all occurrences of s_{i+1} in u.

At the end we have the desired π_{i+1} with $w^{\pi_{i+1}} = w{\downarrow_{\Delta_{i+1}}}$.

For our example we repeat the procedure only once, with $r^9, q^{10}, \ldots, q^{16}$ and $u_3 \ldots u_5 = S_2S_1s_1$. The first occurrence of s_1 is u_5. We find prefix $r^9, \ldots, q^{16}$ with path-word S_2S_1, and s_1 unstable in q^{16}. We insert state $r^{16} = 1010000001101$ and obtain path π_1 in Table 5.2.

The procedure above allows us to construct a sequence of paths

$$\pi_0, \pi_1, \ldots, \pi_k,$$

where $\pi_0 = \pi$ and for all $i \in [k]$, $w^{\pi_i} = w{\downarrow_{\Delta_i}}$. The path γ that we are looking for is exactly π_k, since $w^{\pi_k} = w{\downarrow_{\Delta_k}} = w$.

For our example the sequence is π_0, π_1, π_2, where π_2, obtained from π_1 by the prescribed procedure, is shown in Table 5.3.

Since w is balanced with respect to V, we know from Remark 5.2.1 that, for all $i \in [k]$, w satisfies $w{\downarrow_{\{S_i,s_i\}}} = (S_is_i)^{c_i}$, with $c_i \geq 0$. By construction, $w^\gamma = w$, so, for all $i \in [k]$, $w^\gamma{\downarrow_{\{S_i,s_i\}}} = (S_is_i)^{c_i}$, with $c_i \geq 0$. This means that S_i changes as many times as s_i along γ, for any $s_i \in V$. Since each $s_i \in V$ is initially stable, it follows that, for all $s_i \in V$, $\boldsymbol{\sigma}_i^\gamma = \boldsymbol{\Sigma}_i^\gamma$. Hence $\gamma \in \mathcal{H}_V$, and $w \in \mathcal{W}_V$. □

The path γ constructed from π by the procedure above has some useful properties that we will exploit in Chapter 7. Path π is $s^0, \ldots, s^h$, with each s^i being

$$(\mathit{left}^{s_i}(V), V^{s_i}, \mathit{right}^{s_i}(V)),$$

for all $i \in [h]$. Let γ be $r^0, \ldots, r^k$, where each r^i, for $i \in [k]$, is $(\mathit{left}^{r_i}(V), V^{r_i}, \mathit{right}^{r_i}(V))$.

Table 5.2: Path π_1.

state	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8	λ_9	λ_{10}	s_1	s_2	ρ_1	label
s^0	$\underline{0}$	$\underline{1}$	0	1	0	0	0	0	0	1	1	0	1	
s^1	1	$\underline{1}$	$\underline{0}$	1	0	0	0	0	0	1	1	0	1	
s^2	1	0	$\underline{0}$	$\underline{1}$	0	0	0	0	0	1	1	0	1	
s^3	1	0	1	$\underline{1}$	$\underline{0}$	0	0	0	0	1	1	0	1	
s^4	1	0	1	$\underline{1}$	1	$\underline{0}$	0	0	0	1	1	0	1	
s^5	1	0	1	$\underline{1}$	1	1	$\underline{0}$	$\underline{0}$	0	1	1	0	1	
s^6	1	0	1	$\underline{1}$	1	1	1	$\underline{0}$	$\underline{0}$	1	1	0	1	
s^7	1	0	1	$\underline{1}$	1	1	1	$\underline{0}$	1	$\underline{1}$	1	0	1	
														S_1
s^8	1	0	1	$\underline{1}$	1	1	1	$\underline{0}$	1	0	$\underline{1}$	0	1	
														S_2
s^9	1	0	1	$\underline{1}$	1	1	1	1	1	0	$\underline{1}$	$\underline{0}$	1	
														s_1
r^9	1	0	1	$\underline{1}$	1	1	1	1	1	0	0	$\underline{0}$	$\underline{1}$	
q^{10}	1	0	1	0	$\underline{1}$	1	1	1	1	0	0	$\underline{0}$	$\underline{1}$	
q^{11}	1	0	1	0	0	$\underline{1}$	1	1	1	0	0	$\underline{0}$	$\underline{1}$	
q^{12}	1	0	1	0	0	0	$\underline{1}$	$\underline{1}$	1	0	0	$\underline{0}$	$\underline{1}$	
														S_2
q^{13}	1	0	1	0	0	0	$\underline{1}$	0	1	0	0	0	$\underline{1}$	
q^{14}	1	0	1	0	0	0	0	0	$\underline{1}$	0	0	0	$\underline{1}$	
q^{15}	1	0	1	0	0	0	0	0	0	$\underline{0}$	0	0	$\underline{1}$	
														S_1
q^{16}	1	0	1	0	0	0	0	0	0	1	$\underline{0}$	0	$\underline{1}$	
														s_1
r^{16}	1	0	1	0	0	0	0	0	0	1	1	0	$\underline{1}$	

The procedure does not affect the sequence of partial states

$$\pi_{\mathit{left}} = \mathit{left}^{s_0}(V), \ldots, \mathit{left}^{s_h}(V)$$

in π; it at most duplicates some of them, so that, if we take the corresponding sequence $\gamma_{\mathit{left}} = \mathit{left}^{r_0}(V), \ldots, \mathit{left}^{r_k}(V)$ in γ and eliminate duplicates, we obtain back π_{left}.

Therefore, we have $\boldsymbol{\sigma}_i^{\pi} = \boldsymbol{\sigma}_i^{\gamma}$, for all $s_i \in \mathit{left}(V)$. Moreover, for each prefix π' of π there exists a prefix γ' of γ such that $\boldsymbol{\sigma}_i^{\pi'} = \boldsymbol{\sigma}_i^{\gamma'}$, for all $s_i \in \mathit{left}(V)$.

Let π' be a prefix of π and w' a prefix of the word w in the procedure, such that $w^{\pi'}\downarrow_{\Xi(V)} = w'\downarrow_{\Xi(V)}$. We argue that it is possible to construct γ so that it has a prefix γ' that satisfies $\boldsymbol{\sigma}_i^{\pi'} = \boldsymbol{\sigma}_i^{\gamma'}$, for all $s_i \in$ *left*(V), and $w^{\gamma'} = w'$. From the discussion above, we know that γ has a prefix γ_1 such that $\boldsymbol{\sigma}_i^{\pi'} = \boldsymbol{\sigma}_i^{\gamma_1}$, for all $s_i \in$ *left*(V). By construction, $w^{\gamma} = w$; since w' is a prefix of w, γ must have a prefix γ_2 such that $w^{\gamma_2} = w'$. We can make $\gamma_1 = \gamma_2$ by using the following rule: in the induction step, when we process an occurrence of s_{i+1}, we choose the *shortest* matching prefix if that occurrence belongs to w', otherwise we choose the *longest* matching prefix. We illustrate this policy with the next example.

Example 5.2.5 *Suppose π is as shown in Figure 5.8(a), and w is $S_1S_2s_2 \cdot s_1S_2s_2$, with the prefix $w' = S_1S_2s_2$ separated by " $\cdot$ " from the rest of the word. The path prefix π' is separated in the figure by a horizontal line from the rest of the path.*

In the first step of the procedure we construct π_1 from π, with $w^{\pi_1} = S_1S_2s_1S_2$. Since s_1 is not from w', we match its preceding S_1S_2 with the longest matching prefix $s^0 \ldots s^6$ of π. We insert the corresponding state r^6 as described in the procedure and obtain the path π_1 shown in Figure 5.8(b). Observe that the longest matching prefix policy ensures that we do not insert in π' steps labeled with characters that are not w'.

In the second step we construct γ from π_1, having $w^{\gamma} = w = S_1S_2s_2 \cdot s_1S_2s_2$. For the first occurrence of s_2 we choose the shortest matching prefix, that is $s^0 \ldots s^4$, because s_2 belongs to w'. For the second occurrence of s_2 we have only one matching prefix. We finally obtain the path γ shown in Figure 5.8(c). Observe that the shortest matching prefix policy ensures that we do insert in π' steps labeled with characters in w'.

Observe that the prefix π' of π has grown into the prefix $\gamma' = s^0 \ldots s^4r^4q^5$ of γ that has $w^{\gamma'} = w'$, and $\boldsymbol{\sigma}_i^{\gamma'} = \boldsymbol{\sigma}_i^{\pi'}$ for all $s_i \in$ left(V).

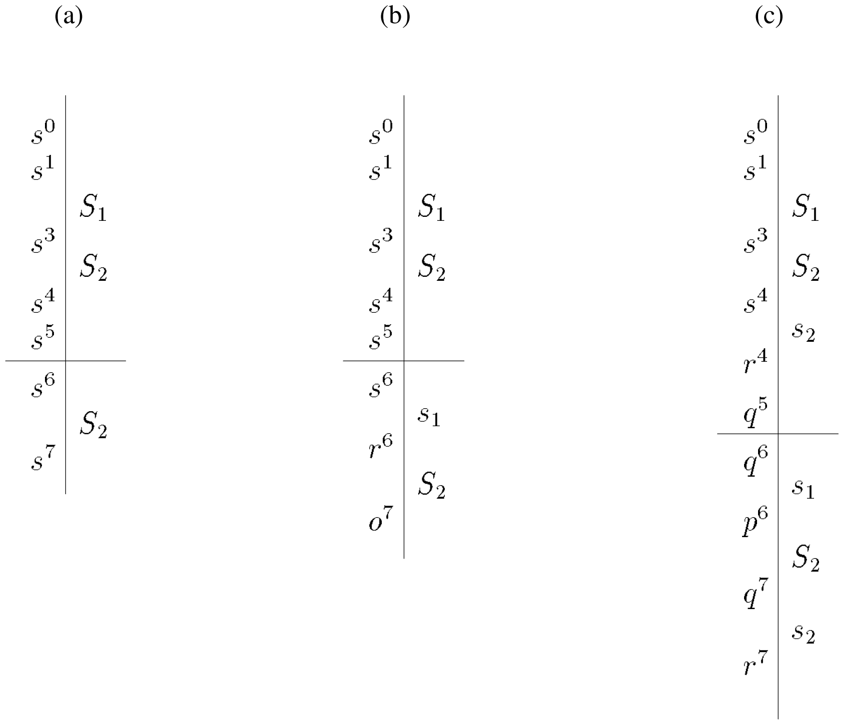


Figure 5.8: Sample path construction.

5.3 Conclusions

In conclusion, in this chapter we have introduced delay automata to model the hazard-preserving behavior of delays in binary analysis. Informally, a delay is hazard-preserving if it does not lose any changes. For any set V of state variables that are initially stable, are unrelated by the precedence relation, and have distinct excitations, we can define a delay automaton $\mathcal{D}_V$. We have proved that the problem of finding a hazard-preserving path with respect to V reduces to finding a relevant balanced word (*i.e.*, a word accepted by $\mathcal{D}_V$ that corresponds to a path in binary analysis).

Table 5.3: Path π_2.

state	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8	λ_9	λ_{10}	s_1	s_2	ρ_1	label
s^0	$\underline{0}$	$\underline{1}$	0	1	0	0	0	0	0	1	1	0	1	
s^1	1	$\underline{1}$	$\underline{0}$	1	0	0	0	0	0	1	1	0	1	
s^2	1	0	$\underline{0}$	$\underline{1}$	0	0	0	0	0	1	1	0	1	
s^3	1	0	1	$\underline{1}$	$\underline{0}$	0	0	0	0	1	1	0	1	
s^4	1	0	1	$\underline{1}$	1	$\underline{0}$	0	0	0	1	1	0	1	
s^5	1	0	1	$\underline{1}$	1	1	$\underline{0}$	$\underline{0}$	0	1	1	0	1	
s^6	1	0	1	$\underline{1}$	1	1	1	$\underline{0}$	$\underline{0}$	1	1	0	1	
s^7	1	0	1	$\underline{1}$	1	1	1	$\underline{0}$	1	$\underline{1}$	1	0	1	
														S_1
s^8	1	0	1	$\underline{1}$	1	1	1	$\underline{0}$	1	0	$\underline{1}$	0	1	
														S_2
s^9	1	0	1	$\underline{1}$	1	1	1	1	1	0	$\underline{1}$	$\underline{0}$	1	
														s_1
r^9	1	0	1	$\underline{1}$	1	1	1	1	1	0	0	$\underline{0}$	$\underline{1}$	
														s_2
r^{10}	1	0	1	$\underline{1}$	1	1	1	1	1	0	0	1	1	
o^{10}	1	0	1	0	$\underline{1}$	1	1	1	1	0	0	1	1	
o^{11}	1	0	1	0	0	$\underline{1}$	1	1	1	0	0	1	1	
o^{12}	1	0	1	0	0	0	$\underline{1}$	$\underline{1}$	1	0	0	1	1	
														S_2
o^{13}	1	0	1	0	0	0	$\underline{1}$	0	1	0	0	$\underline{1}$	1	
o^{14}	1	0	1	0	0	0	0	0	$\underline{1}$	0	0	$\underline{1}$	1	
o^{15}	1	0	1	0	0	0	0	0	0	$\underline{0}$	0	$\underline{1}$	1	
														S_1
o^{16}	1	0	1	0	0	0	0	0	0	1	$\underline{0}$	$\underline{1}$	1	
														s_1
p^{16}	1	0	1	0	0	0	0	0	0	1	1	$\underline{1}$	1	
														s_2
r^{17}	1	0	1	0	0	0	0	0	0	1	1	0	1	

Chapter 6

Gate Automata

This chapter assumes the *complete network* N *and its graphs* $G_a(b)$ *and* $G'_a(b)$*, as defined in the previous chapter.*

6.1 Worst-Case Paths

In order to match the transients produced by the simulation, we need paths in the binary analysis on which gates have worst-case behavior, that is they produce the largest number of changes on their outputs, given the changes on their inputs; hence those paths are *worst-case paths*, as defined next (a formal argument for the necessity of worst-case paths is presented later in Chapter 7).

Definition 6.1.1 (Worst-case path) *Let* π *be a path from* b *in* $G_a(b)$*, and* s_i *a gate variable whose excitation computes Boolean function* $f : \{0,1\}^k \longrightarrow \{0,1\}$ *that depends only on state variables. Let the fan-in set of* s_i *be* $\phi(s_i) = \{s_{i_1}, \ldots, s_{i_k}\}$*. We call* π worst-case with respect to s_i *if* $\mathbf{\Sigma}_i^\pi = \mathbf{f}(\boldsymbol{\sigma}_{i_1}^\pi, \ldots, \boldsymbol{\sigma}_{i_k}^\pi)$*. For a set of gate variables* $V \subseteq \mathcal{S}$*, path* π *is* worst-case with respect to V *if it is worst-case with respect to each* $s_i \in V$*.*

Example 6.1.1 *For the circuit and $G_a(b)$ graph of Example 5.1.1 on page 47, a sample worst-case path with respect to s_5, for instance, is $\pi = 10101, 11101, 11111$, since $\boldsymbol{\Sigma}_5^\pi = 1$ and $\mathbf{f}(\boldsymbol{\sigma}_3^\pi, \boldsymbol{\sigma}_4^\pi) = \boldsymbol{\sigma}_3^\pi \oplus \boldsymbol{\sigma}_4^\pi = 1 \oplus 01 = 1$; in contrast, a sample path that is not worst-case with respect to s_5 is $\pi' = 10101, 01101, 01011$, since $\boldsymbol{\Sigma}_5^{\pi'} = 1$ but $\mathbf{f}(\boldsymbol{\sigma}_3^{\pi'}, \boldsymbol{\sigma}_4^{\pi'}) = \boldsymbol{\sigma}_3^{\pi'} \oplus \boldsymbol{\sigma}_4^{\pi'} = 10 \oplus 01 = 101$.*

In order to characterize worst-case paths in binary analysis, we introduce in this chapter Moore machines [8] that describe the behavior of gates. These machines are called *gate automata* (gate automata are due to J. A. Brzozowski[1]).

6.2 Characterization of Worst-Case Paths

In this section we give a characterization of worst-case paths using gate automata. For the same reasons as before, *we restrict our attention to $G'_a(b)$.*

Let s_i be a gate variable whose excitation depends only on state variables. For simplicity of notation, let $\phi(s_i) = \{s_1, \ldots, s_k\}$, with $i \notin [k]$. Suppose the excitation of s_i computes Boolean function $f : \{0,1\}^k \longrightarrow \{0,1\}$, that is $S_i = f(s_1, \ldots, s_k)$. In any state in $G'_a(b)$, the tuple $(s_1, \ldots, s_k)$ of fan-in variables may have any value in $\{0,1\}^k$. Thus, we have a machine state for each value in $\{0,1\}^k$. Each machine state has as output the value of the gate excitation in that state. In any state of $G'_a(b)$ only one of the fan-in variables may change, in which case we move to a state that differs from the previous one only in the value of the variable that changes. Thus, we have a transition between any two machine states that differ in exactly one component. Each transition is labeled with the name of the variable that changes. The initial state of the machine is given by the values of the fan-in variables in the initial state b of $G'_a(b)$. The resulting automaton is called the *gate automaton* of variable s_i.

[1]Personal communication.

Example 6.2.1 *If s_i corresponds to a two-input OR gate and $\phi(s_i) = \{s_1, s_2\}$, then $S_i = s_1 \vee s_2$. The corresponding machine is depicted in Figure 6.1. For simplicity, we write binary tuples as words. The assumption is that $b_1 b_2 = 00$. In the figure, a bidirectional transition with a label stands for two opposite unidirectional transitions with the same label. The output of a state is shown in the state label after the "/" symbol.*

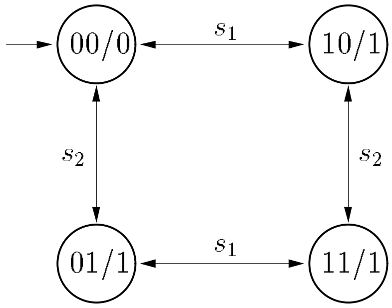


Figure 6.1: OR gate automaton.

The formal definition of a gate automaton is given below. The definition uses the *unit tuples* of $\{0,1\}^k$. For any $j \in [k]$, the jth unit tuple e_j of $\{0,1\}^k$ is the tuple in which only the jth component is 1, the rest being 0. Observe that by taking the component-wise exclusive OR, denoted $\veebar$, of a tuple in $\{0,1\}^k$ with e_j, we obtain the tuple in $\{0,1\}^k$ that differs from the previous one only in the jth component, for any $j \in [k]$.

Definition 6.2.1 (Gate automaton) *The* gate automaton *with respect to $G'_a(b)$ of gate variable s_i whose excitation computes k-ary Boolean function f is a Moore machine*

$$\mathcal{G}_i = (I_i, O, \mathcal{P}_i, p_i^0, \tau_i, o_i),$$

where $I_i = \phi(s_i)$ is the input alphabet, $O = \{0,1\}$ is the output alphabet, $\mathcal{P}_i = \{0,1\}^k$ is the set of states, $p_i^0 = (b_1, \ldots, b_k)$ is the initial state, $\tau_i : \mathcal{P}_i \times I_i \longrightarrow \mathcal{P}_i$ is the transition function defined for all $p \in \mathcal{P}_i$, and all $s_j \in I_i$ as $\tau_i(p, s_j) = p \veebar e_j$, and $o_i : \mathcal{P}_i \longrightarrow O$ is the output function defined for any $p \in \mathcal{P}_i$ as $o_i(p) = f(p)$.

Note that the input language of $\mathcal{G}_i$ is I_i^*. We extend τ_i to words by defining

$$\tau_i : \mathcal{P}_i \times I_i^* \longrightarrow \mathcal{P}_i,$$

with $\tau_i(p, \epsilon) = p,\ \tau_i(p, ws_j) = \tau_i(p, w) \veebar e_j$, for any state $p \in \mathcal{P}_i$ and word $w \in I_i^*$.

In a gate automaton $\mathcal{G}_i$, any input word $u \in I_i^*$ produces a binary output word v as explained below; we write $v = o_i(u)$. If u is the empty word, then $v = f(b)$. A nonempty input word $u = s_{i_1} \dots s_{i_{l-1}} s_{i_l}$ of length $l > 0$ produces output word $v = wf(\tau_i(b, w) \veebar e_{i_l})$, where w is the output word of $s_{i_1} \dots s_{i_{l-1}}$.

Definition 6.2.2 (Output profile) *Let $v = o_i(u)$, for an input word $u \in I_i^*$. Let $\hat{v}$ be the contraction of v. We call $\hat{v}$ the* output profile of u.

Example 6.2.2 *For the gate automaton in Figure 6.1, the output produced by input word $u = s_1 s_2 s_1$, for instance, is $v = 0111$, so its output profile is $\hat{v} = 01$. If $u = s_1 s_1 s_2 s_2$, then $v = 01010$, so $\hat{v} = 01010$.*

6.2.1 Extended Boolean Functions

In this section we establish an equivalence between longest output profiles and extended Boolean functions for transients. We use a gate automaton $\mathcal{G}_i$ defined as before. We also use an alternate representation of transients. Any transient $\mathbf{t} \in \mathbf{T}$ is uniquely determined by the pair $(\alpha(\mathbf{t}), \mathit{length}(\mathbf{t}) - 1)$, where $\mathit{length}(\mathbf{t})$ denotes the number of characters in $\mathbf{t}$. With a slight abuse of notation we write $\mathbf{t} = (\alpha(\mathbf{t}), \mathit{length}(\mathbf{t}) - 1)$.

Every input word $u \in I_i^*$ determines k transients $\mathbf{t}_1, \dots, \mathbf{t}_k$, where $\mathbf{t}_j = (b_j, |u|_{s_j})$, for all $j \in [k]$. Given any k transients $\mathbf{t}_1, \dots, \mathbf{t}_k$, with $\alpha(\mathbf{t}_j) = b_j$, for all $j \in [k]$, there may be more than one input word that determine them. Let $\mathcal{U}(\mathbf{t}_1, \dots, \mathbf{t}_k)$ be the set of input words that determine the k transients $\mathbf{t}_1, \dots, \mathbf{t}_k$. Then

$$\mathcal{U}(\mathbf{t}_1, \dots, \mathbf{t}_k) = \{u \in I_i^* \mid |u|_{s_j} = \mathit{length}(\mathbf{t}_j) - 1,\ \text{for all } j \in [k]\}.$$

Example 6.2.3 *Consider the gate automaton in Figure 6.1. Then* $\mathcal{U}(\epsilon, \epsilon) = \epsilon$*, and*

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

Note that for generating transients the order of the characters does not matter, only their number does. All words in $\mathcal{U}$ have the same length. Note also that any given input word belongs to exactly one set $\mathcal{U}$, since it generates a single tuple of transients.

Let $\mathcal{V}(\mathbf{t}_1, \ldots, \mathbf{t}_k)$ be the set of output profiles for the words in $\mathcal{U}(\mathbf{t}_1, \ldots, \mathbf{t}_k)$, that is

$$\mathcal{V}(\mathbf{t}_1, \ldots, \mathbf{t}_k) = \{\hat{v} \mid v = o_i(u), u \in \mathcal{U}(\mathbf{t}_1, \ldots, \mathbf{t}_k)\}.$$

Let v_{max} be the longest profile in $\mathcal{V}(\mathbf{t}_1, \ldots, \mathbf{t}_k)$.

Definition 6.2.3 (Worst-case word) *An input word* $u \in I_i^*$ *is called a* worst-case word *for* $\mathcal{G}_i$ *if* u *has the longest output profile among the words in* $\mathcal{U}(\mathbf{t}_1, \ldots, \mathbf{t}_k)$*, where* $\mathbf{t}_1, \ldots, \mathbf{t}_k$ *are the transients generated by* u*; formally,* $o_i(u) = v$ *and* $\hat{v} = v_{max}$*.*

Note that the empty word is always worst-case, as are all words over one letter, in particular all words of length 1. In all these cases $\mathcal{U}$ is a singleton; hence $\mathcal{V}$ is a singleton as well. For words over at least two letters, this is no longer true, since, for instance, for the model of Figure 6.1, if we take 10 as the initial state, the profile of s_1s_2 is 101, but that of s_2s_1 is 1.

For the gate automaton of Figure 6.1 (with initial state 00), if we take $u = s_1s_2s_1s_2$, then $v = 01110$ so the output profile of u is 010. If $u = s_1s_1s_2s_2$ then the output profile is 01010. The longest profile for $\mathcal{U}(010, 010)$ is $v_{max} = 01010$, so the latter u is a worst-case word.

Lemma 6.2.1 $v_{max} = \mathbf{f}(\mathbf{t}_1, \ldots, \mathbf{t}_k)$.

Proof: We consider the D-graph $D(\mathbf{t}_1, \ldots, \mathbf{t}_k)$ (defined in Section 2.5). The source node of the D-graph is b and the sink node is $(\mathbf{t}_1, \ldots, \mathbf{t}_k)$. We refer to a path from source to sink in the D-graph as a *complete path*.

Each node $(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ of the D-graph corresponds to state $(\omega(\mathbf{x}_1), \ldots, \omega(\mathbf{x}_k))$ of $\mathcal{G}_i$. The label $f(\omega(\mathbf{x}_1), \ldots, \omega(\mathbf{x}_k))$ of each D-graph node in its associated graph of labels is the output $o_i(\omega(\mathbf{x}_1), \ldots, \omega(\mathbf{x}_k))$ of the state corresponding to that node.

We label each edge $(\mathbf{x}, \mathbf{y})$ in the D-graph with s_i, where $i \in [k]$ is the single component that differs between $\mathbf{x}$ and $\mathbf{y}$. The edge labels along each complete path in the D-graph form a word belonging to I_i^*. Then each complete path in the D-graph corresponds to a path from $b/f(b)$ in $\mathcal{G}_i$. Moreover, the set of edge-label words of all complete paths in the D-graph is exactly $\mathcal{U}(\mathbf{t}_1, \ldots, \mathbf{t}_k)$. Therefore, the set of the contracted node-label sequences of all complete paths in the D-graph is exactly $\mathcal{V}(\mathbf{t}_1, \ldots, \mathbf{t}_k)$. Since $\mathbf{f}(\mathbf{t}_1, \ldots, \mathbf{t}_k)$ is by definition the longest of those sequences, the claim follows immediately. $\square$

Example 6.2.4 *We illustrate the result above with the gate automaton in Figure 6.1 and the D-graph of Figure 2.13 that gives* $01 \oplus 010$. *We show the correspondence of these, as established by the proof above, in Figure 6.2. The node labels of the D-graph are written as exponents of the nodes. Observe that*

$$\mathcal{U}(01, 010) = \{s_1 s_2 s_2,\ s_2 s_1 s_2,\ s_2 s_2 s_1\}.$$

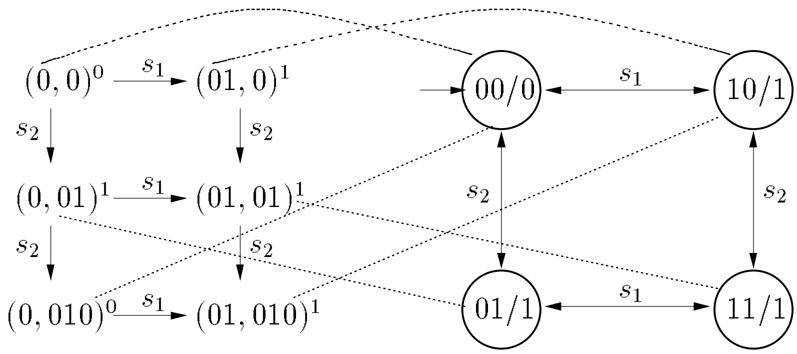


Figure 6.2: Labeled D-graph and gate automaton.

6.2.2 Worst-Case Paths

In the following we find a necessary and sufficient condition for a path in $G'_a(b)$ to be a worst-case path. We use a gate automaton $\mathcal{G}_i$ as before. For any path π from b in $G'_a(b)$ we label with s_j each step in which variable 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 $v^\pi = o_i(u^\pi)$ shows how the excitation S_i changes on π. Word u^π determines transients $\mathbf{t}_1, \ldots, \mathbf{t}_k$.

Remark 6.2.1 *For π as above, we have*

$$1)\ \boldsymbol{\sigma}_j^\pi = \mathbf{t}_j,\ \textit{for all } j \in [k].$$

$$2)\ \boldsymbol{\Sigma}_i^\pi = \widehat{v^\pi}.$$

Proposition 6.2.1 *Let π be a path from b in $G'_a(b)$ labeled as above with u^π. Path π is worst-case with respect to s_i if and only if u^π is a worst-case word.*

Proof:

π is a worst-case path with respect to s_i

$\Leftrightarrow$ { definition of worst-case paths }

$\boldsymbol{\Sigma}_i^\pi = \mathbf{f}(\boldsymbol{\sigma}_1^\pi, \ldots, \boldsymbol{\sigma}_k^\pi)$

$\Leftrightarrow$ { Remark 6.2.1 }

$\widehat{v^\pi} = \mathbf{f}(\mathbf{t}_1, \ldots, \mathbf{t}_k)$

$\Leftrightarrow$ { Lemma 6.2.1 }

$\widehat{v^\pi} = v_{max}$

$\Leftrightarrow$ { definition of worst-case words }

u^π is a worst-case word. □

6.3 Gate Automata and Delay Automata

In this section we prove an important technical lemma that relates gate automata and delay automata, and is used later in Chapter 7.

Proposition 6.3.1 *Let N be a feedback-free complete network and $G'_a(b)$ its binary analysis without simultaneous changes, where $a \cdot b$ is a total state in which the only variables that are unstable are input-gate variables. Let s_i be a gate variable whose excitation depends only on state variables. Then, there exists a delay automaton $\mathcal{D}_{\phi(s_i)}$.*

Proof: In order to be able to define a delay automaton for the variables of $\phi(s_i)$ we have to show that these variables

1) are initially stable in $G'_a(b)$,

2) are unrelated to each other by the precedence relation, and

3) have pairwise-distinct excitations.

Since N is complete, all gate variables other than those of input gates have only wire variables in their fan-in sets. Since we assumed that the excitation of s_i does not depend on inputs, s_i is not an input-gate variable, and all its fan-in variables are wire variables. We assumed that only input-gate variables may be initially unstable in $G'_a(b)$. Therefore the variables in $\phi(s_i)$ are initially stable, which satisfies (1).

If s_i has only one fan-in variable (which happens if s_i corresponds to a buffer, an inverter, or a fork gate) then conditions (2) and (3) are trivially satisfied. Suppose s_i has more than one fan-in variable.

Suppose (2) is not true and there exist two fan-in variables s_g, s_h of s_i such that s_g precedes s_h, *i.e.*, $s_g \mathcal{E}^+ s_h$. Since the gate of s_i is not a fork gate, s_i is the only fan-out variable of s_g. Therefore, we must have $s_i \mathcal{E}^+ s_h$, so s_i precedes s_h. But s_h precedes s_i by virtue of being its fan-in variable. It follows that the network graph of N contains a loop, which contradicts the fact that N is feedback-free.

Since the fan-in variables of gate variable s_i are all wire variables, and only fork-gate variables have identical excitations (see proof of Proposition 5.1.3), the fan-in variables of s_i have pairwise-distinct excitations; hence (2) is true. $\square$

Definition 6.3.1 (Prefix-restricted word) *Let $\mathcal{A}$ be any alphabet. A word $r \in \mathcal{A}^*$ is called* prefix-restricted *if r has a prefix r' satisfying $|r'|_c = 1$ for all letters $c \in \mathcal{A}$ that occur in r (i.e., r has a prefix with exactly one of each of its letters). We call prefix r' the* key prefix *of r.*

Note that the empty word is prefix-restricted, as are all words of length 1 and 2. An example of a word of length 3 that is not prefix-restricted is $r = s_1 s_1 s_2 \in I_i^*$.

Let $\mathcal{G}_i$ be a gate automaton defined as before. By Proposition 6.3.1, we have a delay automaton $\mathcal{D}_{I_i}$. Recall that $\Xi(I_i)$ 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 following lemma 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. In terms of binary analysis this means that as long as we have a prefix-restricted order in which the excitations of the fan-in variables of a gate change, the fan-in variables can always change in a hazard-preserving way, so that the worst-case output is produced by the gate. The result is limited to one- and two-input gates, since a proof for any number of inputs is not known yet, although we believe the result is true.

Lemma 6.3.1 *Let $\mathcal{G}_i$ be the gate automaton of gate variable s_i, corresponding to a one- or two-input gate. For any prefix-restricted word $r \in \Xi(I_i)^*$ with key prefix r', there exists a balanced word $w \in \mathcal{L}(\mathcal{D}_{I_i})$ such that:*

(i) $w{\downarrow_{\Xi(I_i)}} = r$,

(ii) $w{\downarrow_{I_i}}$ *is a worst-case word, and*

(iii) w *has a prefix* u *satisfying* $u{\downarrow_{\Xi(I_i)}}= r'$ *and* $u{\downarrow_{I_i}}$ *has an output profile of length* 2 *if the output profile of* $w{\downarrow_{I_i}}$ *is of length* > 1.[2]

Proof: The case where r is the empty word is trivial, since we take w to be the empty word as well. In the following we assume r is nonempty. Let $w' \in \Delta_{I_i}^*$ be the word obtained from r by replacing every letter S_j of r with $S_j s_j$; for example, if $r = S_1S_1$, then $w' = S_1s_1S_1s_1$; if $r = S_1S_2S_1$ then $w' = S_1s_1S_2s_2S_1s_1$. Word w' satisfies $w' \in \mathcal{L}(\mathcal{D}_{I_i})$ and $w'{\downarrow_{\Xi(I_i)}}= r$.

If s_i is a gate variable for a single-input gate, then $I_i = \{s_1\}$, and $r' = S_1$, so w' is $(S_1s_1)^j$, with $j > 0$. Since every word over one letter is worst-case, $w'{\downarrow_{I_i}}$ is a worst-case word. Word $w'{\downarrow_{I_i}}$ has an output profile of length > 1 iff the transition with s_1 from the initial state changes the output of the machine, *i.e.*, the output profile of s_1 has length 2; in that case, we take the prefix $u = S_1s_1$ of w' that satisfies $u{\downarrow_{\Xi(I_i)}}= S_1 = r'$, and $u{\downarrow_{I_i}}= s_1$ has an output profile of length 2. Thus, w' satisfies (iii).

For two-input gates, we have $I_i = \{s_1, s_2\}$. If only one of the two letters occurs in r, then this case reduces to the previous one. Suppose both letters appear in r and r begins with S_g, where $g \in \{1, 2\}$. Let S_h be the other letter, that is $h \in \{1, 2\}, h \neq g$. Since r is prefix-restricted, its key prefix r' is S_gS_h, so

$$w' = S_gs_g(S_hs_h)^{k_1}(S_gs_g)^{k_2}\ldots, \tag{6.1}$$

with $k_1 > 0$. We call the factors $S_gs_g, (S_hs_h)^{k_1}, (S_gs_g)^{k_2}$, etc. of w' *maximal blocks*. Some discussion is needed, depending on the function f of the gate and the initial state of $\mathcal{G}_i$.

There are 16 Boolean functions of two variables. If $f(s_1, s_2)$ is $0, 1, s_1, \overline{s_1}, s_2$, or $\overline{s_2}$, then f does not satisfy the assumption we made in Section 2.1, since f does not depend on all its arguments. We are left with ten alternatives. When $f(s_1, s_2)$ is $s_1 \veebar s_2$, or $s_1 \veebar \overline{s_2}$, any

[2]This property has only technical value. It guarantees that the prefix-restricted property is preserved by the induction step in a proof in Chapter 7.

input word is a worst-case word because the value of the function changes every time one of the inputs changes. Hence a transition with either s_1 or s_2 entails a change in the output of $\mathcal{G}_i$. In both of these cases w' from (6.1) is the desired word. Note that w' satisfies (iii), if we take u to be $S_g s_g S_h$. This leaves 8 functions: $f(s_1, s_2) = s_1 \vee s_2$, $f(s_1, s_2) = s_1 \vee \overline{s_2}$, $f(s_1, s_2) = \overline{s_1} \vee s_2$, $f(s_1, s_2) = \overline{s_1} \vee \overline{s_2}$ and their duals. Suppose f is one of these. Then $\mathcal{G}_i$ has the property that one of its states has an output different from the other three. In other words, $\mathcal{G}_i$ is of the type shown if Figure 6.3 (the initial state is not considered yet), where A, B, C, D are distinct tuples (represented as words) in $\{0, 1\}^2$ and $x, y \in \{0, 1\}, x \neq y$.

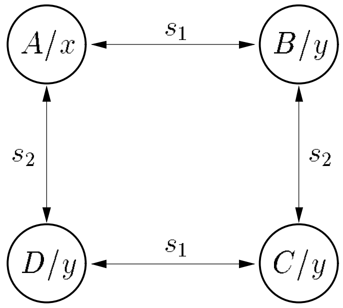


Figure 6.3: Gate automaton type.

We have several cases, depending on the initial state p_i^0 of $\mathcal{G}_i$.

Case I, $p_i^0 = A$. Every input word leads from state A to one of the states A, B, C, D. Among the words that lead from A to A, those of type $(s_1 s_1 + s_2 s_2)^*$ are worst-case, since the output changes at each step, and we cannot do worse than that. Among the words that lead from A to B, those of type $(s_1 s_1 + s_2 s_2)^* s_1$ are worst-case, for the same reason as before. The same happens with words of type $(s_1 s_1 + s_2 s_2)^* s_2$ that lead from A to D. Among those words that lead from A to C, the words of type $(s_1 s_1 + s_2 s_2)^*(s_1 s_2 + s_2 s_1)$ are worst-case; the output changes at every step except the last, but the last step never brings any change, since it is either from B or from D, which have the same output as C. It follows that all the words described by the regular expression

$$R = (s_1 s_1 + s_2 s_2)^*(\epsilon + s_1 + s_2 + s_1 s_2 + s_2 s_1)$$

are worst-case words.

We transform word w' given by (6.1) into a word w such that $w{\downarrow_{I_i}}$ is a worst-case word belonging to R. Note that any word v belonging to R can be uniquely written as *body*(v)*tail*(v), where *body*$(v) \in (s_1s_1 + s_2s_2)^*$, and *tail*$(v) \in (\epsilon + s_1 + s_2 + s_1s_2 + s_2s_1)$. Our aim is to obtain a word w such that $w{\downarrow_{I_i}}$ is of this form. The idea of the transformation is to make each maximal block of w (except for some at the tail) contain an even number of occurrences of either s_g or s_h.

Example 6.3.1 *We illustrate the procedure using the gate automaton of Figure 6.1, in which* $A = 00$. *We take* $w' = S_1s_1S_2s_2S_1s_1S_2s_2$, *with* $w{\downarrow_{I_i}} = s_1s_2s_1s_2$, *that is not worst-case. The worst-case words in this case are* $s_1^2s_2^2$ *and* $s_2^2s_1^2$. *Applying the procedure below, we obtain the former.*

The procedure is the following.

- We process the maximal blocks of w' sequentially, from left to right. For each maximal block $(S_gs_g)^l$ we do the following: if l is even, we do nothing, since we already have an even number of s_g's; if l is odd, we move the last s_g over the next maximal block $t \in \{S_h, s_h\}^*$; thus, $(S_gs_g)^l\, t$ is replaced with $(S_gs_g)^{l-1}S_g\, t\, s_g$. Now $(S_gs_g)^{l-1}{\downarrow_{I_i}}$ is s_g^{l-1}, and $l-1$ is even, so we now have s_g an even number of times. We then process block t, with the roles of g and h reversed. These transformations may create subsequent maximal blocks of the form $s_g(S_gs_g)^l$. For such blocks, if l is odd we do nothing, since we already have an even number of s_g's; if $l = 0$, we do nothing, because that means we are at the end of our word; if l is even and nonzero, then again we move the last s_g over the next maximal block $t \in \{S_h, s_h\}^*$; hence, $s_g(S_gs_g)^l\, t$ becomes $s_g(S_gs_g)^{l-1}S_g\, t\, s_g$. We then process block t, with the roles of g and h reversed, etc.

In our example, the first maximal block is S_1s_1, *so we move* s_1 *over the next maximal block and obtain* $S_1S_2s_2s_1S_1s_1S_2s_2$. *The next maximal block is* S_2s_2; *we move* s_2 *over the next maximal block and obtain* $S_1S_2s_1S_1s_1s_2S_2s_2$. *For the subsequent blocks* $s_1S_1s_1$ *and* $s_2S_2s_2$ *we do nothing.*

The procedure results in a word w whose $w{\downarrow_{I_i}}$ has the form *body*$(w{\downarrow_{I_i}})$*tail*$(w{\downarrow_{I_i}})$, with *body*$(w{\downarrow_{I_i}})$ being $s_g^{l_1} s_h^{l_2} \ldots$, where l_1, l_2, etc. are even, and *tail*$(w{\downarrow_{I_i}}) \in \{\epsilon, s_1, s_2, s_1 s_2, s_2 s_1\}$. But this is just the form of words in R. Thus, $w{\downarrow_{I_i}}$ is a worst-case word, so w satisfies (ii). The order of occurrence of s_j with respect to S_j is not affected by the procedure above, so $w{\downarrow_{\{S_j,s_j\}}} = w'{\downarrow_{\{S_j,s_j\}}}$, for all $j \in \{1,2\}$; hence $w \in \mathcal{L}(\mathcal{D}_{I_i})$. The order of occurrence of the letters in $\Xi(I_i)$ with respect to each other is also unchanged, so $w{\downarrow_{\Xi(I_i)}} = w'{\downarrow_{\Xi(I_i)}} = r$, which satisfies (i). Note also that, by construction and by the prefix assumption, w always has a prefix u, where u is $S_g S_h s_g$, or $S_g S_h s_h$. In any case u satisfies $u{\downarrow_{\Xi(I_i)}} = r'$ and $u{\downarrow_{I_i}} = s_1$ or $u{\downarrow_{I_i}} = s_2$, with output profile of length 2. Thus, w' satisfies (iii), too, so it is the word we were looking for.

Case II, $p_i^0 = B$. With a similar reasoning as for Case I, the words described by the expression $s_1 R$ are worst-case. If $w' = S_1 s_1 (S_2 s_2)^{k_1} (S_1 s_1)^{k_2} \ldots$, we apply the procedure of Case I to the suffix $(S_2 s_2)^{k_1} (S_1 s_1)^{k_2} \ldots$ of w'. If $w' = S_2 s_2 (S_1 s_1)^{k_1} (S_2 s_2)^{k_2} \ldots$, we move the first s_2 over the next maximal block $(S_1 s_1)^{k_1}$ and obtain

$$w'' = S_2 S_1 s_1 (S_1 s_1)^{k_1 - 1} s_2 (S_2 s_2)^{k_2} \ldots;$$

we then apply the procedure of Case I to the suffix $(S_1 s_1)^{k_1-1} s_2 (S_2 s_2)^{k_2} \ldots$ of w''. In any case we obtain a balanced word w satisfying (i) and (ii). Note that w has a prefix u, where $u = S_1 s_1 S_2$, or $u = S_2 S_1 s_1$. Word u satisfies $u{\downarrow_{\Xi(I_i)}} = r'$ and $u{\downarrow_{I_i}} = s_1$, with output profile of length 2. Thus, w is the desired word.

Case III, $p_i^0 = D$. This case is symmetric to the previous one, with the roles of s_1 and s_2 reversed.

Case IV, $p_i^0 = C$. For this case the words described by $(s_1 s_2 + s_2 s_1)R$ are worst-case. Note that on the worst path from C to any other state the output changes at each step except the first, and from C to C the output changes at each step except the first and last. For w' given by (6.1) we apply the procedure of Case I to its suffix $(S_h s_h)^{k_1 - 1} (S_g s_g)^{k_2} \ldots$. The word w obtained is balanced and satisfies (i) and (ii). In addition, w has a prefix u, where

$u = S_g s_g S_h s_h$, so u satisfies $u{\downarrow_{\Xi(I_i)}} = r'$ and $u{\downarrow_{I_i}} = s_g s_h$, with output profile of length 2. Hence w satisfies all requirements. $\square$

It is important to notice that the condition that r be prefix-restricted is essential, as the following example shows. Suppose we change the initial state for the model of Figure 6.1 to be 10. Assume we are given $r = S_2 S_2 S_1$, that is not prefix-restricted. Any balanced word w that satisfies $w{\downarrow_{\Xi I_i}} = r$ must start with $S_2 s_2 S_2$ so $w{\downarrow_{I_i}}$ must start with s_2. To satisfy (ii), $w{\downarrow_{I_i}}$ must be a worst-case word with two s_2's and one s_1. But the only worst-case word of this type is $s_1 s_2 s_2$, that does not start with s_2. Hence it is impossible to satisfy (i) and (ii).

6.4 Conclusions

In this chapter we have introduced gate automata to model the worst-case behavior of gates in binary analysis. Informally, a gate has worst-case behavior if it produces the longest output transient for the given input transients. We have established necessary and sufficient conditions for a path to be a worst-case path. We have also proved an important technical lemma that relates delay automata and gate automata and is instrumental for the constructive proof in Chapter 7. The lemma implies that no matter the order in which the excitations of the fan-in variables of a gate change, as long as this order is prefix-restricted, the fan-in variables can always change in a hazard-preserving way, so that the worst-case output is produced by the gate.

Chapter 7

From Simulation to Binary Analysis

This chapter concludes the process of showing that binary analysis covers simulation, for feedback-free circuits with one- and two-input gates, and initial stable state. We prove that for such circuits the result of Algorithm Ã is covered by the result of binary analysis, in the following sense. Let N and **N** be the two networks modeling a circuit as before. We run Algorithm Ã for **N** started in stable total state $\tilde{a} \cdot b$, with the input tuple $\tilde{a}$ changing to a. We also perform the binary analysis for N with initial total state $a \cdot b$. We show that for any state $\mathbf{s}^h$ resulting from Algorithm Ã, there exists a path π from b in $G_a(b)$ such that $\mathbf{s}^h \leq \boldsymbol{\sigma}^\pi$.

7.1 Network Model

For the result in Chapter 4 the network model is irrelevant; the result holds for any set of state variables. For the implication we prove here that is no longer true. The following example shows that Algorithm Ã can produce states that are not covered by binary analysis, if input-gate and wire variables are not included in the model.

Consider the circuit in Figure 7.1 with associated networks N and $\mathbf{N}$.

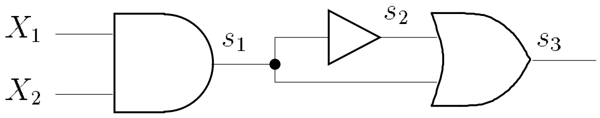


Figure 7.1: Sample circuit for network model relevance.

The excitations in N are:

$$S_1 = X_1 \wedge X_2, \quad S_2 = s_1, \quad S_3 = s_1 \vee s_2,$$

and those in **N** are:

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

We run Algorithm Ã for **N** started in stable total state $\tilde{a} \cdot b$, where $\tilde{a} = 01$ and $b = 000$; we change the input $\tilde{a}$ to $a = 10$. The result is in Table 7.1.

Table 7.1: Result of Algorithm Ã.

$\mathbf{X}_1$	$\mathbf{X}_2$	$\mathbf{s}_1$	$\mathbf{s}_2$	$\mathbf{s}_3$	state
01	10	0	0	0	$\mathbf{s}^0$
01	10	010	0	0	$\mathbf{s}^1$
01	10	010	010	010	$\mathbf{s}^2$
01	10	010	010	01010	$\mathbf{s}^3$

Graph $G_{10}(000)$ resulting from the binary analysis of N has only one state 000, since total state $10 \cdot 000$ is stable. Hence, there is no path in $G_{10}(000)$ whose history covers states $\mathbf{s}^1, \mathbf{s}^2$, or $\mathbf{s}^3$ of the simulation. This shows that there exist networks whose binary analysis does not cover simulation.

One problem in our example is the static hazard 010 on s_1 that is predicted by the simulation but does not occur in binary analysis. As in [8], we add input-gate variables to fix this problem.[1] We add an input-gate variable for each input port. The new network is in Figure 7.2.

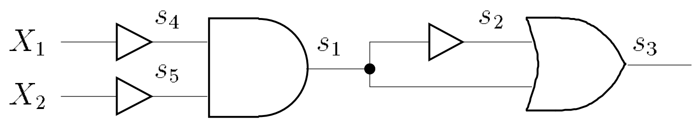


Figure 7.2: Circuit of Figure 7.1 with input-gates.

The excitations in N are now:

$$S_1 = s_4 \wedge s_5, \quad S_2 = s_1, \quad S_3 = s_1 \vee s_2, \quad S_4 = X_1, \quad S_5 = X_2,$$

and those in **N** are:

$$\mathbf{S}_1 = \mathbf{s}_4 \otimes \mathbf{s}_5, \quad \mathbf{S}_2 = \mathbf{s}_1, \quad \mathbf{S}_3 = \mathbf{s}_1 \oplus \mathbf{s}_2, \quad \mathbf{S}_4 = \mathbf{X}_1, \quad \mathbf{S}_5 = \mathbf{X}_2.$$

We take $b = 01000$. The result of the simulation is shown in Table 7.2. We now write state s as $s = (s_4, s_5, s_1, s_2, s_3)$. This time, in $G_{10}(01000)$ we find path

Table 7.2: Simulation for circuit of Figure 7.2.

$\mathbf{X}_1$	$\mathbf{X}_2$	$\mathbf{s}_4$	$\mathbf{s}_5$	$\mathbf{s}_1$	$\mathbf{s}_2$	$\mathbf{s}_3$	state
01	10	0	1	0	0	0	$\mathbf{s}^0$
01	10	01	10	0	0	0	$\mathbf{s}^1$
01	10	01	10	010	0	0	$\mathbf{s}^2$
01	10	01	10	010	010	010	$\mathbf{s}^3$
01	10	01	10	010	010	01010	$\mathbf{s}^4$

$$\pi = \underline{0}\,\underline{1}000, 1\underline{1}\,\underline{0}00, 10\underline{1}\,\underline{0}\,\underline{0}, 100\underline{1}\,\underline{1}, 10000$$

[1]Input-gate variables are equivalent to wire variables. We choose to call them input-gate variables for emphasizing the fact that they are associated to the inputs.

that covers states $\mathbf{s}^0, \mathbf{s}^1, \mathbf{s}^2, \mathbf{s}^3$. However, we show next that there is no path that covers state $\mathbf{s}^4$.

We know from Proposition 3.2.1 that the result of Algorithm Ã is the same as the result of Algorithm A for networks containing input-gate variables, in particular for our network N. Hence, we can apply the propositions and corollaries of Chapter 4 for the result of Algorithm Ã and the graph $G_{10}(01000)$ in our example.

Since $\mathbf{s}_1^4 = 010$ and $\mathbf{s}_2^4 = 010$, from Corollary 4.2.4 we know that, for any path π from 01000 in $G_{10}(01000)$, we have $\boldsymbol{\sigma}_1^\pi \leq 010$ and $\boldsymbol{\sigma}_2^\pi \leq 010$. The graph of Figure 7.3(a) shows all possible orders in which (s_1,s_2) may change along paths starting from 01000 in $G_{10}(01000)$. Any path from $(0,0)$ in this graph shows a possible scenario. Note that we do not claim all such scenarios actually take place in $G_{10}(01000)$. If we replace each pair

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

(a) (b)

Figure 7.3: Possible changes of s_1, s_2, and $s_1 \vee s_2$.

(s_1, s_2) of values in this graph with the corresponding value of $s_1 \vee s_2$, we obtain the graph of Figure 7.3(b) which shows all possible orders in which $s_1 \vee s_2$ may change along paths from the initial state in $G_{10}(01000)$. Since $S_3 = s_1 \vee s_2$, this graph gives us the possible excitation histories of s_3. Observe that the longest history of S_3 in this graph, 01010, can be obtained in only two cases:

$(0,0) \longrightarrow (0,1) \longrightarrow (0,0) \longrightarrow (1,0) \longrightarrow (0,0)$,

or

$(0,0) \longrightarrow (1,0) \longrightarrow (0,0) \longrightarrow (0,1) \longrightarrow (0,0)$.

With s_3 initially stable, by Corollary 4.2.2 we know that $\boldsymbol{\sigma}_3^\pi \le 01010$. Suppose there exists a path π from the initial state in $G_{10}(01000)$ with $\boldsymbol{\sigma}_3^\pi = 01010$. The two cases above show that π must have a prefix π' with (i) $\boldsymbol{\sigma}_1^{\pi'} = 010$ and $\boldsymbol{\sigma}_2^{\pi'} = 0$, or (ii) $\boldsymbol{\sigma}_1^{\pi'} = 0$ and $\boldsymbol{\sigma}_2^{\pi'} = 010$. Case (ii) is impossible, as explained next.

By Corollary 4.2.2, $\boldsymbol{\sigma}_2^{\pi'} \le \boldsymbol{\Sigma}_2^{\pi'}$. Hence case (ii) implies that $\boldsymbol{\Sigma}_2^{\pi'} \ge 010$. But $S_2 = s_1$ means $\boldsymbol{\Sigma}_2^{\pi'} = \boldsymbol{\sigma}_1^{\pi'}$. Therefore $010 \le \boldsymbol{\sigma}_1^{\pi'}$, which contradicts the fact that in case (ii) $\boldsymbol{\sigma}_1^{\pi'} = 0$.

Thus only case (i) is possible, with $\boldsymbol{\sigma}_1^{\pi'} = 010$, and $\boldsymbol{\sigma}_2^{\pi'} = 0$. It follows that the last state of path π', say s^l, is such that $s_1^l = 0$, and $s_2^l = 0$. In such a state neither one of these variables is unstable and cannot become unstable since all changes on s_1 have been exhausted. So s_1 and s_2 do not change further along π. Hence π has $\boldsymbol{\sigma}_3^\pi \le 010$, which contradicts our supposition. This shows there does not exist a path π from 01000 in $G_{10}(01000)$ with $\boldsymbol{\sigma}_3^\pi \ge 01010$. Thus, there is no path π from 01000 in $G_{10}(01000)$ that covers $\mathbf{s}^4$.

The problem is solved if we add a wire variable s_6 on the input wire of the OR gate coming from the buffer. We now have the circuit in Figure 7.4. The excitation of s_6 is $S_6 = s_2$ and the excitation of s_3 is now $S_3 = s_1 \vee s_6$. The other excitations are unchanged. The simulation with $b = 010000$ is in Table 7.3 (the inputs are not shown). State s is now $s = (s_4, s_5, s_1, s_2, s_6, s_3)$.

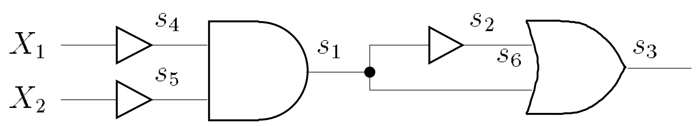


Figure 7.4: Circuit of Figure 7.2 with a wire variable.

In $G_{10}(010000)$ we find path π shown in Table 7.4 with

$$\boldsymbol{\sigma}^\pi = (01, 10, 010, 010, 010, 01010),$$

which satisfies $\mathbf{s}_i^h \le \boldsymbol{\sigma}_i^\pi$, for all $i \in [6], h \in [5] \cup \{0\}$.

Table 7.3: Simulation for circuit of Figure 7.4.

$\mathbf{s}_4$	$\mathbf{s}_5$	$\mathbf{s}_1$	$\mathbf{s}_2$	$\mathbf{s}_6$	$\mathbf{s}_3$	state
0	1	0	0	0	0	$\mathbf{s}^0$
01	10	0	0	0	0	$\mathbf{s}^1$
01	10	010	0	0	0	$\mathbf{s}^2$
01	10	010	010	0	010	$\mathbf{s}^3$
01	10	010	010	010	010	$\mathbf{s}^4$
01	10	010	010	010	01010	$\mathbf{s}^5$

Table 7.4: Path with desired history.

s_4	s_5	s_1	s_2	s_6	s_3
$\underline{0}$	$\underline{1}$	0	0	0	0
1	$\underline{1}$	$\underline{0}$	0	0	0
1	0	$\underline{1}$	$\underline{0}$	0	$\underline{0}$
1	0	0	$\underline{1}$	$\underline{0}$	$\underline{1}$
1	0	0	0	$\underline{1}$	$\underline{0}$
1	0	0	0	0	$\underline{1}$
1	0	0	0	0	0

The lesson learned from the previous example is that forks constrain the inputs of the gates to change only in certain orders, that might not be worst-case. Wire variables can solve the problem since they can perturb the order of the changes on their inputs, and produce worst-case orders at their outputs, as we have proved in Chapters 5 and 6. The example above motivates us to use a network model that contains wire variables. Note that if we associate a state variable to each wire, we automatically have a state variable for each input port, because there is a wire for each such port, so input gates might not be needed. We decide, however, to use input gates as well, since they provide a simple basis of induction for the main proof in the next section. We choose the complete network model that contains input-gate and wire variables. It also contains fork-gate variables. They are needed to guarantee that no two wire variables have the same excitations, and thus to permit the use of delay automata in our proof.

7.2 Covering of Simulation by Binary Analysis

For an arbitrary feedback-free circuit, let N be its binary complete network model. Suppose that in stable total state $\tilde{a} \cdot b$ of network N we change the input tuple $\tilde{a}$ to a. In the resulting total state $a \cdot b$ only input gates are unstable, namely those corresponding to the inputs that change; all other state variables are stable. We perform the binary analysis of N started in state $a \cdot b$ and obtain graph $G_a(b)$. Network N and graph $G_a(b)$ thus defined are like those in Chapters 5 and 6; therefore, all the results of those chapters apply here.

Let **N** be the transient counterpart of N. We run Algorithm Ã for **N** with initial total state $\tilde{a} \cdot b$, and the input changing to a. Since the circuit is feedback-free, the algorithm terminates and we obtain a finite sequence $\mathbf{s}^0, \ldots, \mathbf{s}^H$ of states. Since **N** contains input-gate variables, by Proposition 3.2.1, this sequence is the same as that resulting from Algorithm A for **N** with initial state $a \cdot b$, so all the results of Chapter 4 apply here. If we find a path from b in $G_a(b)$ whose history matches the last state of the simulation, then that path covers all simulation states, due to the monotonicity of Algorithm Ã. Thus, we are looking for a *transient-matching* path, in the sense that we define next.

Definition 7.2.1 (Transient-matching path) *Let π be a path from b in $G_a(b)$. Let $V \subseteq \mathcal{S}$ be a subset of the state variables in N. We call path π* transient-matching *with respect to V if $\boldsymbol{\sigma}_i^\pi = \mathbf{s}_i^H$, for all $s_i \in V$.*

The main claim we prove is stated next. The analysis that follows for the rest of this chapter serves for proving this claim.

Theorem 7.2.1 *There exists a path π from b in $G_a(b)$ that is transient-matching with respect to $\mathcal{S}$, i.e., that satisfies*

$$\boldsymbol{\sigma}^{\pi} = \mathbf{s}^{H}.$$

Proposition 5.1.1 guarantees that in looking for a transient-matching path it is enough to restrict ourselves to $G'_a(b)$. Therefore we use $G'_a(b)$ from now on.

We arrange the state variables of N in levels as described in Section 3.1. This level assignment results in even levels containing gate variables and odd levels containing wire variables. We refer by *level*(s_i) to the level of a state variable s_i. The last level of any network is always a gate level, so N has an even number $2L$ of levels.

Example 7.2.1 *We illustrate the level arrangement with the circuit in Figure 7.5.*

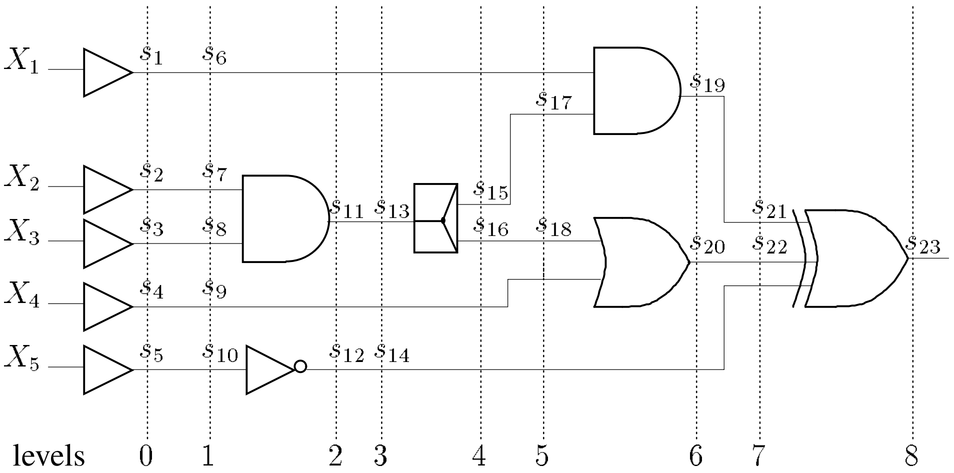


Figure 7.5: Sample feedback-free circuit with levels.

Let $2l$, where $0 < l \leq L$, be a gate level of the circuit. Let V_{2l} be the set of gate variables of level $2l$, that is

$$V_{2l} = \{s_i \in \mathcal{S} \mid \textit{level}(s_i) = 2l\}.$$

Let V be the set of all fan-in variables of the variables of level $2l$, that is

$$V = \bigcup_{s_i \in V_{2l}} \phi(s_i).$$

Suppose $V = \{s_1, \ldots, s_K\}$. Note that $s_1, \ldots, s_K$ are all wire variables; they are not necessarily all of level $2l - 1$, but they belong to levels $< 2l$. Since N is feedback-free and complete, the variables in V are unrelated to each other by the precedence relation and their excitations are pairwise distinct; also, since they are all wire variables, and thus different from input-gate variables, they are initially stable in $G'_a(b)$. Hence we have a delay automaton $\mathcal{D}_V$.

The view that we have of the circuit is pictured in Figure 7.6. The figure depicts the K

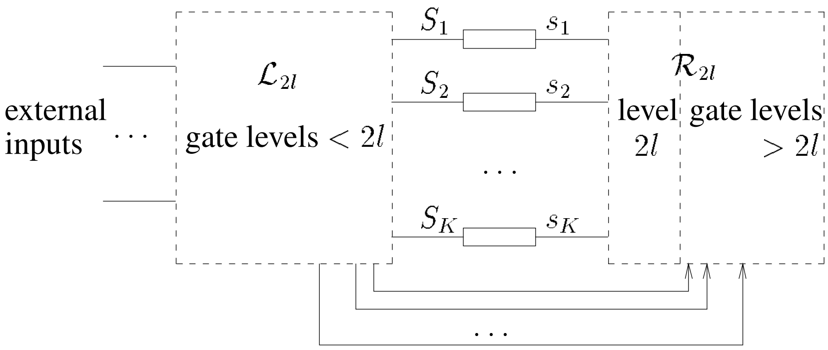


Figure 7.6: View of a feedback-free circuit.

wire variables represented by delays, together with their excitations. 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 form $\mathcal{R}_{2l}$ 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, $\mathcal{L}_{2l}$ and $\mathcal{R}_{2l}$ are sets defined as

$$\mathcal{L}_{2l} = \bigcup_{\substack{\mathit{level}(s_i)<2l,\\ \mathit{level}(s_i)\text{ even}}} (\{s_i\} \cup \phi(s_i)),$$

$$\mathcal{R}_{2l} = \bigcup_{\mathit{level}(s_i)=2l} \{s_i\} \cup \bigcup_{\substack{\mathit{level}(s_i)>2l,\\ \mathit{level}(s_i)\text{ even}}} (\{s_i\} \cup \phi(s_i)).$$

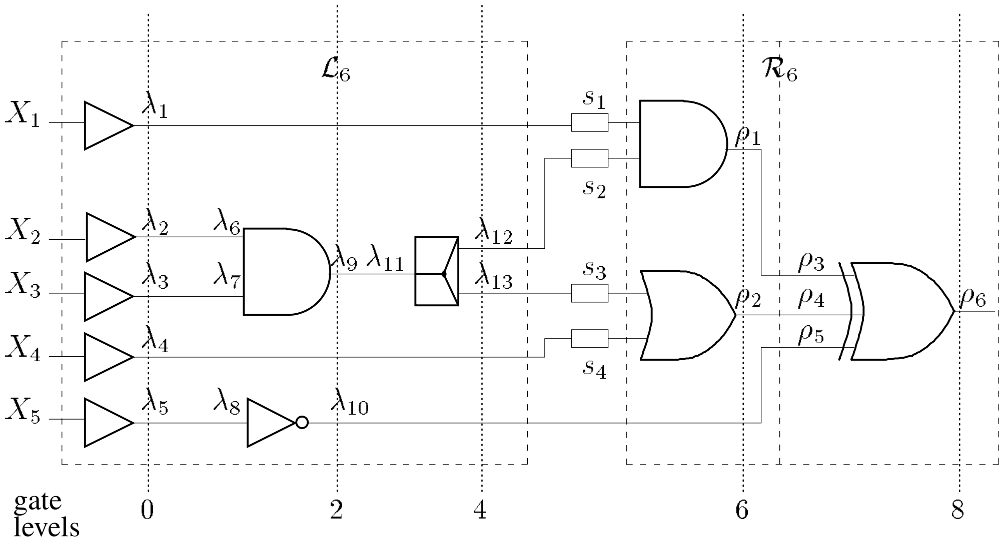


Figure 7.7: Sample circuit with partition.

Observe that $\mathcal{L}_{2l}, \{s_1, \ldots, s_K\}, \mathcal{R}_{2l}$ form a partition of the set $\mathcal{S}$ of state variables in N.

Note that $\mathcal{L}_2 = \bigcup_{level(s_i)=0} \{s_i\}$. Note also that $\mathcal{R}_{2L+2} = \emptyset$ and $\mathcal{L}_{2L+2} = \mathcal{S}$, if we assume a fictitious gate level $2L + 2$.

Example 7.2.2 *We illustrate the partition with the previous sample circuit. The partition, shown in Figure 7.7, is done with respect to the fan-in variables of level-*6 *gates. We relabel the state variables in* $\mathcal{L}_{2l}$ *with subscripted* λ *labels, and state variables in* $\mathcal{R}_{2l}$ *with subscripted* ρ *labels.*

The proof of Theorem 7.2.1 is by induction on l, where $1 \leq l \leq L$. We prove that there exists a path from the initial state in $G'_a(b)$ that is transient-matching with respect to $\mathcal{L}_{2L+2} = \mathcal{S}$. The basis of the induction consists of showing that there exists such a path that is transient-matching with respect to $\mathcal{L}_2$, *i.e.*, with respect to the input-gate variables. In the induction step we assume we have a path τ in $G'_a(b)$ that is transient-matching with respect to $\mathcal{L}_{2l}$. We show that there exists a path π in $G'_a(b)$ that is transient-matching with respect to $\mathcal{L}_{2l+2}$. The proof is given in Section 7.2.1. We first prove a preliminary result. The following lemma gives necessary and sufficient conditions for a path to be transient-matching with respect to $\mathcal{L}_{2l+2}$. Recall that V_{2l} is the set of gate variables of level $2l$, and V is the set of fan-in variables of the variables in V_{2l}.

Lemma 7.2.1 *Path π from b in $G'_a(b)$ is transient-matching with respect to $\mathcal{L}_{2l+2}$ if and only if it is*

1) transient-matching with respect to $\mathcal{L}_{2l}$,

2) hazard-preserving with respect to V,

3) worst-case with respect to V_{2l}, and

4) hazard-preserving with respect to V_{2l}.

Proof: Since $s_1, \ldots, s_K$ are wire variables, each of them has a single fan-in variable. Let $s_{i_1}, \ldots, s_{i_K}$ be the fan-in variables of $s_1, \ldots, s_K$, respectively. Then $S_j = s_{i_j}$, for all $j \in [K]$. By the definition of extended excitations in $\mathbf{N}$, we have, for all $j \in [K]$,

$$\mathbf{S}_j = \mathbf{s}_{i_j}. \tag{7.1}$$

For all state variables $s_j \in \mathcal{S}$, by the termination condition for Algorithm Ã,

$$\mathbf{s}_j^H = \mathbf{S}_j(a \cdot \mathbf{s}^H). \tag{7.2}$$

Then, for all $j \in [K]$,

$$\begin{aligned} & \mathbf{s}_j^H \\ = \quad & \{\ (7.2)\ \} \\ & \mathbf{S}_j(a \cdot \mathbf{s}^H) \\ = \quad & \{\ (7.1)\ \} \\ & \mathbf{s}_{i_j}^H. \end{aligned}$$

Hence, for all $j \in [K]$,

$$\mathbf{s}_j^H = \mathbf{s}_{i_j}^H. \tag{7.3}$$

We first prove the necessity of conditions (1) - (4). Let π be a path in $G'_a(b)$ that is transient-matching with respect to $\mathcal{L}_{2l+2}$. Since $\mathcal{L}_{2l} \subset \mathcal{L}_{2l+2}$, π is transient-matching with respect to $\mathcal{L}_{2l}$; hence it satisfies condition (1) of the lemma.

For all $s_j \in V$ we have

$$
\begin{aligned}
& \boldsymbol{\sigma}_j^\pi \\
= \quad & \{ s_j \in \mathcal{L}_{2l+2}, \text{hypothesis, definition of transient-matching paths} \} \\
& \mathbf{s}_j^H \\
= \quad & \{ (7.3) \} \\
& \mathbf{s}_{i_j}^H \\
= \quad & \{ s_{i_j} \in \mathcal{L}_{2l+2}, \text{hypothesis, definition of transient-matching paths} \} \\
& \boldsymbol{\sigma}_{i_j}^\pi \\
= \quad & \{ S_j = s_{i_j} \} \\
& \boldsymbol{\Sigma}_j^\pi .
\end{aligned}
$$

By the definition of hazard-preserving paths, π is hazard-preserving with respect to V. Therefore (2) is satisfied.

The variables of V_{2l} are all gate variables. Since $l > 0$, they are not input-gate variables; hence they are initially stable in $G'_a(b)$, and their excitations depend only on state variables. For any s_j in V_{2l}, let f be the Boolean function computed by the excitation S_j, and $\phi(s_j) = \{s_{j_1}, \ldots, s_{j_k}\}$. Then $S_j = f(s_{j_1}, \ldots, s_{j_k})$, and

$$\mathbf{S}_j(a \cdot \mathbf{s}^H) = \mathbf{f}(\mathbf{s}_{j_1}^H, \ldots, \mathbf{s}_{j_k}^H). \tag{7.4}$$

Then

$$
\begin{aligned}
& \boldsymbol{\Sigma}_j^\pi \\
\geq \quad & \{ s_j \text{ initially stable, Corollary 4.2.2} \} \\
& \boldsymbol{\sigma}_j^\pi \\
= \quad & \{ s_j \in \mathcal{L}_{2l+2}, \text{hypothesis, definition of transient-matching paths} \}
\end{aligned}
$$

$$
\begin{aligned}
& \mathbf{s}_j^H \\
= \quad & \{ \text{ (7.2) } \} \\
& \mathbf{S}_j(a \cdot \mathbf{s}^H) \\
= \quad & \{ \text{ (7.4)} \} \\
& \mathbf{f}(\mathbf{s}_{j_1}^H, \ldots, \mathbf{s}_{j_k}^H) \\
= \quad & \{ \ s_{j_i} \in \mathcal{L}_{2l+2}, \text{ for all } i \in [k], \text{ hypothesis, definition of transient-matching paths } \} \\
& \mathbf{f}(\boldsymbol{\sigma}_{j_1}^\pi, \ldots, \boldsymbol{\sigma}_{j_k}^\pi) \\
\geq \quad & \{ \text{ Corollary 4.2.3 } \} \\
& \mathbf{\Sigma}_j^\pi .
\end{aligned}
$$

It follows that $\boldsymbol{\sigma}_j^\pi = \mathbf{\Sigma}_j^\pi = \mathbf{f}(\boldsymbol{\sigma}_{j_1}^\pi, \ldots, \boldsymbol{\sigma}_{j_k}^\pi)$, for all $s_j \in V_{2l}$. This shows that π is both hazard-preserving and worst-case with respect to V_{2l}; hence it satisfies (3) and (4).

Next we prove the sufficiency of conditions (1) - (4). Let us take a path π in $G'_a(b)$ that satisfies (1) - (4).

For all $s_j \in V$, we have

$$
\begin{aligned}
& \boldsymbol{\sigma}_j^\pi \\
= \quad & \{ \text{ (2), definition of hazard-preserving paths } \} \\
& \mathbf{\Sigma}_j^\pi \\
= \quad & \{ \ S_j = s_{i_j} \ \} \\
& \boldsymbol{\sigma}_{i_j}^\pi \\
= \quad & \{ \ s_{i_j} \in \mathcal{L}_{2l}, \text{ (1), definition of transient-matching paths } \} \\
& \mathbf{s}_{i_j}^H \\
= \quad & \{ \text{ (7.3) } \} \\
& \mathbf{s}_j^H .
\end{aligned}
$$

Hence, for all $s_j \in V$,

$$\boldsymbol{\sigma}_j^\pi = \mathbf{s}_j^H . \tag{7.5}$$

By the definition of transient-matching paths, π is transient-matching with respect to V.

For all $s_j \in V_{2l}$ we have

$$\begin{array}{ll} & \boldsymbol{\sigma}^{\pi}_{j} \\ = & \{\ \text{(4), definition of hazard-preserving paths}\ \} \\ & \boldsymbol{\Sigma}^{\pi}_{j} \\ = & \{\ \text{(3), definition of worst-case paths}\ \} \\ & \mathbf{f}(\boldsymbol{\sigma}^{\pi}_{j_1}, \ldots, \boldsymbol{\sigma}^{\pi}_{j_k}) \\ = & \{\ s_{j_i} \in V\text{, for all } i \in [k]\text{, (7.5), definition of transient-matching paths}\ \} \\ & \mathbf{f}(\mathbf{s}^{H}_{j_1}, \ldots, \mathbf{s}^{H}_{j_k}) \\ = & \{\ \text{(7.4)}\ \} \\ & \mathbf{S}_j(a \cdot \mathbf{s}^{H}) \\ = & \{\ \text{(7.2)}\ \} \\ & \mathbf{s}^{H}_{j}. \end{array}$$

Hence, for all $s_j \in V_{2l}$,

$$\boldsymbol{\sigma}^{\pi}_{j} = \mathbf{s}^{H}_{j}. \tag{7.6}$$

By the definition of transient-matching paths, π is transient-matching with respect to V_{2l}.

Since $\mathcal{L}_{2l+2} = \mathcal{L}_{2l} \cup V \cup V_{2l}$, (1), (7.5), and (7.6) imply that π is transient-matching with respect to $\mathcal{L}_{2l+2}$. $\square$

7.2.1 Proof of Theorem 7.2.1

This section concludes the proof of Theorem 7.2.1. We assume the given circuit contains only one- and two-input gates. Recall that the circuit is partitioned as shown in Figure 7.6, V_{2l} is the set of gate variables of level $2l$, and V is the set of fan-in variables for the variables of V_{2l}. We have a delay automaton $\mathcal{D}_V$. We take $s_{i_1}, \ldots, s_{i_K}$ to be the fan-in variables of the variables in V.

Example 7.2.3 *Consider the circuit in Figure 7.7, with $l = 3$. In this case $V_6 = \{\rho_1, \rho_2\}$, and $V = \{s_1, s_2, s_3, s_4\}$. The fan-in variables $s_{i_1}, \ldots, s_{i_K}$ are $\lambda_1, \lambda_{12}, \lambda_{13}, \lambda_4$, respectively.*

Let V'_{2l} be the subset of V_{2l} obtained in the following way: from V_{2l} we select for V'_{2l} all variables that are not fork-gate variables, and from all the variables of each fork gate we select for V'_{2l} any one of them. By this construction, the variables in V'_{2l} have pairwise-distinct excitations; since they are in the same level, they are unrelated to each other by the precedence relation. Also, since $l > 0$, the variables in V'_{2l} are not input-gate variables; thus they are initially stable. Hence there exists a delay automaton $\mathcal{D}_{V'_{2l}}$.

Since $l > 0$, the variables of V_{2l} are not input-gate variables, and their excitations depend only on state variables. Then, for each gate variable $s_i \in V'_{2l}$ we have a gate automaton $\mathcal{G}_i$ defined as in Chapter 6, with $I_i = \phi(s_i)$.

Example 7.2.4 *With the same circuit as in the previous example, $V'_6 = \{\rho_1, \rho_2\} = V_6$, since there are no fork-gate variables in V_6. We also have $I_1 = \{s_1, s_2\}$, and $I_2 = \{s_3, s_4\}$ (where by I_i we mean the fan-in set of ρ_i).*

Proposition 7.2.1 *Given the set V'_{2l} defined as above, the sets $\Xi(I_i)$, for all $s_i \in V'_{2l}$, form a partition of $\Xi(V)$.*

Proof: Since N is complete, by Proposition 5.1.3, any two variables with distinct excitations have disjoint fan-in sets. The variables in V'_{2l} have pairwise-distinct excitations, so for any distinct $s_i, s_j \in V'_{2l}$ we have $I_i \cap I_j = \emptyset$. By the construction of V'_{2l}, for any variable s_j in $V_{2l} \setminus V'_{2l}$ there exists a variable s_i in V'_{2l} such that $I_i = I_j$. Thus

$$\bigcup_{s_i \in V'_{2l}} I_i = \bigcup_{s_i \in V_{2l}} I_i = V.$$

Hence the sets I_i, for all s_i in V'_{2l}, form a partition of V. From here we get

$$\bigcup_{s_i \in V'_{2l}} \Xi(I_i) = \Xi(\bigcup_{s_i \in V'_{2l}} I_i) = \Xi(V),$$

and for any distinct $s_i, s_j \in V'_{2l}$,

$$\Xi(I_i) \cap \Xi(I_j) = \Xi(I_i \cap I_j) = \Xi(\emptyset) = \emptyset.$$

Thus, the sets $\Xi(I_i)$, for all $s_i \in V'_{2l}$, form a partition of $\Xi(V)$. $\square$

Example 7.2.5 *For the same circuit as before,* $\Xi(V) = \{S_1, S_2, S_3, S_4\}$ *is partitioned as* $\Xi(I_1) = \{S_1, S_2\},\ \Xi(I_2) = \{S_3, S_4\}$.

Definition 7.2.2 (Edge) *For any* $W \subseteq \mathcal{S}$*, we define the* edge *of* W *as the subset of variables from* W *whose fan-out variables are not in* W*, i.e.,*

$$\mathit{edge}(W) = \{s_i \in W \mid \psi(s_i) \cap W = \emptyset\}.$$

For the circuit of the previous examples, we have, for instance,

$$\mathit{edge}(\mathcal{L}_6) = \{\lambda_1, \lambda_{12}, \lambda_{13}, \lambda_4, \lambda_{10}\}.$$

Note that $s_{i_1}, \ldots, s_{i_K}$ belong to $\mathit{edge}(\mathcal{L}_{2l})$.

Definition 7.2.3 (Path-prefix property) *A path* π *in* $G_a(b)$ *is said to have the* path-prefix property *with respect to a set* W *of state variables if* π *has a prefix* π' *with length*$(\boldsymbol{\sigma}_j^{\pi'}) = 2$*, for all* $s_j \in$ *edge*(W) *that change along* π *(i.e.,* π' *is such that each variable in edge*(W) *that changes along* π *changes exactly once along* π'*).*

The proof of Theorem 7.2.1 is by induction on l, where $1 \leq l \leq L$. The induction step involves a lengthy construction that proceeds in several steps. The steps of the construction are outlined below.

1. From a path τ from b in $G'_a(b)$ that is transient-matching with respect to $\mathcal{L}_{2l}$ and has the path-prefix property with respect to $\mathcal{L}_{2l}$, we construct a path γ from b in $G'_a(b)$ that is transient-matching with respect to $\mathcal{L}_{2l}$, hazard-preserving with respect to V, and worst-case with respect to V_{2l}.

2. From path γ constructed in the previous step, we obtain a path β from b in $G_a(b)$ that is transient-matching with respect to $\mathcal{L}_{2l}$, hazard-preserving with respect to V and V_{2l}, and worst-case with respect to V_{2l}. Path β also has the path-prefix property with respect to $\mathcal{L}_{2l+2}$. Path β does not necessarily belong to $G'_a(b)$.

3. By eliminating all the simultaneous changes from path β obtained in the previous step, we obtain a path π from b in $G'_a(b)$ that is transient-matching with respect to $\mathcal{L}_{2l}$, hazard-preserving with respect to V and V_{2l}, and worst-case with respect to V_{2l}. By Lemma 7.2.1, π is transient-matching with respect to $\mathcal{L}_{2l+2}$. In addition, π has the path-prefix property with respect to $\mathcal{L}_{2l+2}$.

The following lemmas drive the steps of the construction.

Lemma 7.2.2 *Let τ be a path in $G'_a(b)$ that has the path-prefix property with respect to $\mathcal{L}_{2l}$, and let w^τ be its path-word with respect to V. Then $w^\tau{\downarrow}_{\Xi(V)}$ is prefix-restricted.*

Proof: Let $w = w^\tau$, and let $r = w{\downarrow}_{\Xi(V)}$. We have to show that r is prefix-restricted.

Since τ has the path-prefix property with respect to $\mathcal{L}_{2l}$, by the definition of that property, τ has a prefix τ' such that *length*$(\boldsymbol{\sigma}_i^{\tau'}) = 2$, for all $s_i \in$ *edge*$(\mathcal{L}_{2l})$ that change along τ. Since $s_{i_1}, \ldots, s_{i_K}$ belong to *edge*$(\mathcal{L}_{2l})$, τ' has *length*$(\boldsymbol{\sigma}_{i_j}^{\tau'}) = 2$, for all s_{i_j}, with $j \in [K]$, that change along τ. With $S_j = s_{i_j}$, for all $j \in [K]$, this means τ' has *length*$(\boldsymbol{\Sigma}_j^{\tau'}) = 2$, for all S_j, with $j \in [K]$, that change along τ. Let w' be the path-word of τ' with respect to V. Word w' is a prefix of w. We take r' to be $w'{\downarrow}_{\Xi(V)}$. Then r' is a prefix of r with $|r'|_{S_j} = 1$, for all $S_j \in \Xi(V)$ that occur in r. Then, by the definition of prefix-restricted words, r is prefix-restricted, with key prefix r'. □

Lemma 7.2.3 *Given any prefix-restricted word $r \in \Xi(V)^*$ with key prefix r', there exists a balanced word $w \in \mathcal{L}(\mathcal{D}_V)$ that satisfies $w{\downarrow}_{\Xi(V)} = r$, and has a prefix u with $u{\downarrow}_{\Xi(V)} = r'$ such that, for each $s_i \in V'_{2l}$, the following hold:*

1) $w{\downarrow}_{I_i}$ is a worst-case word for the gate automaton $\mathcal{G}_i$ of s_i, and

2) if the output profile of $w{\downarrow_{I_i}}$ has length > 1, the output profile of $u{\downarrow_{I_i}}$ has length 2.

Proof: We write $r = r'r''$. For any $s_i \in V'_{2l}$ we take $r_i = r{\downarrow_{\Xi(I_i)}}$ and $r'_i = r'{\downarrow_{\Xi(I_i)}}$. Word r being prefix-restricted, with key prefix r', r'_i is a prefix of r_i such that $|r'_i|_{S_j} = 1$, for all $S_j \in \Xi(I_i)$ that occur in r_i. In other words, r_i is prefix-restricted, with key prefix r'_i. By Lemma 6.3.1 we obtain a word $w_i \in \Xi(I_i) \cup I_i$ that is balanced with respect to I_i and satisfies:

i) $w_i{\downarrow_{\Xi(I_i)}}= r_i$,

ii) $w_i{\downarrow_{I_i}}$ is a worst-case word, and

iii) w_i has a prefix u_i such that $u_i{\downarrow_{\Xi(I_i)}}= r'_i$, and $u_i{\downarrow_{I_i}}$ has an output profile of length 2 if the output profile of $w_i{\downarrow_{I_i}}$ has length > 1.

Since, by Proposition 7.2.1, the sets $\Xi(I_i)$ are a partition of $\Xi(V)$, r is an interleaving of all r_i. Similarly, r' is an interleaving of all r'_i. We take an interleaving u of all u_i such that $u{\downarrow_{\Xi(V)}}= r'$. There always exists such an interleaving, since each u_i has $u_i{\downarrow_{\Xi(I_i)}}= r'_i$, and r' is an interleaving of all r'_i. We also take an interleaving v of all v_i such that $v{\downarrow_{\Xi(V)}}= r''$. We always find such an interleaving, since each v_i satisfies $v_i{\downarrow_{\Xi(I_i)}}= r''_i$, and r'' is an interleaving of all r''_i.

Each w_i is balanced with respect to I_i, so, for all $s_j \in I_i$, it satisfies $w_i{\downarrow_{\{S_j,s_j\}}}= (S_j s_j)^{c_j}$, with integer $c_j \geq 0$. Word $w = uv$ is an interleaving of all w_i, so, for all $s_j \in I_i$, and, for all I_i, w satisfies $w{\downarrow_{\{S_j s_j\}}}= (S_j s_j)^{c_j}$, with integer $c_j \geq 0$. Word w is then balanced with respect to V, *i.e.*, w is in $\mathcal{L}(\mathcal{D}_V)$. By construction, the prefix u of w has $u{\downarrow_{\Xi(V)}}= r'$. Word w satisfies (1), since $w{\downarrow_{I_i}}= w_i{\downarrow_{I_i}}$ satisfies (ii), and w satisfies (2), since $u{\downarrow_{I_i}}= u_i{\downarrow_{I_i}}$ is like in (iii). Thus, w is the desired word. □

Lemma 7.2.4 *Given a path τ from b in $G'_a(b)$ that is transient-matching with respect to $\mathcal{L}_{2l}$, and has the path-prefix property with respect to $\mathcal{L}_{2l}$, there exists a path γ from b in*

$G'_a(b)$ that is transient-matching with respect to $\mathcal{L}_{2l}$, hazard-preserving with respect to V, and worst-case with respect to V_{2l}.

Proof: Let $r = w^\tau\!\downarrow_{\Xi(V)}$. Since τ has the path-prefix property with respect to $\mathcal{L}_{2l}$, by Lemma 7.2.2 r is prefix restricted. Let r' be the key prefix of r. With Lemma 7.2.3 we obtain a word $w = uv$ that is balanced with respect to V. Word w is also relevant, since

$$w\!\downarrow_{\Xi(V)} = u\!\downarrow_{\Xi(I_i)} v\!\downarrow_{\Xi(I_i)} = r'r'' = r = w^\tau\!\downarrow_{\Xi(V)} .$$

By Proposition 5.2.2, we find a path γ that is hazard-preserving with respect to V, and whose path-word with respect to V is w. Path γ is constructed from τ. Observe that the partition $(\mathcal{L}_{2l}, V, \mathcal{R}_{2l})$ satisfies *pred*$(V) \subseteq \mathcal{L}_{2l}$, *succ*$(V) \subseteq \mathcal{R}_{2l}$, and there are no state variables with the same excitations as variables in V. Hence, we can choose *left*$(V) = \mathcal{L}_{2l}$ and *right*$(V) = \mathcal{R}_{2l}$ in the procedure that constructs path γ. From the discussion at the end of Chapter 5 we know that, by construction, $\boldsymbol{\sigma}_i^\tau = \boldsymbol{\sigma}_i^\gamma$, for all $s_i \in \mathcal{L}_{2l}$. Since τ is transient-matching with respect to $\mathcal{L}_{2l}$, it follows that γ is also transient-matching with respect to $\mathcal{L}_{2l}$.

Each word $w\!\downarrow_{I_i}$ shows how the fan-in variables of variable s_i change along γ. Thus $w\!\downarrow_{I_i} = u^\gamma$, with u^γ defined as in Section 6.2.2. Since every $w\!\downarrow_{I_i}$ is a worst-case word, by Proposition 6.2.1 γ is worst-case with respect to each $s_i \in V'_{2l}$. Hence γ is worst-case with respect to V'_{2l}. Since the variables in $V_{2l} \setminus V'_{2l}$ have the same excitations as variables in V'_{2l}, γ is worst-case with respect to V_{2l}.

Thus, path γ is transient-matching with respect to $\mathcal{L}_{2l}$, hazard-preserving with respect to V, and worst-case with respect to V_{2l}. □

Lemma 7.2.5 *Any path γ constructed as in the proof of Lemma 7.2.4 has the path-prefix property with respect to $\mathcal{L}_{2l}$ via a prefix γ' that also satisfies length$(\boldsymbol{\Sigma}_i^{\gamma'}) = 2$, for all $S_i \in \Xi(V_{2l})$ that change along γ.*

Proof: Word u is a prefix of w that, by construction, satisfies $u\!\downarrow_{\Xi(V)} = r'$. Recall that r' is $w'\!\downarrow_{\Xi(V)}$, where w' is the path-word of τ' with respect to V, τ' being the prefix of τ from the definition of the path-prefix property. So, on one hand we have path τ with prefix τ', and on the other, word w with prefix u such that $u\!\downarrow_{\Xi(V)}= w^{\tau'}\!\downarrow_{\Xi(V)}$. From the discussion at the end of Chapter 5 we know we can construct γ so that it has a prefix γ' whose path-word $w^{\gamma'}$ is u and whose history with respect to $\mathcal{L}_{2l}$ is the same as that of τ'. Since τ' satisfies *length*$(\boldsymbol{\sigma}_i^{\tau'}) = 2$, for all $s_i \in$ *edge*$(\mathcal{L}_{2l})$ that change along τ, γ' satisfies *length*$(\boldsymbol{\sigma}_i^{\gamma'}) = 2$, for all $s_i \in$ *edge*$(\mathcal{L}_{2l})$ that change along γ. Hence γ has the path-prefix property with respect to $\mathcal{L}_{2l}$.

Word u is an interleaving of all u_i. Recall that each $u_i\!\downarrow_{I_i}$, for $s_i \in V'_{2l}$, has an output profile of length 2 when the output profile of $w_i\!\downarrow_{I_i}$ is > 1. The output profile of each $u_i\!\downarrow_{I_i}$ is the history of the excitation S_i along γ', while the output profile of $w_i\!\downarrow_{I_i}$ is the history of the excitation S_i along γ. The output profile of $w_i\!\downarrow_{I_i}$ is of length > 1 iff S_i changes along γ. In other words, we have *length*$(\boldsymbol{\Sigma}_i^{\gamma'}) = 2$, for all $S_i \in \Xi(V'_{2l})$ that change along γ. Since the variables in $V \setminus V'_{2l}$ have the same excitations as variables in V'_{2l}, we have *length*$(\boldsymbol{\Sigma}_i^{\gamma'}) = 2$, for all $S_i \in \Xi(V_{2l})$ that change along γ. $\square$

Lemma 7.2.6 *Given a path γ constructed as in the proof of Lemma 7.2.4, there exists a path β from b in $G_a(b)$ that is transient-matching with respect to $\mathcal{L}_{2l}$, hazard-preserving with respect to V and V_{2l}, and worst-case with respect to V_{2l}.*

Proof: Let t be the path-word of γ with respect to V'_{2l}. Let $v = t\!\downarrow_{\Xi(V'_{2l})}$. We replace each letter S_i in v with $S_i s_i$ and obtain word z. Word z is a balanced word in $\mathcal{L}(\mathcal{D}_{V'_{2l}})$ that is also relevant. By Proposition 5.2.2, we obtain a path β that is hazard-preserving with respect to V'_{2l}, and whose path-word with respect to V'_{2l} is z. Observe that *pred*$(V'_{2l}) \subseteq \mathcal{L}_{2l} \cup V$, *succ*$(V'_{2l}) \subseteq \mathcal{R}_{2l} \setminus V'_{2l}$, and $\mathcal{R}_{2l} \setminus V'_{2l}$ includes $V_{2l} \setminus V'_{2l}$ that contains all variables with the same excitations as variables in V'_{2l}. These facts permit us to take *left*$(V'_{2l}) = \mathcal{L}_{2l} \cup V$ and *right*$(V'_{2l}) = \mathcal{R}_{2l} \setminus V'_{2l}$ for the construction of β. By construction, β then has the same history as γ with respect to $\mathcal{L}_{2l} \cup V$, so β is transient-matching with respect to $\mathcal{L}_{2l}$, hazard-preserving with respect to V, and worst-case with respect to V_{2l}.

We make β hazard-preserving with respect to V_{2l} when we construct it, by using the following convention: whenever we change along β a gate variable in V'_{2l} that belongs to a fork, we simultaneously change all the variables of that fork. This is possible since all the variables of each fork have identical excitations. This convention does not affect the construction procedure, since all variables in $V_{2l} \setminus V'_{2l}$ are in *right*(V'_{2l}), and they do not influence the variables in *left*(V'_{2l}) or those in V'_{2l}. Note that this convention may introduce simultaneity in β, so that β is not guaranteed to belong to $G'_a(b)$. □

Lemma 7.2.7 *Any path β constructed as in the proof of Lemma 7.2.6 has the path-prefix property with respect to $\mathcal{L}_{2l+2}$.*

Proof: Word t has a prefix t' that is the path-word of γ' with respect to V'_{2l}. We take $v' = t'{\downarrow}_{\Xi(V'_{2l})}$. The word z has a prefix z' that is v' with each S_i changed to $S_i s_i$. Note that z' is balanced with respect to V'_{2l}. Path β has a prefix β' whose path-word with respect to V'_{2l} is z'. Since z' is balanced with respect to V'_{2l}, β' is hazard-preserving with respect to V'_{2l}. With the convention mentioned before, β' is hazard-preserving with respect to V_{2l}.

With a similar reasoning as before, it is possible to construct β so that β' has the same history with respect to $\mathcal{L}_{2l} \cup V$ as γ'. Then β' has *length*$(\boldsymbol{\sigma}_i^{\beta'}) = 2$, for all $s_i \in$ *edge*$(\mathcal{L}_{2l})$ that change along β, and *length*$(\boldsymbol{\Sigma}_i^{\beta'}) = 2$, for all $S_i \in \Xi(V_{2l})$ that change along β. Since β' is hazard-preserving, we get *length*$(\boldsymbol{\sigma}_i^{\beta'}) = 2$, for all $s_i \in V_{2l}$ that change along β. Since *edge*$(\mathcal{L}_{2l+2})$ is

$$(\mathit{edge}(\mathcal{L}_{2l}) \setminus \{s_{i_1}, \ldots, s_{i_K}\}) \cup V_{2l},$$

it follows that *length*$(\boldsymbol{\sigma}_i^{\beta'}) = 2$, for all $s_i \in$ *edge*$(\mathcal{L}_{2l+2})$ that change along β. Hence β has the path-prefix property with respect to $\mathcal{L}_{2l+2}$. □

We are now ready to proceed with our induction.

Basis, $l = 1$. We show that there exists a path in $G'_a(b)$ that is transient-matching with respect to $\mathcal{L}_2$.

Suppose $s_{j_1}, \ldots, s_{j_k}$ are the input-gate variables that are unstable in the initial state b that is common for simulation and binary analysis. By the definition of Algorithm Ã, these variables change in the first step and they never change again, so that in the last state $\mathbf{s}^H$ their values are $\mathbf{s}^H_{j_i} = b_{j_i}\overline{b_{j_i}}$, for all $i \in [k]$. Other input-gate variables s_j that are initially stable never change during simulation so that in state $\mathbf{s}^H$ their value is $\mathbf{s}^H_j = b_j$.

Variables $s_{j_1}, \ldots s_{j_k}$ are unrelated to each other by the precedence relation, so changing one of them in $G'_a(b)$ does not affect the instability of the others. Thus, we can construct path $\pi = s^0, \ldots, s^k$ such that $s^0 = b$ and in each step i only s_{j_i} changes, for all $i \in [k]$. It follows that $\boldsymbol{\sigma}^\pi_{j_i} = b_{j_i}\overline{b_{j_i}}$, for all $i \in [k]$, and $\boldsymbol{\sigma}^\pi_j = b_j$, for all input gates s_j that are stable in b. Hence $\boldsymbol{\sigma}^\pi_j = \mathbf{s}^H_j$, for all input-gate variables s_j. Therefore π is transient-matching with respect to $\mathcal{L}_2$. Note that π also has the path-prefix property with respect to $\mathcal{L}_2$, because *edge*$(\mathcal{L}_2) = \mathcal{L}_2$, and $\boldsymbol{\sigma}^\pi_j = b_j\overline{b_j}$, for all $s_j \in \mathcal{L}_2$ that change on π; hence *length*$(\boldsymbol{\sigma}^\pi_j) = 2$, for all $s_j \in$ *edge*$(\mathcal{L}_2)$ that change along π.

Induction hypothesis. Suppose we have a path τ from b in $G'_a(b)$ that is transient-matching with respect to $\mathcal{L}_{2l}$, and has the path-prefix property with respect to $\mathcal{L}_{2l}$, for some l, with $1 \leq l \leq L$.

Induction step. We show that there exists a path π that is transient-matching with respect to $\mathcal{L}_{2l+2}$ and has the path-prefix property with respect to $\mathcal{L}_{2l+2}$.

From the induction hypothesis, τ is transient-matching with respect to $\mathcal{L}_{2l}$, and has the path-prefix property with respect to $\mathcal{L}_{2l}$. Applying Lemma 7.2.4, we construct a path γ that is transient-matching with respect to $\mathcal{L}_{2l}$, hazard-preserving with respect to V, and worst-case with respect to V_{2l}. We then apply Lemma 7.2.6 to obtain a path β that is transient-matching with respect to $\mathcal{L}_{2l}$, hazard-preserving with respect to V and V_{2l}, and worst-case with respect to V_{2l}. By Lemma 7.2.7, β also has the path-prefix property with respect to $\mathcal{L}_{2l+2}$. Path β, however, does not necessarily belong to $G'_a(b)$. By Proposition 5.1.1 we find a path π from b in $G'_a(b)$ that is equivalent to β, so it is transient-matching with respect to

$\mathcal{L}_{2l}$, hazard-preserving with respect to V and V_{2l}, and worst-case with respect to V_{2l}. Then, by Lemma 7.2.1, π is transient-matching with respect to $\mathcal{L}_{2l+2}$. Path π also has the path-prefix property with respect to $\mathcal{L}_{2l+2}$, since β has it. Therefore, the claim of the induction step is true.

Since $\mathcal{S} = \mathcal{L}_{2L+2}$, the claim of the theorem follows. $\square$

In conclusion, we have shown that there exists a path π from b in $G_a(b)$ that satisfies

$$\boldsymbol{\sigma}^{\pi} = \mathbf{s}^H.$$

By the monotonicity of Algorithm Ã, we know that $\mathbf{s}^0 \leq \ldots \leq \mathbf{s}^H$. It follows that, for any state $\mathbf{s}^h$, with $0 \leq h \leq H$, resulting from Algorithm Ã, π satisfies

$$\mathbf{s}^h \leq \boldsymbol{\sigma}^{\pi}.$$

Hence π covers all the states resulting from Algorithm Ã.

By Corollary 4.2.4, $\mathbf{s}^H$ covers all paths in binary analysis, that is, for any path π in $G_a(b)$,

$$\boldsymbol{\sigma}^{\pi} \leq \mathbf{s}^H.$$

Therefore, we can conclude that the results of Algorithm Ã and binary analysis agree (*i.e.*, they cover each other) under the complete network model, for feedback-free circuits of one- and two-input gates. We can refine this result further, by using the insensitivity of Algorithm Ã to input-gate, fork-gate and wire variables (see Chapter 3). Our result can then be reformulated as follows.

Let N be a binary network modeling a feedback-free circuit with one- and two-input gates, and let $\mathbf{N}$ be its transient counterpart. Let $\tilde{N}$ be the complete version of N. Then, the result of Algorithm Ã for $\mathbf{N}$ agrees with the result of binary analysis for $\tilde{N}$ with respect to the initial state variables in N.

7.3 Conclusions

In this chapter we have concluded the process of showing that the result of simulation is covered by the result of binary analysis, in the sense that all the signal changes predicted by the simulation are possible in binary analysis. This covering has been proven using Algorithm $\tilde{A}$ of the simulation (with stable initial state), applied to feedback-free circuits of one- and two-input gates. The result also assumes a complete network model for the binary analysis.

Chapter 8

Conclusions

8.1 Summary

We have studied the simulation method based on algebra C of Brzozowski and Ésik. We have investigated its correctness by comparing it with binary analysis.

We have shown that, for any gate circuit, the result of binary analysis is covered by the result of simulation, under any network model, and for any initial state. Covering is the property that all the changes that happen to state variables in binary analysis are reflected in the result of simulation. This implies that all the changes that correspond to hazard conditions are detected by simulation.

We have also shown that, for feedback-free circuits of one- and two-input gates, the result of simulation is covered by the result of binary analysis carried out in the complete network model, with stable initial state. Here, by covering we mean that all the changes predicted by simulation happen in binary analysis.

These results provide a full characterization of the simulation for feedback-free circuits with one- and two-input gates, and stable initial state. The characterization states that the result of the simulation for any given transient network agrees with the result of the binary analysis for the complete binary version of the given network.

8.2 Open Problems

The only reason for our restriction of the result in Chapter 7 to circuits composed of one- and two-input gates is Lemma 6.3.1. Generalizing that lemma to gates with any number of inputs remains the main open problem of our work. We expect that a generalization of the lemma to basic gates (*i.e.*, gates implementing AND, OR, XOR and their complements) having any number of inputs is possible by an extension of the current proof. For arbitrary functions, however, generalizing the current proof is not feasible, and other approaches need to be explored.

While the need for wire variables in the model used for the result in Chapter 7 is justified, our use of input and fork gates is motivated only mathematically. We believe that the result of Chapter 7 can be strengthened to network models without input and fork gates.

Another improvement that could be made to the result of Chapter 7 is the generalization to any initial state, by using Algorithm A instead of Algorithm Ã.

Our choice for a proof by construction in Chapter 7 has resulted in a very lengthy and complicated argument. An alternative approach to showing the existence of transient-matching paths without explicit construction seems to be more difficult, but it could result in a more elegant proof, so it is definitely worth pursuing.

It was noted in [9] that the simulation using algebra C is impractical for circuits with feedback, since it may not terminate. For such circuits, Brzozowski and Ésik propose the use of different algebras. These algebras are obtained by coupling algebra C with an integer $k > 0$ that functions as a threshold: all transients of length $\geq k$ are mapped to Φ, where Φ designates an unknown, or uncertain value, similar to the one in ternary algebra. For any value of k, we have an algebra C_k. Simulation with algebra C_k involves two passes: Algorithm A followed by Algorithm B. Algorithm A is identical to the one for algebra C, and it is guaranteed to terminate for any circuit. Algorithm B starts with the last state of Algorithm A, and is meant to remove uncertainty; it is also guaranteed to terminate. A study of the simulation with algebra C_k is an open problem. We hope that our work for the simulation with algebra C can provide some useful insights toward such a study.

Bibliography

[1] R. Andrew, “An Algorithm for Eight-Valued Simulation and Hazard Detection in Gate Networks”, in *Proceedings of the 16th IEEE International Symposium on Multiple-Valued Logic*, pp. 273-280, May 1986.

[2] K. van Berkel, *Handshake Circuits: An Asynchronous Architecture for VLSI Programming*, Cambridge University Press, 1993.

[3] M. A. Breuer and R. L. Harrison, “Procedures for Eliminating Static and Dynamic Hazards in Test Generation”, *IEEE Transactions on Computers*, vol. C-23, no. 10, pp. 1069-1078, October 1974.

[4] Erik Brunvand, “The NSR Processor”, in *Proceedings 26th Hawaiian International Conference on System Sciences*, Maui, Hawaii, Jan 1993.

[5] J. A. Brzozowski and M. Yoeli, “On a Ternary Model of Gate Networks”, *IEEE Transactions on Computers*, vol. C-28(3), pp. 178-183, March 1979.

[6] J. A. Brzozowski and C.-J. H. Seger, “A Characterization of Ternary Simulation of Gate Networks”, *IEEE Transactions on Computers*, vol. C-36, no. 11, pp. 1318-1327, November 1987.

[7] J. A. Brzozowski and J. C. Ebergen, “On the Delay-Sensitivity of Gate Networks”, *IEEE Transactions on Computers*, vol. 41, no. 10, pp. 1349-1360, November 1992.

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

[9] J. A. Brzozowski and Z. Ésik, “Hazard Algebras”, Maveric Research Report 00-2, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, July 2000. Revised December 2001.
http://maveric.uwaterloo.ca/publication.html

[10] J. A. Brzozowski, Z. Ésik, and Y. Iland, “Algebras for Hazard Detection”, in *Proceedings of the 31st IEEE International Symposium on Multiple-Valued Logic*, Warsaw, Poland, pp. 3-12, IEEE Computer Society Press, Los Alamitos, CA, May 2001.

[11] S. Chakraborty, D. L. Dill, and K. Y. Yun, “Min-Max Timing Analysis and an Application to Asynchronous Circuits”, *Proceedings of the IEEE*, vol. 87, no. 2, pp. 332-346, February 1999.

[12] T. J. Chaney and C. E. Molnar, “Anomalous Behavior of Synchronizer and Arbiter Circuits”, *IEEE Transactions on Computers*, vol. C-22, pp. 421-422, 1973.

[13] W. S. Coates, J. K. Lexau, I. W. Jones, S. M. Fairbanks, I. E. Sutherland, “A FIFO Data Switch Design Experiment”, in *Proceedings ASYNC ’98*, pp. 4-16, IEEE Computer Society, April 1998.

[14] A. Davis and S. M. Nowick, “An Introduction To Asynchronous Circuit Design”, in *The Encyclopedia of Computer Science and Technology*, A. Kent and J. G. Williams, eds., vol. 38, Marcel Dekker, New York, February 1998.

[15] J. Ebergen and R. Berks, “Response Time Properties of Some Asynchronous Circuits”, in *Proceedings ASYNC ’97*, pp. 76-86, IEEE Computer Society, April 1997.

[16] E. B. Eichelberger, “Hazard Detection in Combinational and Sequential Circuits”, *IBM Journal of Research and Development*, vol. 9, pp. 90-99, 1965.

[17] S. Eilenberg, *Automata, Languages and Machines*, Academic Press, New York, vol. A, 1974.

[18] G. Fantauzzi, "An Algebraic Model for the Analysis of Logical Circuits", *IEEE Transactions on Computers*, vol. C-23, no. 6, pp. 576-581, June 1974.

[19] J. D. Garside, "AMULET3 Revealed", in *Proceedings ASYNC '99*, pp. 51-59, IEEE Computer Society, April 1999.

[20] J. P. Hayes, "Digital Simulation with Multiple Logic Values", *IEEE Transactions on Computer-Aided Design*, vol. CAD-5, no. 2, pp. 274-283, April 1986.

[21] A. Hlawiczka and D. Badura, "The Method of Recognition of Critical Hazards, Critical Races, Essential Hazards and D-trio", in *Proceedings of the 12th IEEE International Symposium on Multiple-Valued Logic*, pp. 298-311, IEEE, 1982.

[22] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

[23] D. A. Huffman, "The Design and Use of Hazard-Free Switching Networks", *Journal of the ACM*, vol. 4, pp. 47-62, January 1957.

[24] J. Kessels and P. Marston, "Designing Asynchronous Standby Circuits for a Low-Power Pager", in *Proceedings ASYNC '97*, pp. 268-278, IEEE Computer Society, April 1997.

[25] A. Koche and E. Brunvand, "Testing Self-Timed Circuits Using Partial Scan", in *Proceedings 2nd Working Conference on Asynchronous Design Methodologies*, pp. 160-169, IEEE Computer Society, May 1995.

[26] D. W. Lewis, "Hazard Detection by a Quinary Simulation of Logic Devices with Bounded Propagation Delays", *M.Sc. Thesis*, Syracuse University, January 1972.

[27] A. J. Martin, "The Design of a Delay-Insensitive Microprocessor: An Example of Circuit Synthesis by Program Transformation", *Hardware Specification, Verification, and Synthesis: Mathematical Aspects, Lecture Notes in Computer Science* vol. 408, pp. 244-259, Springer-Verlag, 1990.

[28] E. J. McCluskey, "Transients in Combinational Logic Circuits", in *Redundancy Techniques for Computing Systems*, R. H. Wilcox and W. C. Mann, eds., Spartan Books, pp. 9-46, 1972.

[29] G. A. Metze, "Many-Valued Logic and the Design of Switching Circuits", *M.Sc. Thesis*, University of Illinois, Urbana, 1955.

[30] R. E. Miller, *Switching Theory, Volume II: Sequential Circuits and Machines*, John Wiley & Sons, 1965.

[31] D. E. Muller and W. C. Bartky, "A Theory of Asynchronous Circuits", in *Proceedings of an International Symposium on the Theory of Switching*, Annals of Computing Laboratory of Harvard University, vol. 29, pp. 204-243, 1959.

[32] M. Peña, J. Cortadella, A. Kondratyev, and E. Pastor, "Formal Verification of Safety Properties in Timed Circuits", in *Proceedings ASYNC 2000*, pp. 2-11, IEEE Computer Society, April 2000.

[33] S. Rotem, K. Stevens, R. Ginosar, P. Beerel, C. Myers, K. Yun, R. Kol, C. Dike, M. Roncken, and B. Agapiev, "RAPPID: An Asynchronous Instruction Length Decoder", in *Proceedings ASYNC '99*, pp. 60-70, IEEE Computer Society, April 1999.

[34] A. Salomaa, *Formal Languages*, Academic Press, 1973.

[35] C.-J. H. Seger and J. A. Brzozowski, "Generalized Ternary Simulation of Sequential Circuits", *Theoretical Informatics and Applications*, vol. 28, no. 3-4, pp. 159-186, 1994.

[36] S. H. Unger, *Asynchronous Sequential Switching Circuits*, John Wiley & Sons, 1969.

Glossary of Symbols

s_i	state variable
S_i	excitation of variable s_i
X_i	input variable
N	network
$\mathcal{X}$	set of input variables
$\mathcal{S}$	set of state variables
$\mathcal{E}$	set of edges
$a \cdot b$	total state: input tuple a, state tuple b
$S_i(a \cdot b)$	value of excitation S_i in total state $a \cdot b$
$\phi(s_i)$	fan-in set of variable s_i
$\psi(s_i)$	fan-out set of variable s_i
$\hat{w}$	contraction of binary word w
$\alpha(\mathbf{t})$	first character of transient $\mathbf{t}$
$\omega(\mathbf{t})$	last character of transient $\mathbf{t}$
$\mathit{length}(\mathbf{t})$	number of characters in transient $\mathbf{t}$
$\mathbf{s}_i$	state variable in the domain of transients
$\mathbf{S}_i$	excitation in the domain of transients
$\mathbf{X}_i$	input variable in the domain of transients
$\mathbf{N}$	transient network

$\mathbf{f}$	extension to transients of Boolean function f
$G_a(b)$	binary analysis graph with input a and initial state b
$G'_a(b)$	subgraph of $G_a(b)$ with no simultaneous changes
s^i	state in binary analysis
$\mathbf{s}^i$	state in simulation
$\boldsymbol{\sigma}_i^\pi$	history of variable s_i along path π
$\mathbf{\Sigma}_i^\pi$	history of excitation S_i along path π
$\Xi(V)$	excitation labels of state variables in set V
$\mathcal{D}_V$	delay automaton of set V
Δ_V	alphabet of $\mathcal{D}_V$
w^π	path-word of path π
$\mathcal{G}_i$	gate automaton for variable s_i
I_i	input alphabet of $\mathcal{G}_i$

List of Mathematical Concepts

List of Definitions

List of Examples

List of Remarks

List of Propositions

List of Lemmas

List of Theorems

List of Corollaries