Delay-Insensitivity and Semi-Modularity[1]

J. A. BRZOZOWSKI
Dept. of Computer Science, University of Waterloo, Waterloo, ON, Canada N2L 3G1
email: brzozo@uwaterloo.ca

H. ZHANG
Dept. of Computer Science, University of Toronto, Toronto, ON, Canada M5S 3G4
email: richard@dgp.utoronto.ca

Abstract. The study of asynchronous circuit behaviors in the presence of component and wire delays has received a great deal of attention. In this paper, we consider asynchronous circuits whose components can be any non-deterministic sequential machines of the Moore type, and describe a formal model for these circuits and their behaviors under the inertial delay model.

We model an asynchronous circuit C by a network N of modules with delays associated with its components and/or wires. We compute the behavior of N assuming arbitrary inertial delays in the modules, and take this behavior to be correct. We define N to be strongly delay-insensitive if its behavior remains correct in the presence of arbitrary stray delays, where correctness is defined through the notion of observational equivalence (or bisimulation), one of the strongest forms of behavioral equivalence. We introduce the notion of quasi semi-modularity, which generalizes Muller's definition of semi-modularity to non-deterministic networks. We prove that a circuit, with all the wire delays taken into account, is strongly delay-insensitive if and only if its behavior is quasi semi-modular.

Keywords: asynchronous, bisimulation, delay-dense, strongly delay-insensitive, isochronic, module, network, semi-modular, quasi semi-modular, speed-independent

1 Introduction

Although much of today's digital design is synchronous, there has been a considerable interest in asynchronous circuits [2], especially during the past decade. In contrast to a synchronous circuit, whose operation is under the control of a global clock signal, an asynchronous circuit uses local handshaking among its components. Potential advantages in using asynchronous circuits include lower energy consumption, higher speed, and avoidance of clock distribution problems [2].

Among asynchronous designs, the class of so-called delay-insensitive circuits has received special attention [3, 4, 5, 15, 16, 17, 18, 19]. Roughly speaking, a circuit is delay-insensitive if it continues to operate correctly even if the delays in its components and wires change arbitrarily. When such a circuit is designed in a modular fashion, it is possible to replace its components by better ones (faster, more power-efficient, etc.), without changing the correctness of its operation, although, its performance may be affected.

[1]This research was supported by the Natural Sciences and Engineering Research Council of Canada under grant No. OGP0000871, and was done while the second author was at the University of Waterloo.

Another important reason for designing delay-insensitive circuits comes from the fact that wire delays do not scale down proportionally to other factors, such as switching time and clock period, affecting the behavior of a circuit, when the size of that circuit is scaled down [13]. This scaling problem has become increasingly troublesome in VLSI design, as it is often desirable to map the existing layout of a circuit onto a smaller region as technology improves.

We represent delays in the commonly-used inertial delay model. In this model, a component ignores a pulse generated on one of its inputs if the duration of the pulse is shorter than the component's switching delay; so an enabled action on a component may be cancelled prematurely if the pulse is too short.

In the next two sections, we give a brief survey of previous work on delay-insensitivity and semi-modularity. For a more detailed account of the two concepts, refer to the recent survey by J. A. Brzozowski [1].

1.1 Semi-modularity and speed-independence

Early work on circuit behaviors in the presence of delays traces back to D. E. Muller's theory of "speed-independent" circuits [7, 11, 12]. Muller considers only *autonomous* circuits, i.e., circuits without external inputs. He assumes that wire delays are negligible, and only components have delays.

A circuit is composed of a set of nodes representing logic components. A *binary* state variable is associated with each node. In Muller's terminology, a node i is *excited*, and thus *unstable*, in a circuit state s if its current value s_i disagrees with its *implied value* s_i'. The implied value is determined by the Boolean expression for that node. Each node has a unique implied value at any time, so it behaves deterministically.

Muller describes the behavior of a circuit by a set of *allowed sequences* of circuit states, which specify the order in which the state variables change, but not the times of change. Two states a and b belong to the same *equivalence class* if a can be reached from b through an allowed sequence, and vice versa. Let $\mathcal{L}$ be the partial order defined over these equivalence classes as follows: $A \mathcal{L} B$ if there exists $a \in A$ and $b \in B$ such that b can be reached from a through an allowed sequence. For each allowed sequence, there is a unique sequence of equivalence classes and a definite "last" class, called the *terminal class*. Muller defines a circuit to be *speed-independent* with respect to a state u if all allowed sequences starting with u have the same terminal class.

In a *semi-modular* circuit behavior, if node i is excited in state s, but does not change to its implied value s_i' when the circuit goes to state t, then it must still be excited in state t and to the same value s_i'. In other words, semi-modularity requires that no state transition can change the implied value of any unchanged state variable which is unstable. The term "semi-modular" was adopted because a certain partially ordered set corresponding to such a behavior is a semi-modular lattice. Muller shows that circuits exhibiting semi-modular behaviors form a *proper* subclass of speed-independent circuits. More recently, A. J. Martin [5] has used the term "stability" to mean semi-modularity, and has basically taken it as a definition of delay-insensitivity.

Semi-modularity has also been related to freedom from static hazards. Intuitively, haz-

ards represent spurious pulses at the outputs of circuit elements. A. Yakovlev et al. [17] define a static hazard under the inertial delay model to be a change of the implied value of an unstable state variable while the variable remains unchanged. So by their definition, semi-modularity implies freedom from static hazards. Martin [5] also states informally that semi-modularity guarantees the absence of hazards.

Yakovlev et al. [17] compare the traditional notion of speed-independence and semi-modularity to some other properties of circuit behaviors, some of which are local (such as determinism, persistence, commutativity, and local confluence), and others (such as permutability and global confluence) are not. They offer alternative definitions of speed-independence and semi-modularity based on these properties. For example, they define speed-independence as global confluence for behaviors which can be infinite, semi-modularity-1 as local confluence, and semi-modularity-2 as the lattice-theoretic semi-modularity. They also prove equivalence results among the various properties for the same behavior. For example, they revise a proof of Muller's to show that determinism, persistence, and commutativity imply semi-modularity-2.

1.2 Delay-insensitivity

C. E. Molnar et al. introduce the "foam-rubber wrapper" postulate [9] to describe delay-insensitive specifications of circuit components. A component is viewed as being surrounded by a "foam-rubber wrapper." The inner surface of the wrapper corresponds to the component interface, whereas the outer surface defines the environment interface. The foam-rubber analogy suggests that the distance between the inner and outer surfaces along one wire may be different from that along another wire, representing different (and possibly time-varying) delays.

Following Molnar's informal idea, J. T. Udding gives the first formal definition of delay-insensitivity [15]. He considers a mechanism interacting with the environment through signals on its input and output wires. He uses a *trace structure* to specify all possible sequences, called *traces*, of communication actions that can take place between the mechanism and its environment. He suggests that such a specification is delay-insensitive if and only if there is no *transmission interference* or *computation interference*. Transmission interference occurs when two consecutive signals are transmitted along a wire, and computation interference occurs when a signal is sent to the mechanism but the mechanism is not ready to receive it, or a signal is sent to the environment but the environment is not ready to receive it.

Udding defines a trace structure to be delay-insensitive if it satisfies four rules, later called the JTU rules. For example, one of the rules states informally that there should not be any ordering among input signals of a mechanism, and the same holds for output signals. Udding shows that a trace structure and its environment satisfy these four rules if and only if there is no computation or transmission interference.

J. C. Ebergen [4] takes a somewhat different approach to defining delay-insensitivity. He specifies a circuit element by a regular expression-like program describing the communication behavior between the element and its environment. Such a program can be translated into a set of simpler programs in a systematic way. Each of the simpler programs represents the behavior of some basic element, such as a fork, a Muller C-element, or a toggle. The network

of these basic elements forms a realization, called *decomposition*, of the original element. If all the basic elements used are so-called delay-insensitive elements, then the circuit element is said to be delay-insensitive. Ebergen has shown that his definition of delay-insensitivity is equivalent to that of Udding.

D. L. Dill [3] develops methods for the specification and automatic verification of asynchronous circuits. He uses a more general notion of trace structure, called *prefix closed trace structure*, which contains a set of *successful traces* and a set of *failure traces*. Trace structures are required to be *receptive*, meaning that the component should be ready to accept input from the environment at any time. A trace structure S is said to *conform to* a trace structure T if S can *safely* substitute for T in a circuit; safe substitution preserves the *failure-freedom* of the trace structures, where a trace structure is failure-free if its failure set is empty. Dill defines delay-insensitivity via an operator **DI** on trace structures. Basically, **DI** attaches delays to all the inputs and outputs of a component specified by the trace structure (a foam-rubber wrapper). A trace structure T is said to be delay-insensitive if and only if $\mathbf{DI}(T)$ conforms to T.

T. Verhoeff [16] calls an asynchronous circuit component a *process*. A process is specified by a trace structure, as used by Udding. Verhoeff shows that several characterizations of delay-insensitivity are equivalent under certain assumptions. He also extends the JTU Rules to include progress as a correctness concern.

1.3 Contribution of the paper

1.3.1 The network model

In this paper, we consider asynchronous networks whose components are non-deterministic sequential machines of the Moore type [10]. Non-determinism permits us to include arbiters in our networks. We also allow components to have multiple outputs. We describe a formal model for these networks and their behaviors.

Yakovlev et al. [17] also develop a model which allows non-deterministic circuit components. The structural property of a circuit is captured by an *Asynchronous Control Structure (ACS)*, and the behavior of the circuit is described by a *State Transition Diagram (STD)*. There are at least three differences between their model and ours:

- We can associate state variables to components and/or wires any way we want. But an ACS is basically a "wire-state" model, since state variables are associated with wires only. So component delays are incorporated into wire delays. Also, an STD may be *contradictory*, meaning that two states in the STD which are identical in value could in fact represent different physical states. We make such distinctions explicit.

- In their model, only the behavior of an ACS is given (as an STD). So the components of the ACS are in a sense abstract; their behaviors can only be inferred from the STD. In our model, component behaviors are completely specified in the form of finite state machines, and the network behavior is implied.

- We use the notion of "pre-programmed non-determinism," where non-determinism is confined within a single module. This is achieved by expanding the excitation from

a single value to a set of values. In the model of Yakovlev et al., non-determinism is implied from the STD specification.

1.3.2 Strong delay-insensitivity of networks

Recall the intuitive notion of delay-insensitivity. A circuit is delay-insensitive if it continues to operate correctly even if the delays in its components and wires change arbitrarily. All the existing definitions of delay-insensitivity are concerned with a component interacting with its environment. The component is said to behave delay-insensitively if the variations of delays in its input and output wires do not violate its specification, which is typically given in the form of a trace structure. These definitions focus on the delay-insensitivity of communications between two parties (the component and its environment). Also, component delays are incorporated into wire delays.

In this paper, we take a different approach. We are concerned with the insensitivity of a *network* of components to wire delays.

Consider the following scenario. A designer has come up with a preliminary design of an asynchronous circuit. This design consists of a collection of logic components, which we call *modules*, and these components are interconnected by wires, which we call *connections*. Our model is quite flexible, for it permits the designer to use low-level modules, like logic gates, or higher-level modules, like counters and arbiters. Each module is represented by its binary inputs and outputs, by a single multi-valued state variable (to keep the model simple and to abstract the internal design details of the module), and by its next-state and output functions. The state variable permits us to associate a delay with each component. To keep our model general, we assume that all such delays are arbitrary, finite, inertial delays.

As to wire delays, our model offers several choices. For simplicity, the designer might want to assume that only the logic components have delays, i.e., to use a speed-independent model for the first analysis. Or, if it is expected that some of the connections may be long and may have appreciable delays, the designer might model such wire delays as delay modules. Finally, if a more conservative model is desired, *all* wires may be assumed to have delays, in which case each wire will have a delay module inserted in it. In summary, the first model of the circuit to be designed is a network N of modules, some of which are logic components and others delay modules, as determined by the designer. We call these modules the *original* modules.

It should be noted that, since delays are modules, a network with some wire delays added is still a network of modules, and the same analysis method applies to both the original network, and the network with added delays.

Assume now that the behavior of network N obtained as above has been analyzed and that it satisfies the given specification. We are interested in finding out whether the presence of *stray* delays, which were not taken into account in N, can cause incorrect behavior. For this purpose, we study *delay extensions* of N. A delay extension $\hat{N}$ of N is a network obtained from N by inserting any number of (stray) delays in the connections of N. Note that several delays may be inserted in any connection. We now compare the behaviors of N and $\hat{N}$. Since $\hat{N}$ has additional state variables corresponding to the stray delays, we must assume that changes in the stray delays are unobservable. Having done that, we insist that

the behaviors of N and $\hat{N}$ (ignoring the stray delays) should be the same. If the behavior of N agrees with the behavior of $\hat{N}$ *for each delay extension* $\hat{N}$ of N, then we declare N to be strongly delay-insensitive.

It remains to define what we mean by the statement "the behaviors of N and $\hat{N}$ should be the same." Here, we use the notion of *observational equivalence*, also known as *bisimulation*. This is one of the strongest forms of behavioral equivalence, since it requires that N and $\hat{N}$ should be capable of simulating each other's behaviors step-by-step. A similar approach has been used by N. Shintel and M. Yoeli [14], but with a different motivation and in a different context.

One of our main results requires that the network under consideration should be *delay-dense*, meaning that, of each pair of adjacent components of the network, at least one is a delay. For example, to obtain a delay-dense model it suffices to insert a delay in each wire; a network with all wire delays taken into account is called *delay-complete*. We argue that the delay-complete model is the appropriate one, if we are interested in studying delay-insensitivity. Clearly, if one is interested in delay-insensitivity, one should not assume that wires have zero delay.

1.3.3 Quasi semi-modularity and strong delay-insensitivity

To permit the handling of non-deterministic modules, we generalize Muller's definition of semi-modularity. We choose the term *quasi semi-modularity* for our new concept.

The main result of the paper is that for delay-dense networks, strong delay-insensitivity is equivalent to quasi semi-modularity.

1.4 Organization of the paper

The paper is organized as follows. In Section 2, we present our model for modules, whereas in Section 3, we discuss networks of modules and network behaviors. Sections 4 and 5 concentrate on the definitions of strong delay-insensitivity of networks and quasi semi-modularity of network behaviors, respectively. In Section 6, we give the proof of our main result. Section 7 concludes the paper.

2 Modules

We now introduce our model of asynchronous components, which we call "modules." To hide the details of the internal design of a module, we represent it only by an abstract internal state, which is not necessarily binary. This also helps to keep the model simple. However, the inputs and outputs of a module remain binary.

We do not introduce delays in the input and output wires of a module for at least two reasons. First, wire delays are frequently ignored in modules designed on a small area of a chip; this small area is referred to as an "equipotential region" by Seitz [13]. Second, we want to have the ability to model isochronic forks and similar components, since they are used in many practical designs.

As a convention, if $x = (x_1, \ldots, x_n)$ is an ordered n-tuple, then $\mathcal{X} = \{x_1, \ldots, x_n\}$ is the corresponding set. Also, if $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of elements explicitly represented in that order, then $x = (x_1, \ldots, x_n)$ is the corresponding n-tuple.

Definition 2.1 A *module* M is a sequential machine $M = \langle \mathcal{S}, \mathcal{X}, y, \mathcal{Z}, \delta, \lambda \rangle$, where

- $\mathcal{S}$ is a finite set of *internal states* of M;
- $\mathcal{X} = \{x_1, \ldots, x_m\}$ is the set of *binary input variables*, where $m \geq 0$;
- y is the *internal state variable*;
- $\mathcal{Z} = \{z_1, \ldots, z_p\}$ is the set of *binary output variables*, where $p \geq 0$;
- δ is the *excitation function*, $\delta : \{0,1\}^m \times \mathcal{S} \rightarrow 2^{\mathcal{S}} \setminus \{\emptyset\}$, satisfying the restriction that, for any $a \in \{0,1\}^m$ and $b \in \mathcal{S}$, either $\delta(a,b) = \{b\}$ or $b \notin \delta(a,b)$;
- $\lambda = (\lambda_1, \ldots, \lambda_p)$ is the *output function*, $\lambda : \mathcal{S} \rightarrow \{0,1\}^p$. □

Note that $m = 0$ is allowed; in this case, the module is a *source*. Similarly, $p = 0$ is allowed, and the module is a *sink*. A module M can be represented by a directed graph $G = \langle \mathcal{V}, \mathcal{E} \rangle$, where $\mathcal{V} = \mathcal{X} \cup \{y\} \cup \mathcal{Z}$, and $\mathcal{E} = (\mathcal{X} \times \{y\}) \cup (\{y\} \times \mathcal{Z})$, as shown in Figure 2.1.

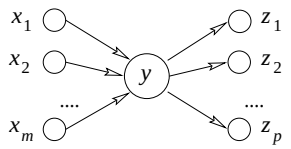

Figure 2.1: Graphical representation of a module.

A *total state* t of module M is a pair (a, b), where a is a binary m-tuple representing input values, and $b \in \mathcal{S}$ represents the internal state. The set $T = \delta(t)$ is an non-empty subset of $\mathcal{S}$, called the *excitation state set*, or simply, the *excitation* of M in state t. If $T = \{b\}$, then the state t and the module M are said to be *stable*, and the internal state cannot change. If $T \neq \{b\}$, then t and M are *unstable* and, at any time, the internal state may change to any state $b' \in T$ non-deterministically selected by the module. If the cardinality of T is always 1, the module is said to be *deterministic*. *In that case, we view δ as a function from the set of total states to $\mathcal{S}$.*

The output of the module M is determined solely by the present internal state. If the internal state changes, and the implied output differs from the output before the change, the new output value appears *instantaneously* together with the internal state change.

Definition 2.1 permits us to treat any non-deterministic sequential machine of the Moore type [10] as a module. In particular, delays (or wires with delays), forks, logic gates, latches, counters, C-elements, and arbiters are included. Note that in this model forks are *isochronic*, meaning that all the output branches change at exactly the same time. The delay of the internal state of a fork becomes the common delay of its output branches. *Anisochronic*

forks can be modelled within a network by associating delays to the wires connected to the fork outputs; see Section 3.

Example 2.1 A delay is a module $M^d = \langle \{0,1\}, \{x_1^d\}, y^d, \{z_1^d\}, \delta^d, \lambda^d \rangle$ with δ^d and λ^d defined as in Table 2.1. For future applications, we often use the superscript d to identify a delay module. Note that we could also define a module that behaves like a delay but has several outputs; however, the term *delay module*, or *delay*, will be reserved for one-output delays.

Table 2.1: Functions of a delay

	x		
y	0	1	$\lambda^d(y)$
0	0	1	0
1	0	1	1
	$\delta^d(x,y)$		

Table 2.2: Functions of a fork

	x		
y	0	1	$\lambda(y)$
0	0	1	00
1	0	1	11
	$\delta(x,y)$		

Example 2.2 An (isochronic) fork has $\mathcal{S} = \{0,1\}$, $m = 1$, $p = 2$, and δ and λ as defined in Table 2.2.

Example 2.3 A three-input majority element has $\mathcal{S} = \{0,1\}$, $m = 3$, and $p = 1$.

Table 2.3: Functions of a three-input majority element

	x								
y	000	001	010	011	100	101	110	111	$\lambda(y)$
0	0	0	0	1	0	1	1	1	0
1	0	0	0	1	0	1	1	1	1
	$\delta(x,y)$								

Example 2.4 A set-dominant set-reset latch has $\mathcal{S} = \{0,1\}$, $m = 2$, $p = 2$, and δ and λ as defined in Table 2.4.

Example 2.5 A Muller C-element has $\mathcal{S} = \{0,1\}$, $m = 2$, $p = 1$, and δ and λ as defined in Table 2.5.

Table 2.4: Functions of a set-reset latch

	x				
y	00	01	10	11	$\lambda(y)$
0	0	0	1	1	01
1	1	0	1	1	10
	$\delta(x,y)$				

Table 2.5: Functions of a Muller C-element

	x				
y	00	01	10	11	$\lambda(y)$
0	0	0	0	1	0
1	0	1	1	1	1
	$\delta(x,y)$				

Example 2.6 A modulo-4 counter has $\mathcal{S} = \{0,1,2,3\}$, $m = 1$, $p = 2$, and δ and λ as defined in Table 2.6.

Table 2.6: Functions of a modulo-4 counter

	x		
y	0	1	$\lambda(y)$
0	0	1	00
1	2	1	01
2	2	3	10
3	0	3	11
	$\delta(x,y)$		

Table 2.7: Functions of an arbiter

	x				
y	00	01	10	11	$\lambda(y)$
0	$\{0\}$	$\{2\}$	$\{1\}$	$\{1,2\}$	00
1	$\{0\}$	$\{0\}$	$\{1\}$	$\{1\}$	10
2	$\{0\}$	$\{2\}$	$\{0\}$	$\{2\}$	01
		$\delta(x,y)$			

Example 2.7 A simple arbiter has $\mathcal{S} = \{0,1,2\}$, $m = 2$, $p = 2$, and δ and λ as defined in Table 2.7. This module is non-deterministic, functioning as an arbitration device. When both inputs are 1 and the module is in state 0, the next state is chosen to be either 1 or 2, arbitrarily.

3 Networks

We now introduce our circuit model, which we call a "network." Networks are composed of interconnected modules. We restrict our analysis to closed, or autonomous, networks. Typically, an open network can be transformed into a closed one by modeling the environment as a set of modules. For example, to analyze the behavior of a two-way arbitration device interacting with its environment, we can model the two agents sending and receiving requests as two modules. If an agent may send a new request only when its previous request has been granted, then we can model this simply as a delay module. If requests may be sent arbitrarily, then we can use a binary module whose excitation is always the inverse of its present internal state; note that in this case, the module is a source. Also, we can use a non-deterministic module with two outputs to model random requests from the environment.

We use the following conventions. Modules are indexed by superscripts; all the components, such as the state variables, associated with module M^i are given the superscript i. Also, specific components of a tuple are indexed using subscripts, unless otherwise specified.

Definition 3.1 A *network* N is a pair $\langle \mathcal{M}, \mathcal{K} \rangle$, where

- $\mathcal{M} = \{M^1, \ldots, M^n\}$, $n \geq 1$, is a set of modules;
- $\mathcal{K}$ is a set of *connections*, each of which is an ordered pair (z_i, x_j), where z_i is the output variable of some module, and x_j is the input variable of some module. Moreover, for each input variable x_j (respectively, for each output variable z_i), there is exactly one output variable z_i (respectively, one input variable x_j) such that $(z_i, x_j) \in \mathcal{K}$. □

We assume that all networks in consideration are connected as graphs. Note that we do not allow wired-and or wired-or connections; these are replaced by corresponding multi-input modules. Normally, an output z from a module M can be distributed to k other modules through a fork, for some $k > 1$. Our model also permits this in one of two ways: We can either use a fork module, or we can replace M by a module in which the output z is replaced

by k copies of z. Figure 3.1 shows a network composed of a C-element and a two-output inverter, where the two outputs of the inverter are simply replicas of each other.

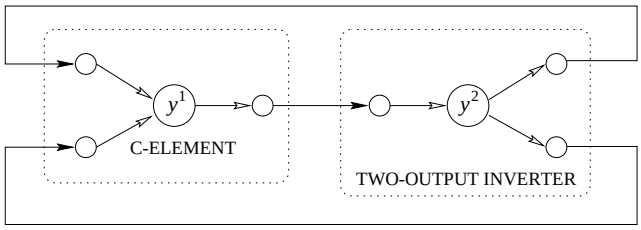

Figure 3.1: An example of a network.

The set of state variables of the network N is $\mathcal{Y} = \{y^1, \ldots, y^n\}$. A state of N is a combination of the internal states of all the modules in N. So the set of states of N is $\mathcal{S} = \mathcal{S}^1 \times \ldots \times \mathcal{S}^n$. If $s \in \mathcal{S}$, the i-th component of s is denoted by s_i.

Definition 3.2 Let $y^i \in \mathcal{Y}$. The *network excitation function* $\Delta_i : \mathcal{S} \to 2^{\mathcal{S}^i} \setminus \{\emptyset\}$ of y^i, is the module excitation function δ^i of M^i with arguments changed as follows. If (z_k^h, x_l^i) is a connection, then the l-th input argument of δ^i is $\lambda_k^h(y^h)$. For $y^i \in \mathcal{Y}$ and $s \in \mathcal{S}$, the *excitation* of y^i in state s, denoted by S_i, is $S_i = \Delta_i(s)$.

If (z_k^h, x_l^i) is a connection, the output function λ^h performs an instantaneous encoding of the state of module M^h and provides the k-th bit of the encoded value as the l-th input to module M^i. Since λ^h depends solely on y^h, the excitation function Δ_i becomes a function of y^h. This functional dependency is captured by the "feed" relation: A module M^i is said to *feed* a module M^j, denoted by $(M^i, M^j) \in \mathcal{F}$, if some output of M^i and some input of M^j form a network connection. Note that the case where $i = j$ (a self-loop) is allowed. The relation $\mathcal{F}$ can also be viewed as being defined over the set of state variables. That is, $(y^i, y^j) \in \mathcal{F}$ if and only if $(M^i, M^j) \in \mathcal{F}$.

A network state s is stable if all the modules are stable in s, that is, if $S_i = \{s_i\}$ for all i; otherwise, s is *unstable*. We denote the set of unstable state variables in state s by

$$\mathcal{U}(s) = \{y^i \in \mathcal{Y} \mid S_i \neq \{s_i\}\}.$$

We now define the behavior of a network. The basic notions are those of Muller [11], though we prefer to use the terminology in [2]. For simplicity and other reasons, we assume that only one state variable changes at a time. This is essentially the *general single-winner* (GSW) model [2]. We support our decision as follows. Since we are mainly interested in delay-insensitive circuits, for which wire delays have to be considered, it would seem inappropriate to choose a model in which simultaneous input or output changes play a significant role. Also, for semi-modularity, the other network property we are concerned with, we have the observation that for a semi-modular behavior, the GSW model produces the same terminal class as the *general multiple-winner* model, in which simultaneous variable changes are allowed. This is because semi-modularity requires that no unstable state variable

can become stable without changing its value. Thus, variables changing simultaneously can always be made to change sequentially.

The *GSW relation* over the set of states of a network N is a binary relation $\mathbf{R}$, such that $(s,t) \in \mathbf{R}$, or $s\mathbf{R}t$, if and only if s differs from t in exactly one component i, $s_i \neq t_i$, $y^i \in \mathcal{U}(s)$, and $t_i \in S_i$.

Definition 3.3 A (GSW) *behavior* of a network N is an initialized directed graph $B = \langle q, \mathcal{Q}, \mathcal{R} \rangle$, where

- $q \in \mathcal{S}$ is the initial state;
- $\mathcal{Q}$, specifying the vertices of B, is the set of states reachable from q via the GSW relation $\mathbf{R}$,
$$\mathcal{Q} = \{s \in \mathcal{S} \mid q\mathbf{R}^* s\};$$
- $\mathcal{R}$, specifying the edges of B, is the GSW relation $\mathbf{R}$ restricted to $\mathcal{Q}$.

If $(s,t) \in \mathcal{R}$, we attach to edge (s,t) a *tag* $\tau(s,t) \in \mathcal{Y}$, which denotes the state variable in which s and t differ. □

With the notation for behaviors established, we have the following proposition, which merely states that if the excitation of a state variable y^j changes due to a change of variable y^i, then y^i must feed y^j.

Proposition 3.1 *Let $(s,t) \in \mathcal{R}$ and $\tau(s,t) = y^i$. If $S_j \neq T_j$ for some $j \neq i$, then $(y^i, y^j) \in \mathcal{F}$.*

Proof: Since $j \neq i$, we have $s_j = t_j$. Thus, if $S_j \neq T_j$, the value of some input argument of Δ_j must have changed during the state transition. As the only state variable changed is y^i, we must have $(y^i, y^j) \in \mathcal{F}$. □

A behavior $B = \langle q, \mathcal{Q}, \mathcal{R} \rangle$ of a network N can be viewed as a non-deterministic finite automaton $\mathcal{B} = \langle \mathcal{Y}, \mathcal{Q}, q, \rho, \mathcal{Q} \rangle$, where

- $\mathcal{Y}$ is the input alphabet;
- $\mathcal{Q}$ is the state set;
- q is the initial state;
- ρ is the state transition function, $\rho : \mathcal{Q} \times \mathcal{Y} \to 2^{\mathcal{Q}}$, such that
$$\rho(s, y^i) = \{t \in \mathcal{Q} \mid s\mathcal{R}t \text{ and } \tau(s,t) = y^i\}$$
for $s \in \mathcal{Q}$ and $y^i \in \mathcal{Y}$;
- $\mathcal{Q}$ is the set of accepting states.

Note that $\rho(s, y^i) = \emptyset$ implies that there is no transition labelled y^i leaving state s.

We extend the state transition function ρ to $\rho^* : \mathcal{Q} \times \mathcal{Y}^* \to 2^{\mathcal{Q}}$, which can be defined inductively as follows. For $s \in \mathcal{Q}$, $y^i \in \mathcal{Y}$, and $w \in \mathcal{Y}^*$,

- $\rho^*(s, \epsilon) = \{s\}$;
- $\rho^*(s, wy^i) = \{t \in \mathcal{Q} \mid t \in \rho(s', y^i) \text{ for some } s' \in \rho^*(s, w)\}$.

We often represent the state set $\rho^*(s, w)$ by the short-hand form sw. The language $L(\mathcal{B})$ accepted by $\mathcal{B}$ is defined in the usual way,

$$L(\mathcal{B}) = \{w \in \mathcal{Y}^* \mid qw \neq \emptyset\}.$$

It is worth noting that $L(\mathcal{B})$ is always prefix-closed.

Example 3.1 Refer to the network shown in Figure 3.1. The set of state variables is $\mathcal{Y} = \{y^1, y^2\}$. The behavior of this network with initial state $(y^1, y^2) = (0, 0)$ is shown in Figure 3.2. From now on, when showing particular state tuples, we omit parentheses and commas, for brevity; also, unstable state components are in boldface.

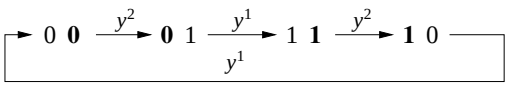

Figure 3.2: Behavior of the network shown in Figure 3.1.

Example 3.2 Figure 3.3 shows a network N' composed of a two-output C-element and two inverters. The two outputs of the C-element are simply replicas of each other. The behavior of N' with initial state 0**00** is shown in Figure 3.4.

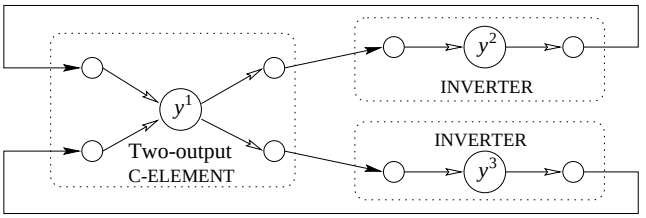

Figure 3.3: Network N'.

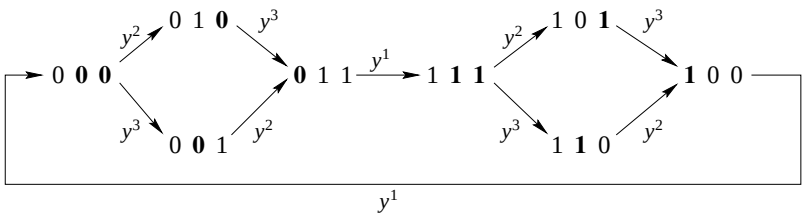

Figure 3.4: Behavior of network N'.

4 Strong delay-insensitivity of networks

We now describe how we model stray delays in a network. We then give a formal definition of strong delay-insensitivity of networks.

4.1 Delay extensions

To determine whether a network is delay-insensitive, we wish to examine how the distribution of delays in the wires affects the behavior of the network. We motivate our approach to this problem by Charles Molnar's "foam-rubber wrapper" principle [9]. This intuitive idea corresponds to surrounding each module in a network by delays. However, instead of "wrapping" all the modules by delays at once, we choose to proceed inductively using the notion of *delay extension*. This facilitates the mathematical treatment of the problem.

Definition 4.1 Let $N = \langle \mathcal{M}, \mathcal{K} \rangle$ be a network. A *delay successor* of N is a network $\hat{N} = \langle \hat{\mathcal{M}}, \hat{\mathcal{K}} \rangle$ obtained from N by inserting a delay M^{n+1} in a connection $e = (z_k^i, x_l^j)$ of N. That is, z_k^i is connected to the input of M^{n+1}, the output of M^{n+1} is connected to x_l^j, and the connection e is removed. The set $\mathcal{D}(N)$ of *delay extensions of a network* N is defined inductively, as follows,

- $N \in \mathcal{D}(N)$;
- if $\hat{N} \in \mathcal{D}(N)$, then each delay successor of $\hat{N}$ is also in $\mathcal{D}(N)$. □

By definition, there cannot be a cycle of inserted delays in a delay extension. This fact is used in the proofs of Propositions 4.2 and 4.3 below. As a convention, we denote a delay extension of a network N by $\hat{N}$. Furthermore, all the components of $\hat{N}$, any object associated with $\hat{N}$, e.g., a behavior, and the components of that object are all marked by the hat $\hat{}$.

In order to compare the behavior of a network N with the behavior of a delay extension of N, it is essential that we relate corresponding states and state transition sequences in the two behaviors.

Definition 4.2 Let $\hat{N} \in \mathcal{D}(N)$. The *projection of a state* $\hat{s}$ of $\hat{N}$ with respect to N is the $|\mathcal{Y}|$-tuple $s = \hat{s} \downarrow \mathcal{Y}$ obtained from $\hat{s}$ by removing all components corresponding to variables not in $\mathcal{Y}$. Similarly, the *projection of a word* $\hat{w} \in \hat{\mathcal{Y}}^*$ with respect to N is the word $w = \hat{w} \downarrow \mathcal{Y}$ obtained from $\hat{w}$ by erasing all the symbols not in $\mathcal{Y}$.

Note that in general, any module can be "projected out"; however, for the purpose of our analysis, only delays are projected out. In contrast to projections of states, we also have the notion of extensions of states.

Definition 4.3 Let $\hat{N} \in \mathcal{D}(N)$, and let s be a state of N. An *extension* of s with respect to $\hat{N}$ is a state $\hat{s}$ of $\hat{N}$ for which $\hat{s} \downarrow \mathcal{Y} = s$. If $\hat{s}$ is an extension of s and $\mathcal{U}(\hat{s}) \subseteq \mathcal{Y}$ (in other words, if all the inserted delays are stable in $\hat{s}$), then we say that $\hat{s}$ is the *stable-delay extension* of s.

Note that the stable-delay extension of a state is unique. We now prove two useful propositions concerning state extensions. In the following, $\hat{N} \in \mathcal{D}(N)$, and s and $\hat{s}$ are states of N and $\hat{N}$, respectively. We call a module of $\hat{N}$ which is also present in N an *original* module.

Proposition 4.1 *If $\hat{s}$ is the stable-delay extension of s, then the excitations of all the original modules are the same in s and $\hat{s}$, that is, $S_i = \hat{S}_i$ for all i such that $M^i \in \mathcal{M}$.*

Proof: By Definition 3.2, it suffices to prove that in state s, the value of each argument of Δ_i agrees with that of the corresponding argument of $\hat{\Delta}_i$ in state $\hat{s}$. The arguments of Δ_i are the input values to M^i and the internal state s_i. Since $s_i = \hat{s}_i$, we only need to worry about the input arguments. Let the k-th input argument of $\hat{\Delta}_i$ be $\lambda_l^h(\hat{s}_h)$. If M^h is original, then we are done. Otherwise, M^h is an inserted delay. Then there exists a "chain" of inserted delays of which M^h is the tail, and some original module M^g feeds the head of the chain. As $\hat{s}$ is the stable-delay extension of s, all the delays in the chain are stable in $\hat{s}$. Thus, $\lambda_j^g(\hat{s}_g) = \hat{s}_h$, where we assume that the j-th output of M^g feeds the head of the delay chain. Furthermore, $\lambda_l^h(\hat{s}_h) = \hat{s}_h = \lambda_j^g(\hat{s}_g)$. On the other hand, it is clear that $(M^g, M^i) \in \mathcal{F}$ and the value of the k-th input argument of Δ_i is $\lambda_j^g(s_g) = \lambda_j^g(\hat{s}_g)$, which is the same as that of $\hat{\Delta}_i$. □

Corollary 4.1.1 *If $\hat{s}$ is the stable-delay extension of s, then $\mathcal{U}(\hat{s}) = \mathcal{U}(s)$.*

Proposition 4.2 *Let $\hat{s}$ be an extension of s; then the stable-delay extension of $\hat{s}$ can be reached from $\hat{s}$ by a sequence of changes on the inserted delays only. Formally, there exists $\hat{w} \in (\hat{\mathcal{Y}} - \mathcal{Y})^*$ such that $\hat{s}' \in \hat{s}\hat{w}$, where $\hat{s}'$ is the stable-delay extension of s.*

Proof: Let us assign "levels" to the inserted delays in $\hat{N}$ inductively as follows: a) A delay is of level 1 if it is fed by an original module; b) a delay is of level $k+1$ if it is fed by a delay of level k. Note that there are no cycles of inserted delays in $\hat{N}$. Also, every delay has a unique input and output. Therefore, each inserted delay is assigned a unique level. Starting in state $\hat{s}$, we change inserted delays which are unstable in such a way that at each step, we always choose one which has the least level. We stop when all the inserted delays are stabilized. One easily verifies that the process will terminate when the stable-delay extension $\hat{s}'$ of s is reached. □

4.2 Definition of strong delay-insensitivity of networks

Given a network N, we compute its behavior with respect to its initial state, and take this behavior to be correct. The delay extensions of N give a description of all possible distribution of stray wire delays within the network. For N to behave correctly regardless of the delay distributions, that is, for N to be delay-insensitive, the behavior of any delay extension of N should also be correct. Thus, any delay extension of N should behave like N as long as their initial states are "compatible."

One of the strongest forms of behavioral equivalence is *observation equivalence* [8], which we choose as the criterion for comparing the behavior of a network with the behavior of its delay extension, and for defining strong delay-insensitivity of networks. In this context,

transitions occurring on the inserted delays are treated as non-observable "internal" actions; in fact, they can be viewed simply as a passing of time. Note that observation equivalence is also often referred to as *weak bisimulation*. Strong bisimulation corresponds to graph isomorphism.

As noted above, in order to compare the behavior of a network with the behavior of a delay extension of the network, we have to choose an appropriate "compatible" initial state for the latter behavior. The following definition makes this precise.

Definition 4.4 Let $\hat{N}$ be a delay extension of N, and let $B = \langle q, \mathcal{Q}, \mathcal{R} \rangle$ be a behavior of N. A behavior $\hat{B} = \langle \hat{q}, \hat{\mathcal{Q}}, \hat{\mathcal{R}} \rangle$ of $\hat{N}$ is said to be *initial-state compatible with* B if $\hat{q}$ is the stable-delay extension of q.

In general, two behaviors are observationally equivalent if they can simulate each other in a step-by-step fashion when transitions on the non-observable variables are ignored. More specifically, if two states from the two behaviors are similar, meaning that they agree in value on the observable variables, then an observable transition enabled in one state can also be observed from the state in the other behavior, and vice versa. Moreover, it is undesirable to have a behavior going into an infinite loop without producing any observable transitions. We call such a loop a *livelock*. We now define these concepts in the context of network behaviors.

Definition 4.5 Let $\hat{N}$ be a delay extension of N. Let $B = \langle q, \mathcal{Q}, \mathcal{R} \rangle$ be a behavior of N, and let $\hat{B} = \langle \hat{q}, \hat{\mathcal{Q}}, \hat{\mathcal{R}} \rangle$ be the behavior of $\hat{N}$ which is initial-state compatible with B. Two states $s \in \mathcal{Q}$ and $\hat{s} \in \hat{\mathcal{Q}}$ are said to be *similar* if $\hat{s}$ is an extension of s.

- Behavior $\hat{B}$ is said to be *safe with respect to* B if, whenever $\hat{s} \in \hat{\mathcal{Q}}$ and $s \in \mathcal{Q}$ are similar, then for all $\hat{t} \in \hat{\mathcal{Q}}$ and $\hat{w} \in \hat{\mathcal{Y}}^*$, if $\hat{t} \in \hat{s}\hat{w}$ and $\hat{w} \downarrow \mathcal{Y} = y^i$ for some $y^i \in \mathcal{Y}$, then there exists $t \in \mathcal{Q}$ such that $t \in sy^i$ and t and $\hat{t}$ are similar. That is,

$$\hat{s} \xrightarrow{\text{unobservable}} \hat{u} \xrightarrow{y^i} \hat{v} \xrightarrow{\text{unobservable}} \hat{t} \implies s \xrightarrow{y^i} t.$$

- Behavior $\hat{B}$ is said to be *complete with respect to* B if, whenever $\hat{s} \in \hat{\mathcal{Q}}$ is an extension of $s \in \mathcal{Q}$, then for all $t \in sy^i$, there exist $\hat{w} \in \hat{\mathcal{Y}}^*$ and $\hat{t} \in \hat{\mathcal{Q}}$ such that $\hat{t} \in \hat{s}\hat{w}$, $\hat{w} \downarrow \mathcal{Y} = y^i$, and t and $\hat{t}$ are similar. That is,

$$s \xrightarrow{y^i} t \implies \hat{s} \xrightarrow{\text{unobservable}} \hat{u} \xrightarrow{y^i} \hat{v} \xrightarrow{\text{unobservable}} \hat{t}.$$

- Behavior $\hat{B}$ is said to be *livelock-free with respect to* B if all sequences of transitions involving only the inserted delays are finite. Formally, for all $\hat{s} \in \hat{\mathcal{Q}}$ and $\hat{w} \in (\hat{\mathcal{Y}} - \mathcal{Y})^+$, $\hat{s} \notin \hat{s}\hat{w}$. □

With unobservable transitions occurring on delay modules only, it can be shown that completeness and livelock-freedom are satisfied by all delay extensions.

Proposition 4.3 *Behavior $\hat{B}$ of Definition 4.5 is always complete and livelock-free with respect to B.*

Proof: Refer to Appendix A.1. □

This results in our definition of strong delay-insensitivity of networks.

Definition 4.6 A network N is *strongly delay-insensitive with respect to a state q* if, for any delay extension $\hat{N}$ of N, $\hat{B}$ is safe with respect to B, where B and $\hat{B}$ are as defined in Definition 4.5.

Example 4.1 Consider part of a network N, where two delays are connected to the arbiter of Example 2.7, as shown in Figure 4.1(a). Figure 4.1(c) shows the corresponding part of a delay successor $\hat{N}$ of N, where the inserted delay is shaded. Let the current values and excitations of the modules be as shown, where excitations are shown in brackets.

Each figure represents a network state. State t of N, as shown in Figure 4.1(b), results from state s (Figure 4.1(a)) by a delay change; state $\hat{t}$ of $\hat{N}$ (Figure 4.1(d)) results from state $\hat{s}$ (Figure 4.1(c)) by the same delay change; also, $\hat{s}$ is the stable-delay extension of s. We assume that the states of network modules not shown are the same in all four states.

Clearly, $\hat{t}$ is an extension of t. In state $\hat{t}$, the arbiter may change to state 1 or 2, whereas in state t, the arbiter can only change to 2. This demonstrates a violation of safety. Therefore, N is not strongly delay-insensitive with respect to state s. □

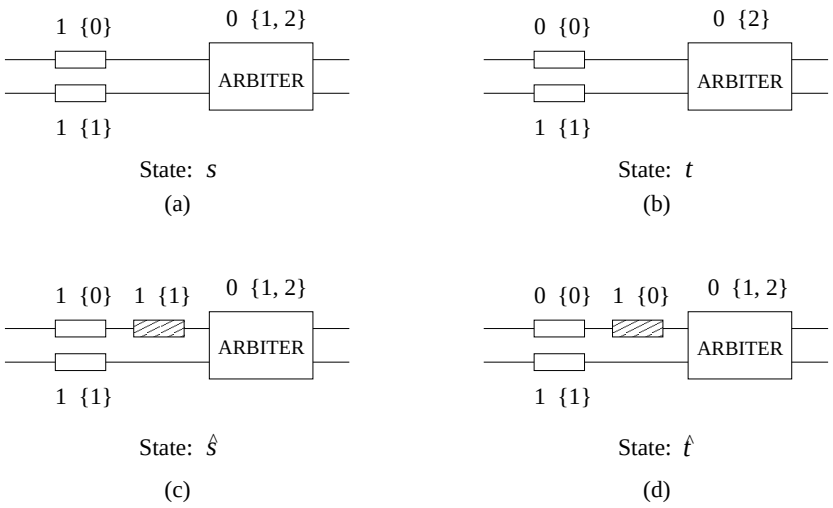

Figure 4.1: Example of safety violation.

5 Quasi semi-modularity and delay-dense networks

In this section, we first generalize Muller's definition of semi-modularity to quasi semi-modularity to allow non-deterministic modules. We compare quasi semi-modularity with semi-modularity. We then define a special class of networks, the delay-dense networks, which characterize asynchronous circuits for the purpose of delay-insensitivity analysis. We prove that for delay-dense networks, quasi semi-modularity is preserved under delay extensions.

5.1 Quasi semi-modularity and semi-modularity

Semi-modularity was initially defined in [11] for deterministic circuit behaviors. We generalize this original definition to suit our network model, which allows non-deterministic modules.

Definition 5.1 Let $B = \langle q, \mathcal{Q}, \mathcal{R} \rangle$ be a behavior of a network N. A state $s \in \mathcal{Q}$ is said to be *quasi semi-modular* if, for all $t \in \mathcal{Q}$, if $(s,t) \in \mathcal{R}$ with $\tau(s,t) = y^i$, then $S_j \subseteq T_j$ for all $j \neq i$ such that $y^j \in \mathcal{U}(s)$. If s is quasi semi-modular for all $s \in \mathcal{Q}$, then the behavior B is said to be quasi semi-modular.

Intuitively, quasi semi-modularity requires that once a state variable y^i becomes unstable and b is in its excitation, b should remain in the excitation until y^i changes. Our generalization is quite natural. If all the modules are deterministic, then quasi semi-modularity coincides with semi-modularity. That is, semi-modularity implies quasi semi-modularity.

Example 5.1 For a violation of quasi semi-modularity, refer to Figure 4.1(a) and (b). The delay change involved alters the excitation of the arbiter and violates quasi semi-modularity.

Example 5.2 Figure 5.1 shows a network N consisting of an arbiter, a $\overline{C}$-element, and two delays. A $\overline{C}$-element functions like a Muller C-element with inverted outputs; the two outputs are replicas of each other, and equal to the complement of the internal state.

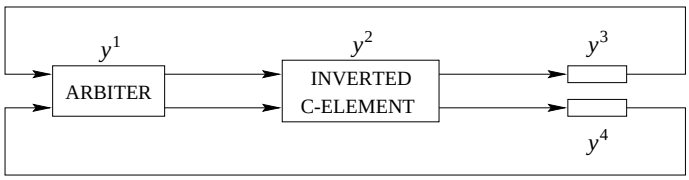

Figure 5.1: Network N.

Figure 5.2 shows a quasi semi-modular behavior of N starting from state $y^1y^2y^3y^4 = 0\mathbf{1}00$. Since the inputs to the arbiter are 00, and the arbiter is in state 0, it is stable and produces output 00. Since the $\overline{C}$-element is in state 1 and has inputs 00, it is unstable; it produces output 00, making the two delays stable. Hence, in the initial state, only the $\overline{C}$-element is unstable and the next state is 00**00**. The rest of the behavior can be computed similarly. □

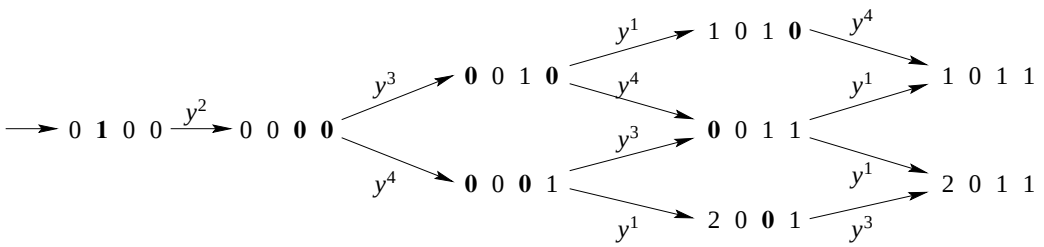

Figure 5.2: A quasi semi-modular behavior of N.

This example illustrates that the set inclusion in Definition 5.1 can be *proper*. For example, in state **0**010, the arbiter can change to state 1 only; in a subsequent state **0**011, resulting from a change on a delay, the arbiter can change to either 1 or 2.

Example 5.2 also shows that quasi semi-modularity is less restrictive than semi-modularity. According to Muller, semi-modular circuits form a proper subclass of speed-independent circuits. By his definition, the behavior shown in Figure 5.2 is not speed-independent with respect to the state 0**1**00, since there are two terminal classes; however, we see that it is quasi semi-modular. The existence of two terminal classes is attributable to the non-deterministic behavior of the arbiter, rather than to the race between the two delays y^3 and y^4. In that sense, the circuit should be considered speed-independent.

Note that there are non-deterministic speed-independent circuits exhibiting quasi semi-modular behaviors. An example can be found in Figure 1 of the paper by Yakovlev et al. [17].

Finally, we comment on quasi semi-modularity and lattice theory. Muller [11] shows that the partially ordered set of so-called *cumulative states* of a semi-modular behavior is a semi-modular lattice with a zero. A cumulative state is an n-tuple of integers recording the number of changes on the n *binary* state variables. We were unable to establish a similar correspondence between quasi semi-modularity and some appropriate lattice property.

5.2 Delay-dense networks

An asynchronous circuit consists of a set of components interconnected by wires, where each component can be modelled by a module. To model wire delays, we treat wires as delay modules. Intuitively, an asynchronous circuit is delay-insensitive if the correctness of its operation is independent of the delays in its components and in the wires connecting the components. We assume that the delays are finite, but otherwise arbitrary. As both component and wire delays are to be taken into account, any asynchronous circuit can be modelled by a *delay-dense* network, defined as follows:

Definition 5.2 A network N, as in Definition 3.1, is *delay-dense* if, whenever M^i feeds M^j in N, then either M^i or M^j is a delay module.

For example, the networks of Examples 3.1 and 3.2 are not delay-dense. They become delay-dense if we insert a delay in each of the network connections.

In defining networks, we do not rule out self-loops around modules. However, if a network is delay-dense, the only case where self-loops can exist is when the network consists of a single delay module.

5.3 Quasi semi-modularity and delay-density

We prove that quasi semi-modularity of behaviors of delay-dense networks is preserved under delay extensions. This result is essential in the proof of the main theorem of the paper, which is presented in the next section.

Theorem 5.1 *Let N be a delay-dense network, and let $\hat{N} \in \mathcal{D}(N)$. Let $B = \langle q, \mathcal{Q}, \mathcal{R} \rangle$ be a behavior of N, and let $\hat{B} = \langle \hat{q}, \hat{\mathcal{Q}}, \hat{\mathcal{R}} \rangle$ be the behavior of $\hat{N}$ which is initial-state compatible with B. If B is quasi semi-modular, then so is $\hat{B}$.*

Proof: We first prove a lemma concerning delay successors.

Lemma 5.1.1 *Let N be a network, and let $B = \langle q, \mathcal{Q}, \mathcal{R} \rangle$ be a quasi semi-modular behavior of N. Let $\hat{N}$ be a delay successor of N, where the extra delay M^d is inserted in a connection between two distinct modules. Without loss of generality, assume that the connection is from M^1 to M^2. If M^2 is a delay, then $\hat{B} = \langle \hat{q}, \hat{\mathcal{Q}}, \hat{\mathcal{R}} \rangle$, the behavior of $\hat{N}$ which is initial-state compatible with B, is also quasi semi-modular.*

Proof: Note that in this lemma, we do not require the network N to be delay-dense. For a proof of the lemma, see Appendix A.2.□

We now prove the theorem by induction on the number of inserted delays. The base case, where $\hat{N} = N$, is trivial. Now assume that $B = \langle q, \mathcal{Q}, \mathcal{R} \rangle$ is quasi semi-modular. We insert a delay M^d into a connection e of N. If e connects a module to itself, then N consists of a single delay module; the result holds trivially in that case. Thus we assume that M^d is inserted between two distinct modules M^i and M^j. Denote the resulting delay successor by $\hat{N}$. Thus, (M^i, M^d), $(M^d, M^j) \in \hat{\mathcal{F}}$. Let $\hat{B} = \langle \hat{q}, \hat{\mathcal{Q}}, \hat{\mathcal{R}} \rangle$ be the behavior of $\hat{N}$ which is initial-state compatible with B. We shall prove that $\hat{B}$ is quasi semi-modular.

If M^j is a delay, then the result follows from Theorem 5.1.1; otherwise, since N is delay-dense, M^i must be a delay. Let $(M^h, M^i) \in \mathcal{F}$. Clearly, M^h is unique. Imagine that in the network N, we change the index of M^i to d, and call the new network $\tilde{N}$. Obviously, B is a quasi semi-modular behavior of $\tilde{N}$ as well. Now $\hat{N}$ can be viewed as being obtained from $\tilde{N}$ by inserting M^i in the connection between M^h and M^d, in which case the new delay feeds the delay M^d. So the result we wish to prove again follows from Theorem 5.1.1. The proof of Theorem 5.1 is completed by induction on the number of inserted delays. □

It is worth noting that Theorem 5.1 does not necessarily hold for networks which are not delay-dense. For example, consider the network consisting of a two-output inverter y^1 and an AND-gate y^2, as shown in Figure 5.3(a) (for now, assume that the box labelled y^3 is a delay-free connection). Let the initial state be **00**. The corresponding behavior is quasi semi-modular, as shown in Figure 5.3(b). With a delay y^3 inserted, as shown in Figure 5.3(a), the corresponding behavior (Figure 5.3(c)) is no longer quasi semi-modular; more specifically, the states **100** and **001** are not quasi semi-modular.

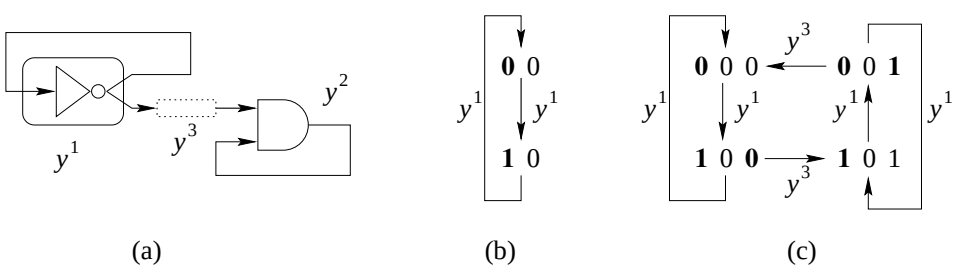

Figure 5.3: Theorem 5.1 fails to hold for networks which are not delay-dense.

6 Strong delay-insensitivity and quasi semi-modularity

In this section, the proof of the main theorem of the paper is presented. We first prove that strong delay-insensitivity implies quasi semi-modularity; here, delay-density of networks is not required. We then prove that, for delay-dense networks, quasi semi-modularity implies strong delay-insensitivity. As any asynchronous circuit can be modelled by a delay-dense network, we conclude that an asynchronous circuit is strongly delay-insensitive if and only if its behavior is quasi semi-modular.

Theorem 6.1 *If a network is strongly delay-insensitive with respect to a state, then its behavior originating from that state is quasi semi-modular.*

Proof: Let N be a network, which is strongly delay-insensitive with respect to a state q. Let $B = \langle q, \mathcal{Q}, \mathcal{R} \rangle$ be the corresponding behavior. Let $\hat{N} \in \mathcal{D}(N)$ be obtained from N by inserting one delay in *each* connection of N. Let $\hat{B} = \langle \hat{q}, \hat{\mathcal{Q}}, \hat{\mathcal{R}} \rangle$ be the behavior of $\hat{N}$ which is initial-state compatible with B. By Definition 4.6, $\hat{B}$ is safe with respect to B. Let us first prove a lemma for later use.

Lemma 6.1.1 *Let $s \in \mathcal{Q}$. There exists $\hat{s} \in \hat{\mathcal{Q}}$ such that $\hat{s}$ is the stable-delay extension of s.*

Proof: We prove the result by induction. The base case, where $s = q$, is trivial. Let $s \in \mathcal{Q}$ and $t \in sy^i$, for some $y^i \in \mathcal{Y}$. Assume that there exists $\hat{s} \in \hat{\mathcal{Q}}$ such that $\hat{s}$ is the stable-delay extension of s. By Proposition 4.1, $\hat{S}_i = S_i$. Thus, there exists $\hat{t} \in \hat{\mathcal{Q}}$ such that $\hat{t} \in \hat{s}y^i$ and $\hat{t} \downarrow \mathcal{Y} = t$. By Proposition 4.2, there exists $w \in (\hat{\mathcal{Y}} - \mathcal{Y})^*$ such that $\hat{t}' \in \hat{t}w$, where $\hat{t}' \in \hat{\mathcal{Q}}$ is the stable-delay extension of t. Thus, the lemma is proven. □

Now suppose that B is not quasi semi-modular. Then there exists $(r, s) \in \mathcal{R}$ with $\tau(r, s) = y^i$ such that $R_j \not\subseteq S_j$ for some $j \neq i$ with $y^j \in \mathcal{U}(r)$. Let $b \in R_j$ such that $b \notin S_j$.

By Lemma 6.1.1, there exists $\hat{r} \in \hat{\mathcal{Q}}$ such that $\hat{r}$ is the stable-delay extension of r. By Proposition 4.1, $R_i = \hat{R}_i$ and $R_j = \hat{R}_j$. Thus, there exists $\hat{s} \in \hat{\mathcal{Q}}$ such that $\hat{s} \in \hat{r}y^i$ and $\hat{s} \downarrow \mathcal{Y} = s$. By our construction of the network $\hat{N}$, $(y^i, y^j) \notin \hat{\mathcal{F}}$. By Proposition 3.1, we have $\hat{R}_j = \hat{S}_j$. Note that $y^j \in \mathcal{U}(r) = \mathcal{U}(\hat{r})$. It follows that $y^j \in \mathcal{U}(\hat{s})$. Also, $b \in R_j = \hat{R}_j = \hat{S}_j$. Thus, there exists $\hat{t} \in \hat{\mathcal{Q}}$ such that $\hat{t} \in \hat{s}y^j$ and $\hat{t}_j = b$. Recall that $\hat{s}$ is an extension of s and $\hat{B}$ is safe with respect to B. Therefore, there exists $t \in \mathcal{Q}$ such that $t \in sy^j$ and $t = \hat{t} \downarrow \mathcal{Y}$. Thus, $t_j = b$, which implies that $b \in S_j$—a contradiction. Hence, B must be quasi semi-modular. □

Theorem 6.2 *If a behavior of a delay-dense network is quasi semi-modular, then the network is strongly delay-insensitive with respect to the initial state of that behavior.*

Proof: Let N be a delay-dense network. Let $B = \langle q, \mathcal{Q}, \mathcal{R} \rangle$ be a quasi semi-modular behavior of N. Let $\hat{N} \in \mathcal{D}(N)$, and let $\hat{B}$ be the behavior of $\hat{N}$ which is initial-state compatible with B. We shall prove that $\hat{B}$ is safe with respect to B.

Let $\hat{s} \in \hat{\mathcal{Q}}$ be an extension of $s \in \mathcal{Q}$. Let $\hat{t} \in \hat{\mathcal{Q}}$ and $\hat{w} \in \hat{\mathcal{Y}}^*$ be such that $\hat{t} \in \hat{s}\hat{w}$ and $\hat{w} \downarrow \mathcal{Y} = y^i$ for some $y^i \in \mathcal{Y}$. Let $\hat{t}_i = b$. Clearly, all we need to show is that $b \in S_i$. Let us write $\hat{w}$ as $\hat{w} = \hat{u}y^i\hat{u}'$, where $\hat{u}, \hat{u}' \in (\hat{\mathcal{Y}} - \mathcal{Y})^*$. Denote the state reached after $\hat{u}$ by $\hat{s}'$,

$\hat{s}' \in \hat{s}\hat{u}$. Clearly, $y^i \in \mathcal{U}(\hat{s}')$ and $b \in \hat{S}_i'$. Also, $\hat{s}'$ is an extension of s. By Proposition 4.2, there exists $\hat{v} \in (\hat{\mathcal{Y}} - \mathcal{Y})^*$ such that $\hat{s}'' \in \hat{s}'\hat{v}$, where $\hat{s}''$ is the stable-delay extension of s.

By Theorem 5.1, $\hat{B}$ is quasi semi-modular. Since y^i, an original module, does not change in $\hat{v}$, we must have $\hat{S}_i' \subseteq \hat{S}_i''$, which implies that $b \in \hat{S}_i''$. On the other hand, by Proposition 4.1, we have $S_i = \hat{S}_i''$. Hence, $b \in S_i$. By definition, $\hat{B}$ is safe with respect to B. Since $\hat{N}$ was chosen arbitrarily, by Definition 4.6, N is strongly delay-insensitive with respect to q. □

Table 6.1: Functions of $M^{\updownarrow}$.

	x		
y	0	1	$\lambda(y)$
0	$\{2\}$	$\{1,2\}$	0
1	$\{0\}$	$\{2\}$	0
2	$\{2\}$	$\{1\}$	1
	$\delta(x,y)$		

Note that Theorem 6.2 does not necessarily hold for networks which are not delay-dense. For example, consider network N, shown in Figure 6.2 (ignore y^2 for now), consisting of a single module $M^{\updownarrow}$, whose excitation and output function are given in Table 6.1.

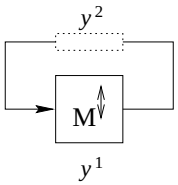

Figure 6.2: Network N.

Let the initial state be **0**. The corresponding behavior B is quasi semi-modular, as shown in Figure 6.3(a). Now let us insert a delay y^2, as shown in Figure 6.2. The corresponding behavior $\hat{B}$ of the resulting delay extension $\hat{N}$ is given in Figure 6.3(b). Note that in state **11** of $\hat{B}$, $M^{\updownarrow}$ may change to 2; but in state **1** of B, $M^{\updownarrow}$ can only change to 0. Therefore, $\hat{B}$ is not safe with respect to B, and consequently, N is not strongly delay-insensitive with respect to state **0**.

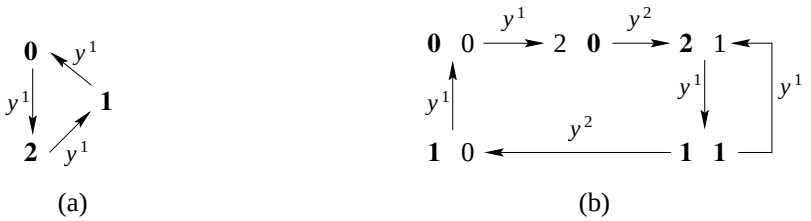

Figure 6.3: Theorem 6.2 fails to hold for networks which are not delay-dense.

Theorem 6.1 holds for all networks. Combining Theorem 6.1 and 6.2, we have the main theorem of the paper:

Theorem 6.3 *A delay-dense network is strongly delay-insensitive with respect to a state if and only if its behavior originating from that state is quasi semi-modular.*

7 Conclusion

In this section, we first show the relationships among various classes of autonomous asynchronous networks. We then discuss how the main result of the paper may be applied to test whether a given network is strongly delay-insensitive. Finally, we mention briefly how strong delay-insensitivity is related to the well-accepted notion of delay-insensitivity due to Udding.

7.1 A classification of asynchronous networks

Figure 7.1 shows the relationship among strong delay-insensitivity, quasi semi-modularity, and delay-density. The universal set is the set of *initialized* autonomous networks, i.e., autonomous networks with designated initial states. For regions corresponding to the empty set, we show the relevant theorem. For other regions, we give the figure which shows a network in that class.

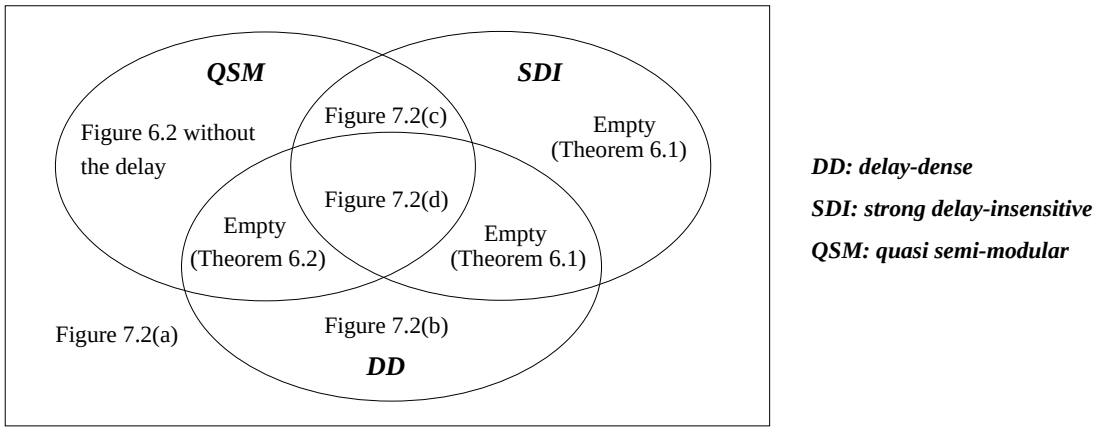

Figure 7.1: A first classification of asynchronous networks.

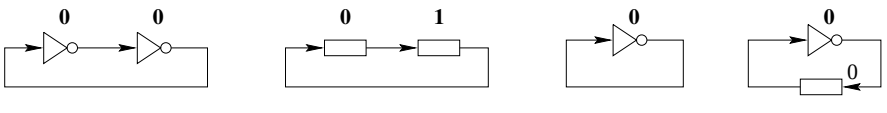

Figure 7.2: Network examples used in the first classification.

Figure 7.3 shows a classification of *deterministic* initialized networks. Speed-independence is as defined by Muller [11]. For general networks, we have discussed the relationship among quasi semi-modularity, semi-modularity, and Muller's speed-independence in Section 5.1.

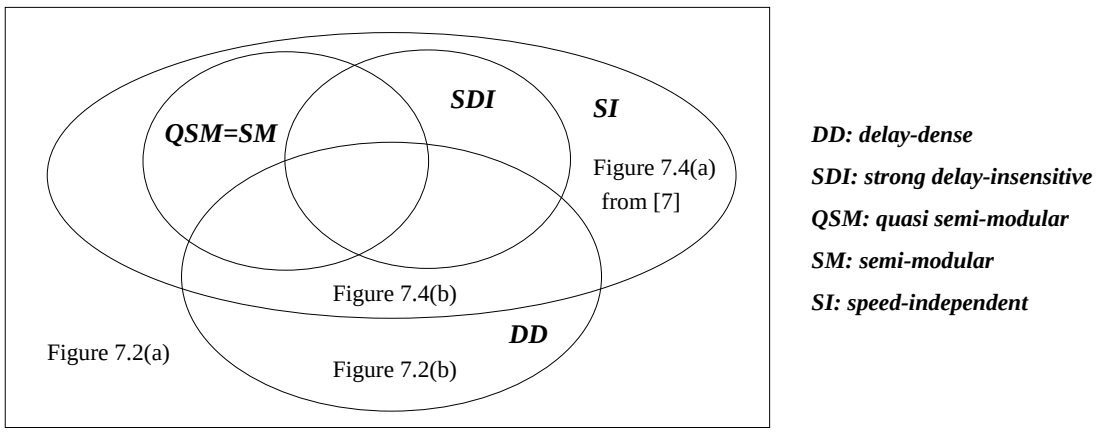

Figure 7.3: A classification of deterministic asynchronous networks.

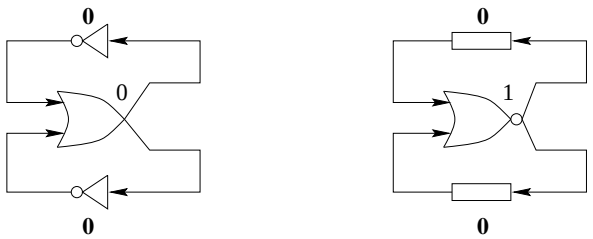

Figure 7.4: Network examples used in the second classification.

7.2 Testing strong delay-insensitivity of networks

Suppose that we are given a network N and an initial state q, and we are to determine whether N is strongly delay-insensitive with respect to q. We cannot apply Definition 4.6 directly, since it would involve an infinite test. If N happens to be delay-dense, then by Theorem 6.3, we only need to check whether the behavior of N originating from q is quasi semi-modular; this problem is decidable. The test for quasi semi-modularity is linear in the size of the behavior graph, but potentially exponential in the number of state variables. Here, we assume that the size of the domain of each state variable is constant.

We do not have a complete test for all networks. However, we do have one positive test and one negative test, both involving a test for quasi semi-modularity only. The negative test uses Theorem 6.1: If the behavior of N originating from q is not quasi semi-modular, then N is not strongly delay-insensitive with respect to q. The positive test is concerned with *delay-completion*: The delay-completion of a network N is a delay extension of N obtained by inserting one delay module in each connection of N.

Theorem 7.1 *Let N' be the delay-completion of a network N. Let $B = \langle q, \mathcal{Q}, \mathcal{R} \rangle$ be a behavior of N, and let $B' = \langle q', \mathcal{Q}', \mathcal{R}' \rangle$ be the behavior of N' which is initial-state compatible with B. If B' is quasi semi-modular, then N is delay-insensitive with respect to q.*

Proof: See the proof of Theorem 5.5 in [18]. □

Thus, given N and q, we first construct the delay-completion of N. We then check whether the behavior of the delay-completion is quasi semi-modular. If the answer is yes, then by Theorem 7.1, we conclude that N is strongly delay-insensitive with respect to q. Unfortunately, there are networks for which neither the positive test nor the negative test gives a definitive answer. An example of such a network was given in Figure 5.3(a).

7.3 Strong delay-insensitivity and Udding's delay-insensitivity

Recall that the behavior of a network can be captured by a language (Section 3). If the network is delay-dense, then for each component, we can obtain a description of its behavior by projecting that language onto only the delay modules connected to the component. This sublanguage is a trace set of the component. H. Zhang [18, 19] has shown that if the network is strongly delay-insensitive, then the trace set of each component satisfies the JTU-rules; but the converse is not true. We see this as a justification for the use of the term strong delay-insensitivity.

References

[1] J. A. Brzozowski, "Delay-Insensitivity and Ternary Simulation," *Theoretical Computer Science*, to appear.

[2] J. A. Brzozowski and C-J. Seger, *Asynchronous Circuits,* Springer-Verlag, New York, NY, 1995.

[3] D. L. Dill, *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*, The MIT Press, Cambridge, Massachusetts, 1988.

[4] J. C. Ebergen, *Translating Programs into Delay-Insensitive Circuits,* Ph.D. Thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands, October 1987. Also, CWI Tract 56, Center for Mathematics and Computer Science, Amsterdam, The Netherlands, 1989.

[5] A. J. Martin, "The Limitations to Delay-Insensitivity in Asynchronous Circuits," *Proceedings of the 6th MIT Conference on Advanced Research in VLSI,* W. J. Dally, ed., MIT Press, pp. 263–278, 1990.

[6] C. Mead and L. Conway, *Introduction to VLSI Systems,* Addison-Wesley, Reading, MA, 1980.

[7] R. E. Miller, "Speed-Independent Switching Circuit Theory," Chapter 10 in *Switching Theory, Volume II: Sequential Circuits and Machines,* John Wiley & Sons, 1965.

[8] R. Milner, *Communication and Concurrency,* Prentice Hall, 1989.

[9] C. E. Molnar, T. P. Fang, and F. U. Rosenberger, "Synthesis of Delay-Insensitive Modules," *Proceedings of the 1985 Chapel Hill Conference on VLSI,* H. Fuchs, ed., Computer Science Press, Rockville, Maryland, pp. 67–86, 1985.

[10] E. F. Moore, "Gedanken Experiments on Sequential Machines," in C. E. Shannon and J. McCarthy, eds., *Automata Studies, Annals of Mathematics Study 34,* Princeton University Press, Princeton NJ, pp. 129–153, 1956.

[11] D. E. Muller, *A Theory of Asynchronous Circuits,* Technical Report 66, Digital Computer Lab., University of Illinois, Urbana-Champaign, Illinois, USA, 1955.

[12] D. E. Muller and W. C. Bartky, "A Theory of Asynchronous Circuits," in *Annals of Computing Laboratory of Harvard University*, pp. 204–243, 1959.

[13] C. L. Seitz, "System Timing," Chapter 7 in [6], pp. 218–262.

[14] N. Shintel and M. Yoeli, "Synthesis of Modular Networks from Petri-Net Specifications," Technical Report #743, Israel Institute of Technology, Haifa 32000 Israel, 1992.

[15] J. T. Udding, *Classification and Composition of Delay-Insensitive Circuits,* Doctoral Dissertation, Eindhoven University of Technology, Eindhoven, The Netherlands, 1984.

[16] T. Verhoeff, *A Theory of Delay-Insensitive Systems*, PhD Thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands, May 1994.

[17] A. Yakovlev, L. Lavagno, and A. Sangiovanni-Vincentelli, "A Unified Signal Transition Graph Model for Asynchronous Control Circuit Synthesis," *Formal Methods in System Design,* 9, pp. 139–188, 1996.

[18] H. Zhang, *Delay-Insensitive Networks,* Master Thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 1997.

[19] H. Zhang and J. A. Brzozowski, "Strong Delay-Insensitivity and the JTU-Rules," Technical Report, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 1998.

8 Appendix

A.1

Proof of Proposition 4.3: We prove livelock-freedom first. Suppose that there exist $\hat{s} \in \hat{\mathcal{Q}}$ and $\hat{w} \in (\hat{\mathcal{Y}} - \mathcal{Y})^+$ such that $\hat{s} \in \hat{s}\hat{w}$. Let y^i be an inserted delay which appears in $\hat{w}$. Clearly, y^i must appear at least twice in $\hat{w}$. Thus, the (unique) state variable y^h that feeds y^i must have changed and appears in $\hat{w}$. It follows that y^h is an inserted delay. Applying the

same argument repeatedly (this time starting with y^h) leads to a contradiction, since there are no cycles of inserted delays in $\hat{N}$.

To prove completeness of $\hat{B}$, let $\hat{s} \in \hat{\mathcal{Q}}$ be an extension of $s \in \mathcal{Q}$, and let $t \in sy^i$. By Proposition 4.2, there exists $\hat{w} \in (\hat{\mathcal{Y}} - \mathcal{Y})^*$ such that $\hat{s}' \in \hat{s}\hat{w}$, where $\hat{s}'$ is the stable-delay extension of s. By Proposition 4.1, $S_i = \hat{S}_i'$. Therefore, there exists $\hat{t} \in \hat{s}'y^i$ such that $\hat{t}$ is an extension of t with respect to N. Thus, we have $\hat{t} \in \hat{s}'y^i \subseteq \hat{s}\hat{w}y^i$, $\hat{w}y^i \downarrow \mathcal{Y} = y^i$, and $\hat{t}$ is an extension of t with respect to N. By Definition 4.5, $\hat{B}$ is complete with respect to B. □

A.2

Proof of Lemma 5.1.1: By the construction of network $\hat{N}$, we have $(y^1, y^2) \in \mathcal{F}$ and (y^1, y^d), $(y^d, y^2) \in \hat{\mathcal{F}}$. The result follows from a series of four lemmata. In the following, we assume that $B = \langle q, \mathcal{Q}, \mathcal{R} \rangle$ is quasi semi-modular. Also, let $\hat{s} \in \hat{\mathcal{Q}}$ be any state of $\hat{N}$. The state $\hat{s}$ is said to be *normal* if $\hat{s} \downarrow \mathcal{Y} \in \mathcal{Q}$; in that case, we let $s = \hat{s} \downarrow \mathcal{Y}$.

Lemma 8.1 *Let $\hat{s}$ be normal. If $i \neq d$ and $i \neq 2$, then $\hat{S}_i = S_i$ and $\mathcal{U}(\hat{s}) \backslash \{y^d, y^2\} \subseteq \mathcal{U}(s)$.*

Proof: Clearly, M^i is an original module. One easily verifies that the input connections to M^i are identical in the two networks N and $\hat{N}$. Thus, the value of each argument of Δ_i in s is equal to the value of the corresponding argument of $\hat{\Delta}_i$ in $\hat{s}$. Hence, $\hat{S}_i = S_i$. The second result follows immediately. □

Lemma 8.2 *If $\hat{s}$ is normal, but not quasi semi-modular, then $y^d \in \mathcal{U}(\hat{s})$.*

Proof: Let $s = \hat{s} \downarrow \mathcal{Y} \in \mathcal{Q}$. Suppose that $\hat{s}$ is not quasi semi-modular. Then there exists $\hat{t} \in \hat{\mathcal{Q}}$ such that

1. $(\hat{s}, \hat{t}) \in \hat{\mathcal{R}}$ with $\tau(\hat{s}, \hat{t}) = y^i$;
2. there exists $j \neq i$ such that $y^j \in \mathcal{U}(\hat{s})$ and $\hat{S}_j \not\subseteq \hat{T}_j$.

By Proposition 3.1, $(y^i, y^j) \in \hat{\mathcal{F}}$. Also, $y^i \in \mathcal{U}(\hat{s})$, by definition. Assume that $y^d \notin \mathcal{U}(\hat{s})$. Then, $i \neq d$ and $j \neq d$. Since M^2 (a delay) has exactly one input, $i \neq d$, and (y^i, y^j), $(y^d, y^2) \in \hat{\mathcal{F}}$, we must have $j \neq 2$.

Since $y^d \notin \mathcal{U}(\hat{s})$, $\hat{s}$ is the stable-delay extension of s. Also, M^i and M^j are original. By Proposition 4.1, $\hat{S}_j = S_j$ and $\hat{S}_i = S_i$. It follows that there exists $t \in \mathcal{Q}$ such that $s\mathcal{R}t$, $\tau(s,t) = y^i$, and $t = \hat{t} \downarrow \mathcal{Y}$. Thus, $\hat{t}$ is normal. Recall that $j \neq d$ and $j \neq 2$. By Lemma 8.1, $\hat{T}_j = T_j$. On the other hand, $y^j \in \mathcal{U}(\hat{s}) = \mathcal{U}(s)$. Also, B is quasi semi-modular and $j \neq i$. Therefore, $S_j \subseteq T_j$. It follows that $\hat{S}_j = S_j \subseteq T_j = \hat{T}_j$—a contradiction. Hence, if $\hat{s}$ is not quasi semi-modular, then $y^d \in \mathcal{U}(\hat{s})$. □

Lemma 8.3 *If $\hat{s}$ is normal, but not quasi semi-modular, then either $y^2 \in \mathcal{U}(\hat{s})$ or there exists $\hat{t} \in \hat{\mathcal{Q}}$ such that $(\hat{s}, \hat{t}) \in \hat{\mathcal{R}}$, $\tau(\hat{s}, \hat{t}) = y^1$, and $\hat{S}_d \not\subseteq \hat{T}_d$.*

Proof: Let $s = \hat{s} \downarrow \mathcal{Y} \in \mathcal{Q}$. Suppose that $\hat{s}$ is not quasi semi-modular. Let i, j, and $\hat{t}$ be as defined in the proof of Lemma 8.2. Again, we have $(y^i, y^j) \in \hat{\mathcal{F}}$ and $y^i \in \mathcal{U}(\hat{s})$. Assume that $y^2 \notin \mathcal{U}(\hat{s})$. Then, $i \neq 2$ and $j \neq 2$. Since M^d (a delay) has exactly one output, $j \neq 2$, and (y^i, y^j), $(y^d, y^2) \in \hat{\mathcal{F}}$, we must have $i \neq d$.

Suppose that $j \neq d$. By Lemma 8.1, $\hat{S}_j = S_j$. Similarly, $\hat{S}_i = S_i$. It follows that there exists $t \in \mathcal{Q}$ such that $s\mathcal{R}t$, $\tau(s,t) = y^i$, and $t = \hat{t} \downarrow \mathcal{Y}$. Thus, $\hat{t}$ is normal. By Lemma 8.1 again, $\hat{T}_j = T_j$. On the other hand, by Lemma 8.1, $y^j \in \mathcal{U}(\hat{s}) - \{y^d, y^2\} \subseteq \mathcal{U}(s)$. Also, B is quasi semi-modular and $j \neq i$. Therefore, $S_j \subseteq T_j$. It follows that $\hat{S}_j = S_j \subseteq T_j = \hat{T}_j$—a contradiction. Hence, $j = d$. Since $(y^i, y^j) \in \hat{\mathcal{F}}$, we must have $i = 1$. Thus, $(\hat{s}, \hat{t}) \in \hat{\mathcal{R}}$, $\tau(\hat{s}, \hat{t}) = y^1$, and $\hat{S}_d \not\subseteq \hat{T}_d$, as in the lemma. □

Lemma 8.4 *For all $\hat{s} \in \hat{\mathcal{Q}}$, $\hat{s}$ is quasi semi-modular and normal.*

Proof: We proceed by induction. For the base case, let $\hat{s} = \hat{q}$. Clearly, $\hat{s}$ is normal and $y^d \notin \mathcal{U}(\hat{s})$. By Lemma 8.2, $\hat{s}$ is quasi semi-modular. Now, let $(\hat{r}, \hat{s}) \in \hat{\mathcal{R}}$ with $\tau(\hat{r}, \hat{s}) = y^i$. Assume that $\hat{r}$ is quasi semi-modular and normal. We shall prove that $\hat{s}$ is quasi semi-modular and normal.

Let $r = \hat{r} \downarrow \mathcal{Y} \in \mathcal{Q}$. If $y^d \notin \mathcal{U}(\hat{r})$, then $i \neq d$ and $\hat{R}_i = R_i$. It follows that $\hat{s} \downarrow \mathcal{Y} \in \mathcal{Q}$. Now assume that $y^d \in \mathcal{U}(\hat{r})$. If $y^2 \in \mathcal{U}(\hat{r})$, then by changing y^d in state $\hat{r}$, we would have a violation of quasi semi-modularity, contradicting the quasi semi-modularity of $\hat{r}$. Therefore, $y^2 \notin \mathcal{U}(\hat{r})$ and $i \neq 2$. If $i = d$, then $\hat{s} \downarrow \mathcal{Y} = \hat{r} \downarrow \mathcal{Y} \in \mathcal{Q}$; otherwise, by Lemma 8.1, we have $\hat{R}_i = R_i$, which implies that $\hat{s} \downarrow \mathcal{Y} \in \mathcal{Q}$. Hence, $\hat{s}$ is normal.

Suppose that $\hat{s}$ is not quasi semi-modular. Let the output vertex of M^1 which is connected to M^d be z_k^1. Clearly, z_k^1 is connected to M^2 in N. By Lemma 8.2, $y^d \in \mathcal{U}(\hat{s})$. It follows that $i \neq d$. By Lemma 8.3, there are two cases to consider.

- $y^2 \in \mathcal{U}(\hat{s})$:

  Note that y^2 is a delay. Since $i \neq d$, $y^2 \in \mathcal{U}(\hat{r})$. By an argument similar to the one given above, we have $y^d \notin \mathcal{U}(\hat{r})$. Thus, $i = 1$ and $y^1 \in \mathcal{U}(\hat{r})$. On the other hand, since $y^d \notin \mathcal{U}(\hat{r})$, $\hat{r}$ is the stable-delay extension of r. It follows that $y^1, y^2 \in \mathcal{U}(\hat{r}) = \mathcal{U}(r)$ and there exists $s \in \mathcal{Q}$ such that $r\mathcal{R}s$, $\tau(r,s) = y^1$, and $s = \hat{s} \downarrow \mathcal{Y}$. We have $R_2 = \lambda_k^1(r_1) = \lambda_k^1(\hat{r}_1) = \hat{R}_d = \hat{r}_d$ and $S_2 = \lambda_k^1(s_1) = \lambda_k^1(\hat{s}_1) = \hat{S}_d$. Since $y^d \in \mathcal{U}(\hat{s})$, $\hat{s}_d \neq \hat{S}_d$. But $\hat{r}_d = \hat{s}_d$. Therefore, $R_2 \neq S_2$, which violates quasi semi-modularity of B—a contradiction.

- There exists $\hat{t} \in \hat{\mathcal{Q}}$ such that $(\hat{s}, \hat{t}) \in \hat{\mathcal{R}}$, $\tau(\hat{s}, \hat{t}) = y^1$, and $\hat{S}_d \not\subseteq \hat{T}_d$:

  Since $y^2 \notin \mathcal{U}(\hat{s})$, $\hat{s}_2 = \hat{s}_d$. Also, $\hat{s}_d \neq \lambda_k^1(\hat{s}_1)$, since $y^d \in \mathcal{U}(\hat{s})$. Let $s = \hat{s} \downarrow \mathcal{Y} \in \mathcal{Q}$. Then, $s_2 \neq \lambda_k^1(s_1)$, which implies that $y^2 \in \mathcal{U}(s)$. On the other hand, by Lemma 8.1, we have $\hat{S}_1 = S_1$. Therefore, there exists $t \in \mathcal{Q}$ such that $s\mathcal{R}t$, $\tau(s,t) = y^1$, and $t = \hat{t} \downarrow \mathcal{Y}$. We have $S_2 = \lambda_k^1(s_1) = \lambda_k^1(\hat{s}_1) = \hat{S}_d$, and similarly, $T_2 = \hat{T}_d$. Since $\hat{S}_d \not\subseteq \hat{T}_d$, i.e., $\hat{S}_d \neq \hat{T}_d$, we have $S_2 \neq T_2$, which violates quasi semi-modularity of B—a contradiction again.

Hence, $\hat{s}$ is quasi semi-modular. The lemma is proven by induction. □

By Lemma 8.4, $\hat{s}$ is quasi semi-modular for all $\hat{s} \in \hat{\mathcal{Q}}$. Hence, $\hat{B}$ is quasi semi-modular.□