



The United Nations
University

UNU-IIST

International Institute for
Software Technology

Timing models for asynchronous circuits

Kamel Boumaza, J. W. Sanders and Wang Xu

May 2008

UNU-IIST and UNU-IIST Reports

UNU-IIST (United Nations University International Institute for Software Technology) is a Research and Training Centre of the United Nations University (UNU). It is based in Macao, and was founded in 1991. It started operations in July 1992. UNU-IIST is jointly funded by the government of Macao and the governments of the People's Republic of China and Portugal through a contribution to the UNU Endowment Fund. As well as providing two-thirds of the endowment fund, the Macao authorities also supply UNU-IIST with its office premises and furniture and subsidise fellow accommodation.

The mission of UNU-IIST is to assist developing countries in the application and development of software technology.

UNU-IIST contributes through its programmatic activities:

1. Advanced development projects, in which software techniques supported by tools are applied,
2. Research projects, in which new techniques for software development are investigated,
3. Curriculum development projects, in which courses of software technology for universities in developing countries are developed,
4. University development projects, which complement the curriculum development projects by aiming to strengthen all aspects of computer science teaching in universities in developing countries,
5. Schools and Courses, which typically teach advanced software development techniques,
6. Events, in which conferences and workshops are organised or supported by UNU-IIST, and
7. Dissemination, in which UNU-IIST regularly distributes to developing countries information on international progress of software technology.

Fellows, who are young scientists and engineers from developing countries, are invited to actively participate in all these projects. By doing the projects they are trained.

At present, the technical focus of UNU-IIST is on formal methods for software development. UNU-IIST is an internationally recognised center in the area of formal methods. However, no software technique is universally applicable. We are prepared to choose complementary techniques for our projects, if necessary.

UNU-IIST produces a report series. Reports are either Research , Technical , Compendia or Administrative . They are records of UNU-IIST activities and research and development achievements. Many of the reports are also published in conference proceedings and journals.

Please write to UNU-IIST at P.O. Box 3058, Macao or visit UNU-IIST's home page: <http://www.iist.unu.edu>, if you would like to know more about UNU-IIST and its report series.

G. M. Reed, Director



The United Nations
University

UNU-IIST

International Institute for
Software Technology

P.O. Box 3058
Macao

Timing models for asynchronous circuits

Kamel Boumaza, J. W. Sanders and Wang Xu

Abstract

The purpose of this paper is to explore the models and forms of reasoning appropriate for analysing the temporal behaviour of circuits. That leads from various Boolean models appropriate for combinational and synchronous circuits, to the event model for asynchronous circuits. Circuit behaviour is captured at each level of abstraction using model-specific operators, whose laws are used to perform algebraic (rather than semantic) reasoning about transient behaviour. The results have a particularly satisfying form: for example the hazards of a conditional are calculated algebraically to be a conditional of hazards. A method is developed for simulating an asynchronous combinational circuit, that starts with an untimed, logic-level, specification, passes to a timed refinement and then moves to a simple timed automaton for simulation in UPPAAL. Each stage in the method is supported by a projection relating the models at the various levels, so that correctness can be confirmed.

Kamel Boumaza has completed a nine-month UNU-IIST Fellowship and is now pursuing a PhD at the Universit Badji Mokhtar de Annaba, Algeria.

Xu Wang is currently Assistant Research Fellow at UNU-IIST. His main research topic is process algebra and state-space reduction for model checking.

Jeff Sanders is Principal Research Fellow at UNU-IIST, with interests lying largely in Formal Methods.

Contents

1	Introduction	1
1.1	Contents	1
2	Circuits	2
2.1	Examples	2
2.2	Definition	4
2.3	Induction principle	4
3	Signals	5
3.1	Boolean model	5
3.2	Binary model	6
3.3	Ternary model	6
3.4	Signal model	7
3.5	Circuits	9
4	Timed behaviour	9
4.1	Signal delay	9
4.2	Gate delay	10
4.3	Feedback	10
4.4	Signal change	11
4.5	Between models	12
5	Hazard freedom	14
5.1	Constant input	14
5.2	Single input change	15
5.3	Transients	16
5.4	Restricted case of a conditional gate	17
5.5	Conditional gate	19
5.6	Iterative multiplexor	21
5.7	Recursive multiplexor	22
6	Events	23
6.1	Introducing events	24
6.2	The event model	25
6.3	Timed automata	28
6.4	Multitrace semantics	30
6.5	An <i>and</i> gate	34
7	Conclusion	38
7.1	Further work	38

1 Introduction

The purpose of this paper is to consider a hierarchy of models for the temporal behaviour of circuits in an environment that is either benign (for example in a clocked environment so that input is assured to arrive conveniently, in which case Boolean-based models suffice) or not (in an asynchronous environment, in which case an event-based model is relevant). Of interest are the relationships between the models and the choice of model-specific laws that are sufficient to perform algebraic reasoning about behaviour captured at the level of abstraction represented by the model (and whose extension would ultimately characterise the model algebraically). Most of our concern here lies with combinational circuits, for which a method is developed that starts from an untimed, Boolean, specification of a combinational circuit, refines it to a temporal, signal-based, specification, translates that to a certain kind of timed automaton and finally converts that to UPPAAL for simulation.

Use of the laws is demonstrated on results like hazard freedom of basic circuits and choke-refinement between variant circuits modelled as circuit automata.

1.1 Contents

Circuits (including those with feedback) are defined recursively in Section 2 and the definition accompanied by a (strong) induction principle. The definition distinguishes input signals (or actions) from output and internal signals (or co-actions).

Section 3 introduces a hierarchy of models: the Boolean, in which a signal is represented (if possible) by a single stable Boolean value; the binary, in which it is represented by a value before and after some change; the ternary, in which an intermediate transient Boolean value is captured; and the signal model which captures a Boolean value at each (integral) time point. The models are shown to form a hierarchy under a sequence of projections taking each (except the first) to its predecessor in the hierarchy. In each model, a circuit is captured by a collection of named observables (signals) at the relevant level of abstraction, bound by some constraint, captured as a predicate whose free variables are the observables.

In Section 4 the previous logical operators (used to define combinational circuits) are extended by the definition of temporal operators for delay and signal change (or differentiation). The assumption is made that the circuit's environment is benign: it accepts output whenever the circuit supplies it but delivers input only when convenient for the circuit (for example not whilst an output is being evaluated). Of interest here is the derived operators—like that for a transient—and their laws that are convenient to give compact incisive proofs of hazardous circuit behaviour. A collection of such results are given in Section 5. A typical example is the identification of the hazardous behaviour of a conditional gate, here captured as a conditional of its input behaviours, conditioned on the sign of the inverter delay; and its extension to both iterative and recursive multiplexors.

In Section 6 the assumption that the environment is benign is removed, by consideration of an event model here formalised by differentiation and compression of signals. The result is called a multitrace, and

the standard notion of ‘choke refinement’ is expressed between multitraces. Each circuit is expressed as a simple timed automaton, here called a ‘circuit automaton’, and given a multitrace semantics. A method is developed for simulating a combinational circuit in UPPAAL, starting from a signal specification, and converting it to a circuit automaton. The method is illustrated on a collection of inverters and *and* gates.

2 Circuits

2.1 Examples

This section contains examples to motivate the algebraic notation for circuits to be introduced in the next section.

Inputs to a circuit are labelled with distinct natural numbers, and written $in\ j$. Different occurrences of a gate within a circuit are distinguished similarly (thus: $gate\ i$). The output of a gate is identified with the gate’s name; if that value is used in more than one location (by forking its value) that fact is indicated by multiple occurrences of the gate’s name; in that way explicit fork gates are avoided and the notation abbreviated as a result. Since a circuit may have several outputs, each output is identified with a label (thus: $out\ k$).

For instance a *conditional* gate in disjunctive normal form

$$out0 = (in0 \wedge \neg in2) \vee (in1 \wedge in2)$$

can be expressed with the obvious functionality assigned to each gate

$$out0(or0(and0(in0, not0(in2)), and1(in1, in2))).$$

The circuit has a single output (from the *or* gate), $out0$, and three inputs, $in0$, $in1$ and $in2$. The fork on $in2$ is represented by the repeated occurrence of $in2$. That notation can be thought of as short-hand for a system of mutually recursive equations

$$\begin{aligned} out0 &= or0(a, b) \\ a &= and0(c, d) \\ b &= and1(e, f) \\ c &= in0 \\ d &= not0(f) \\ e &= in1 \\ f &= in2. \end{aligned}$$

A circuit with two outputs, one from each *and* gate, and three inputs, one shared as above,

$$\left(\begin{array}{l} out0 = in0 \wedge in1 \\ out1 = in2 \wedge in1 \end{array} \right)$$

is expressed by combining two single-output circuits in parallel

$$out0(and0(in0, in1)), out1(and1(in1, in2)).$$

Circuits at the gate level are subtly different from those at the (CMOS) transistor level; but it is required to have notation to cover both. In gate-level design a *join* (hard-wired *or*) is too dangerous to be used whilst because of the complementary switching nature of *n*type and *p*type transistors, that is typical practice at the transistor level. For example a CMOS inverter is represented using a *join* and two transistors

$$\begin{aligned} ptran(x, y) &= \neg x \Rightarrow y \\ ntran(x, y) &= x \Rightarrow y \\ join(a, b) &= a \wedge b \end{aligned}$$

using 1 to represent power input and 0 ground input

$$out0(join0(ptran0(1, in0), ntran0(0, in0))).$$

Then correctness follows in the standard manner:

$$\begin{aligned} &out0(join0(ptran0(1, in0), ntran0(0, in0))) \\ = & && \text{definitions of output and join} \\ out0 &= ptran0(1, in0) \wedge ntran0(0, in0) && \text{definitions of transistors} \\ = & && \text{calculus} \\ out0 &= (\neg in0 \Rightarrow 1) \wedge (in0 \Rightarrow 0) \\ = & && \text{calculus} \\ out0 &= \neg in0, \end{aligned}$$

as required for an inverter. Thus the devices appearing in a circuit depend on the design discipline being followed; with arbitrary delays in the circuit, the *join0* would be incorrect even at the CMOS level.

Feedback is expressed using repeated occurrence of gate output. For instance a circuit with output from one of its two *and* gates, and having two inputs but with output fed back to the first *and* gate, is expressed inadequately using logic

$$out0 = in0 \wedge (in1 \wedge out0)$$

(inadequate because nothing distinguishes the two occurrences of *out0*) and as a circuit

$$out0(and0(in0, and1(in1, and0)))$$

wherein it will be the semantics of gates which enables the two occurrences of *out0* to be distinguished: one a delayed instance of the other.

We choose to disqualify syntactically correct circuits that do not have inputs to each of their subcircuits. For example in this circuit

$$out0(and0(in0, not0(not0)))$$

the *and* gate has an (external) input but the inverter does not. So although such a subcircuit might be considered a clock (with period determined by the inverter's delay) we disqualify it because we are unable to determine its output value (absolutely) at any time.

2.2 Definition

Definition of circuit. For a given family of inputs *in* and devices *gate* (each of finite arity), a circuit is defined recursively.

$$c ::= in \mid out(c) \mid c, c \mid gate(\vec{c})$$

The base case of a circuit is formed by its inputs *in*, assumed to be uniquely identified by natural number indices *i*. If *c* and *c'* are circuits (perhaps with common subcircuits) so is their parallel composition *c, c'*. If \vec{c} is an *n*-vector of circuits each with one output and *gate_j* is a device with arity *n* then both *gate_j(\vec{c})* and *out_k(gate_j(\vec{c}))* are circuits, for unique natural-number identifiers *j, k*.

But since that syntax allows circuits like *not0(not0)*, we impose a healthiness condition that each circuit have at least one output and each of the circuit's subcircuits have a path from at least one (external) input. \square

That definition is syntactic, using only labels (with arity) for devices, and containing nothing to reflect semantics. Indeed it is convenient for the definition to be independent of the semantics so that behaviours at different levels of abstraction can later be considered without changing the notion of circuit. However care is later needed to ensure that when devices input and output data they do so with some delay to ensure well-founded recursion.

Examples of gates with arity 2 are *and*, *or*, *exor*, *nand*, *nor*, *ntran*, *ptran* and *join* gates; examples with arity 1 are inverter and buffer gates; examples with arity 0 are power and ground. Later (in Section 5.6) that notation will be used to describe multi-input *and* and *or* gates. Then each gate identifier will be subscripted with its arity. Thus a *k*-input *and* gate will be written *and_{0_k}(\vec{x})*, where \vec{x} is a vector of length *k*.

The definition ensures that each gate has a unique label (gate name plus natural number) as does—in particular—each input and each output. Self reference is mediated by an (explicit, external) input which makes the resulting terms well defined; perhaps that is more evident in the version involving mutual equations.

2.3 Induction principle

Since a circuit is defined in terms of all of its subcircuits (including perhaps itself), the principle for establishing a property of circuits is automatically one of 'strong induction'.

Circuit induction. Suppose that any circuit satisfies a property under the assumption that each of its (strict) subcircuits does so. Then every circuit satisfies the property.

The trivial case of that principle requires the property to hold for inputs, with no assumptions. Applications follow in Section 5.

3 Signals

This section introduces various signal models, with emphasis on the relationship between them and when it is appropriate to use one model rather than another. Although the most detailed model could be used throughout, preference is given to using the simplest model appropriate to the behaviour being studied at any point. Sometimes that involves just the value of a signal when it stabilises; sometimes it involves the values before and after stabilisation; sometimes it involves also an intermediate value; and sometimes it involves values at regular times.

3.1 Boolean model

Under the assumption that a signal value stabilises to a Boolean value, sometimes it is sufficient—for example in the elementary analysis of combinational circuits—to model each signal with just that stable value. The result is the ‘Boolean model’ of signals.

Definition of Boolean model. In the *Boolean model* an observable s is declared

$$\begin{aligned} \mathbb{S}_1 &:= \mathbb{B} \\ s &: \mathbb{S}_1. \end{aligned}$$

In this paper the two values are represented 0 (for low, or false) and 1 for its opposite (high, or true).

Thus a logic gate, *gate*, taking Boolean inputs a and b and with Boolean output $a \otimes b$ is modelled by a typed predicate

$$\begin{aligned} a, b &: \mathbb{S}_1 \\ \text{gate}(a, b) &= a \otimes b. \end{aligned}$$

Examples are provided by the standard combinational gates. An inverter has only a single input:

$$\begin{aligned} a &: \mathbb{S}_1 \\ \text{inv}(a) &= \neg a, \end{aligned}$$

whilst an exclusive-or gate has two:

$$\begin{aligned} a, b &: \mathbb{S}_1 \\ \text{exor}(a, b) &= a \oplus b \end{aligned}$$

where the operator \oplus denotes exor.

3.2 Binary model

For the analysis of state, again under the assumption that signal values stabilise, two values are required: the values before (s) and after (s') stabilisation. This ‘binary model’ is appropriate for the elementary analysis of sequential circuits.

Definition of binary model. Being informal and denoting an observable in the binary model by the same symbol as its initial value, the *binary model* of signals is:

$$\begin{aligned} \mathbb{S}_2 &:= \mathbb{B} \times \mathbb{B} \\ s : \mathbb{S}_2 &:= s, s' : \mathbb{B}. \end{aligned}$$

That model corresponds to the binary-relational model of computation (in which nondeterminism is captured by multiple values s' for a given initial s , and non-stabilisation—divergence—by introduction of a ‘virtual’ final value \perp).

For example a T-latch which toggles its state s according to a combinational input a is modelled

$$(1) \quad \begin{aligned} a &: \mathbb{S}_1 \\ s &: \mathbb{S}_2 \\ s' &= a \oplus s. \end{aligned}$$

One connection between the two models is expressed by a projection π_2 from the more detailed binary model to the Boolean model, that maps the stabilised value of the sequential model to the constant value of the combinational one:

$$\begin{aligned} \pi_2 &: \mathbb{S}_2 \rightarrow \mathbb{S}_1 \\ \pi_2(s, s') &:= s'. \end{aligned}$$

3.3 Ternary model

By capturing a stabilised signal value the sequential model is unable to express transient, or hazardous, behaviour. For that an intermediate, transient, value must be observed. Each signal is represented by its initial value (s), transient value (s') and stabilised value (s''). The result is:

Definition of ternary model. In the *ternary model* we are again informal and use the same symbol for the ternary observable and its initial value.

$$\begin{aligned}\mathbb{S}_3 &:= \mathbb{B} \times \mathbb{B} \times \mathbb{B} \\ s : \mathbb{S}_3 &:= s, s', s'' : \mathbb{B}\end{aligned}$$

For example a rising transient is expressed

$$\begin{aligned}s &: \mathbb{S}_3 \\ \neg s \wedge s' \wedge \neg s'' &.\end{aligned}$$

The more concrete ternary model is connected to the binary model in several ways. Perhaps the most obvious (for the purpose of hazard analysis) is by a projection π_3 that relates the initial and final values in the two models:

$$\begin{aligned}\pi_3 &: \mathbb{S}_3 \rightarrow \mathbb{S}_2 \\ \pi_3(s, s', s'') &:= (s, s'').\end{aligned}$$

Another way is by letting the binary model capture changes of signal in the ternary model

$$\begin{aligned}\pi'_3 &: \mathbb{S}_3 \rightarrow \mathbb{S}_2 \\ \pi'_3(s, s', s'') &:= (s \neq s', s' \neq s'').\end{aligned}$$

That relationship underlies the ‘event’ interpretation of signals which forms the foundation of the various disciplines of asynchronous design; π_3 will be identified in Section 4.4 as (a restricted version of) differentiation.

3.4 Signal model

The repeated observation of a signal at regular discrete times is modelled by a countable number of Boolean values. Use of the integers \mathbb{Z} for the time domain abstracts both initialisation (which would have to be defined explicitly at 0 if instead the natural numbers were used) and number of observations (which could be insufficient if instead any fixed finite set of times were used). Although in principle that choice of time domain suffers by not facilitating intermediate observations (possible with either the rational or real numbers), in this paper that facility is not required and is anyway outweighed by the advantage of having a unique predecessor for each time.

Definition of signal. A signal has a Boolean value at each time. Thus the set of all signals is the set of functions from the time domain to the Booleans.

$$\mathbb{S} := \mathbb{Z} \rightarrow \mathbb{B}$$

For example a clock with period 1 is expressed in the signal model by an alternating function

$$\begin{aligned} \text{clock} &: \mathbb{S} \\ \text{clock}(n+1) &:= \neg \text{clock}(n). \end{aligned}$$

There are two such functions, one with $\text{clock}(0)$ and the other with $\neg \text{clock}(0)$.

The two constant signals will play an important role in analysing circuits. We write them

$$\begin{aligned} \mathbf{0}, \mathbf{1} &: \mathbb{S} \\ \mathbf{0} &:= \lambda n : \mathbb{Z} \cdot 0 \\ \mathbf{1} &:= \lambda n : \mathbb{Z} \cdot 1, \end{aligned}$$

and write $\text{const}(s)$ iff signal s is constant.

The *Heaviside function* represents a signal that is switched on ‘now’ and remains on

$$\begin{aligned} \text{heavi} &: \mathbb{S} \\ \text{heavi}(n) &:= (n \geq 0). \end{aligned}$$

The more concrete signal model may be connected to the transient model in various ways. One obvious but stringent way interprets the three observations of the ternary model to be three consecutive observations in the signal model: previously (time -1), now (time 0) and next (time 1). In other words a signal $s : \mathbb{S}$ is related to a ternary signal $t : \mathbb{S}_3$ iff $t = s(-1)$, $t' = s(0)$ and $t'' = s(1)$.

However for the purposes of this paper that is too restrictive, and delays are appropriate between the three observations (see Theorem 3). The result is a many-to-many relation π between the signal model \mathbb{S} and the ternary model \mathbb{S}_3 , defined in infix:

$$\begin{aligned} \pi &: \mathbb{S} \leftrightarrow \mathbb{S}_3 \\ s \pi t &:= \exists k < l : \mathbb{Z} \cdot \left(\begin{array}{l} \forall n : (-\infty, k] \cdot s(n) = t \\ \forall n : (k, l] \cdot s(n) = t' \\ \forall n : (l, \infty) \cdot s(n) = t'' \end{array} \right). \end{aligned}$$

In other words, signals in the domain of π are piecewise constant with at most three pieces, whose values match the three values of the triple. Using the notation of function restriction: $t = s \upharpoonright (-\infty, k]$, $t' = s \upharpoonright (k, l]$ and $t'' = s \upharpoonright (l, \infty)$.

If the restriction $k < l$ is omitted then the case $(k, l] = \{ \}$ is included and two values t, t'' may result. This can also be achieved by composition with the projection $\pi_2 : \mathbb{S}_3 \rightarrow \mathbb{S}_2$. Indeed that yields an analogous connection between \mathbb{S} and the binary model \mathbb{S}_2 which abstracts the intermediate, transient, values of the signal

$$\begin{aligned} \pi' &: \mathbb{S} \leftrightarrow \mathbb{S}_2 \\ s \pi' t &:= \exists k, l : \mathbb{Z} \cdot \left(\begin{array}{l} \forall n : (-\infty, k] \cdot s(n) = t \\ \forall n : (l, \infty) \cdot s(n) = t' \end{array} \right). \end{aligned}$$

The projection π'_3 from Section 3.3, that records signal changes in the ternary model with results in the binary model, may be realised as the relational composition of signal differentiation and the projections between signals and those two models.

3.5 Circuits

Any of those signal models can be used to formulate a notion of ‘timed circuit’ for use in the previous, entirely syntactic, definition of circuit. The following section contains definitions of the relevant time-based operations to facilitate such definitions. But in each case the definition consists, like those we have seen so far, of

a *signature*: declaring the observables and their types, together with

a *constraint*: a predicate, whose free variables are the observables, which captures the invariant relationship that holds between the observables.

For example the T-latch of (1) has a signature consisting of two observables, the first a signal $a : \mathbb{S}_1$ of Booleans (representing input) and the second a signal $s : \mathbb{S}_2$ of pairs of Booleans (representing the initial and final states). Its constraint is this relationship between them: at any time, the second of the final state equals the exor of the initial state and the input $s' = a \oplus s$.

4 Timed behaviour

This section provides the definitions of the various timed behaviours that underlie the temporal behaviour of circuits.

4.1 Signal delay

The definition of delay, or temporal translation of a signal, exploits the choice of time domain by using integer arithmetic. Delays of both sign are useful for specification and design—speedup as well as slowdown—although only the latter appears in implementations.

Definition of delay. It is convenient to write the value of the delay as a subscript (although see Law (3)) and to write the delay of signal s by m time units as the signal $\delta_m s$.

$$\begin{aligned} \delta &: \mathbb{Z} \rightarrow \mathbb{S} \rightarrow \mathbb{S} \\ (\delta_m s)(n) &:= s(n - m) \end{aligned}$$

The unit delay δ_1 is abbreviated δ ; context will resolve any ambiguity with the previous definition. Thus a clock with period 1 satisfies, by definition,

$$\delta clock = \neg clock.$$

Several laws follow from the laws of arithmetic on \mathbb{Z} : successive delays are the same as one compound delay and so successive delays commute; a delay can be achieved by iterating unit delay; a delay by zero is no delay; delay has no effect on constant signals; Boolean connectives¹ are time invariant (in other words they commute with delay); an adjunction law holds, and it applies to show that the delay of the characteristic function of an interval (k, ∞) is again an interval of that form. See Figure 1.

$$\begin{aligned} (2) \quad & \delta_l \delta_m = \delta_{l+m} = \delta_m \delta_l \\ (3) \quad & \delta_m = \delta^m \\ (4) \quad & \delta_0 = \text{id} \\ (5) \quad & \text{const}(s) \Rightarrow \delta_m s = s \\ (6) \quad & \delta_m (\otimes (r_i)_i) = \otimes (\delta_m r_i)_i \\ (7) \quad & \delta_m s = t \Leftrightarrow s = \delta_{-m} t \\ (8) \quad & \delta_m (\lambda n : \mathbb{Z} \cdot (k < n)) = \lambda n : \mathbb{Z} \cdot (k - m < n) \end{aligned}$$

Figure 1: Some laws for delay, where $l, m : \mathbb{Z}$, \otimes is a Boolean connective with vector $(r_i)_i$ of arguments, and $k : \mathbb{Z} \cup \{\pm\infty\}$ with the usual arithmetic conventions concerning $\pm\infty$, so that $\{n : \mathbb{Z} \mid -\infty < n\} = \mathbb{Z}$ and $\{n : \mathbb{Z} \mid \infty < n\} = \{\}$.

4.2 Gate delay

A gate with inputs $in0$ and $in1$ which delays its output $out0(\text{gate}(in0, in1)) = in0 \otimes in1$ (there expressed by lifting the Boolean connective \otimes to signals) by m time units, is defined

$$\begin{aligned} in0, in1, out0 & : \mathbb{S} \\ out0(\text{gate}(in0, in1)) & = \delta_m (in0 \otimes in1). \end{aligned}$$

4.3 Feedback

Care must be taken, in the various timed models, in interpreting circuits with feedback.

¹Each connective has an arity. For example the constants (power and ground, or high and low respectively, have arity 0; inversion has arity 1; conjunction and disjunction have arity 2; and conditional has arity 3.

In some cases there is no difficulty. For instance the circuit

$$out0(AND0(in0, AND0))$$

has invariant

$$out0 = in0 \wedge out0.$$

Thus if input and output are initially high, the circuit is stable. If input then goes low the invariant fails and the circuit stabilises after some delay with output low. Such behaviour is accurately described in any of the timed models.

However the same is not true of the circuit

$$out0(NAND0(in0, NAND0))$$

which has invariant

$$out0 = \neg(in0 \wedge out0).$$

If initially input is low and output high, the circuit is stable. But if input then goes high, the invariant fails and after some delay output goes low (assuming that input does not change meanwhile). But that also violates the invariant so after some further time (again assuming no input change) output returns high; and so on. This indefinite oscillation of output is captured in the signal model but not in any of the models which assume a stable final value for the output signal.

Thus we must take care to choose an appropriate model for the behaviour we seek to analyse.

4.4 Signal change

Definition of derivatives. For a signal s , its rising derivative D^+s is defined to be the signal which is high now iff s is high now but was previously low. The falling derivate, D^-s , is high iff the signal was previously high and is now low. And the derivative Ds is high iff the signal changes value. Lifting the Boolean operations of negation, conjunction and disjunction pointwise to signals those conditions are expressed as follows.

$$\begin{aligned} D^+, D^-, D &: \mathbb{S} \rightarrow \mathbb{S} \\ D^+s &:= \neg(\delta s) \wedge s \\ D^-s &:= \delta s \wedge \neg s \\ Ds &:= D^+s \vee D^-s \end{aligned}$$

For convenience of bracketing, the delay and differentiation operators are assumed to bind more tightly than the lifted logical operators.

The laws for differentiation follow by routine calculation; see Figure 2. Law (9) connects the two kinds of differentiation; Law (10) records that the only signals without change are the two constant signals;

Law (11) records that the Heaviside function is the only signal whose rising derivative is high at just 0. Law (12) shows that the derivative of a signal equals that of its negation. Laws (13) and (14) provide alternative definitions of derivative using the simple fact that a signal does not rise and fall simultaneously. Law (15) gives time invariance of differentiation. Laws (16) to (18) give the discrete version of the product rule from differential calculus; Laws (19) to (21) do the same for disjunction; and Laws (22) to (24) do the same for *exor* (the last of which is reassuringly simple after its predecessors). Laws (25) to (28) demonstrate a striking difference with the theory of differential calculus: higher derivatives simplify. Law (29) records the fact that a signal does not rise (respectively fall) on consecutive intervals. For further laws we refer to [11].

4.5 Between models

The various connections π between the temporal models facilitate the translation of concepts between them—though evidently a concept translates to a less rich one in a more abstract model. For example the three concepts of signal change, defined in the previous section for the signal model, translate to the ternary model as introduced in Section 3.3.

Theorem 1 The notions of derivative, lifted from the signal model to the ternary model using the connections π and π' , simplify

$$\begin{aligned}\pi'(D^+(\pi^\sim(t, t', t''))) &= (\neg t \wedge t', \neg t' \wedge t'') \\ \pi'(D^-(\pi^\sim(t, t', t''))) &= (t \wedge \neg t', t' \wedge \neg t'') \\ \pi'(D(\pi^\sim(t, t', t''))) &= (t \neq t', t' \neq t'').\end{aligned}$$

Proof Consider just the first case. We evaluate the lifting of $(t, t', t'') : \mathbb{S}_3$ by the relational composition $\pi' \cdot D^+ \cdot \pi^\sim$ starting with those signals $s : \mathbb{S}$ for which $s \pi(t, t', t'')$. By definition (Section 3.4), $s \pi(t, t', t'')$ iff there exist $k, l : \mathbb{Z}$ such that

$$\begin{aligned}s \upharpoonright (-\infty, k] &= t \\ s \upharpoonright (k, l] &= t' \\ s \upharpoonright (l, \infty) &= t''.\end{aligned}$$

The rising derivative D^+s of any such s is low except perhaps at the two times at which s changes value, when its value is determined by that of the triple:

$$\begin{aligned}D^+s \upharpoonright (-\infty, k] \cup (k+1, l] \cup (l+1, \infty) &= \lambda n : \mathbb{Z} \cdot 0 \\ D^+s(k+1) &= \neg t \wedge t' \\ D^+s(l+1) &= \neg t' \wedge t''.\end{aligned}$$

Using π' (Section 3.4) to translate back to the Boolean model \mathbb{S}_2 , the derivative D^+s lies in the domain of π' iff it changes value at most once: it is low at at least one of those two discrete points. In either case a related pair $u : \mathbb{S}_2$ satisfies

$$\begin{aligned}u &= \neg t \wedge t' \\ u' &= \neg t' \wedge t''\end{aligned}$$

as claimed. □

- (9) $D^+(\neg s) = D^-s$
- (10) $Ds = \mathbf{0} \Leftrightarrow \text{const}(s)$
- (11) $s = \text{heavi} \Leftrightarrow D^+s = \lambda n : \mathbb{Z} \cdot (n = 0)$
- (12) $Ds = D(\neg s)$
- (13) $= (s \neq \delta s)$
- (14) $= D^+s \oplus D^-s$
- (15) $D(\delta s) = \delta(Ds)$
- (16) $D^+(s \wedge t) = (D^+s \wedge t) \vee (s \wedge D^+t)$
- (17) $D^-(s \wedge t) = (D^-s \wedge \delta t) \vee (\delta s \wedge D^-t)$
- (18) $D(s \wedge t) = (D^+s \wedge t) \vee (s \wedge D^+t) \vee (D^-s \wedge \delta t) \vee (\delta s \wedge D^-t)$
- (19) $D^+(s \vee t) = (D^+s \wedge \neg \delta t) \vee (\neg \delta s \wedge D^+t)$
- (20) $D^-(s \vee t) = (D^-s \wedge \neg t) \vee (\neg s \wedge D^-t)$
- (21) $D(s \vee t) = (D^+s \wedge \neg \delta t) \vee (\neg \delta s \wedge D^+t) \vee (D^-s \wedge \neg t) \vee (\neg s \wedge D^-t)$
- (22) $D^+(s \oplus t) = (D^+s \wedge \neg t \wedge \neg \delta t) \vee (D^-s \wedge t \wedge \delta t) \vee (\neg s \wedge \neg \delta s \wedge D^+t) \vee (s \wedge \delta s \wedge D^-t)$
- (23) $D^-(s \oplus s') = (D^-s \wedge \neg \delta t \wedge \neg t) \vee (D^+s \wedge \delta t \wedge t) \vee (s \wedge \delta s \wedge D^+t) \vee (\neg s \wedge \neg \delta s \wedge D^-t)$
- (24) $D(s \oplus t) = Ds \oplus Dt$
- (25) $D^+D^+s = D^+s$
- (26) $D^-D^-s = \delta D^-s$
- (27) $D^+D^-s = D^-s$
- (28) $D^-D^+s = \delta D^+s$
- (29) $D^+s \wedge D^+\delta s = \mathbf{0} = D^-s \wedge D^-\delta s$

Figure 2: Some laws of differentiation; throughout, $s, t : \mathbb{S}$ are signals.

5 Hazard freedom

A circuit can be modelled, at any of those levels of abstraction, as a set of observables satisfying a constraint. For example an inverter can be modelled by two signals, *in* and *out* in which the latter is a delayed version of the negation of the former: $out = \delta in$. An and gate can be modelled by three signals, *a*, *b* and *c*; *a* and *b* are inputs and *c* is output and the latter is a delayed conjunction of the two inputs: $c = \delta(a \wedge b)$.

In this section we use the signal view to analyse transient behaviour.

5.1 Constant input

A very simple result is:

Theorem 2 For any circuit, if all its inputs are constant then all its outputs are also constant.

Proof A signal *s* is constant iff its derivative is always low (Law (10)). So the assumption implies that for each input signal *in*, $D(in) = \mathbf{0}$.

Now for each gate, if its inputs are constant so are its outputs: for inverters, conjunction, disjunction and exor that follows from Laws (12), (18), (21) and (24); it is readily established directly for any combinational gate as follows using Law (10). Suppose that *gate* is a combinational gate with inputs *s* and *t* satisfying $gate(s, t) = \delta_k(s \otimes t)$ for some $k > 0$, where \otimes is some logical connective lifted pointwise to signals. Then

$$\begin{aligned}
 D(gate(s, t)) &= \mathbf{0} \\
 &= && \text{Law (10)} \\
 \forall n : \mathbb{Z} \cdot gate(s, t)(n) &= gate(s, t)(n-1) \\
 &= && \text{definition of } gate \\
 \forall n : \mathbb{Z} \cdot \delta_k(s \otimes t)(n) &= \delta_k(s \otimes t)(n-1) \\
 &= && \text{change of variable and pointwise lifting of } \otimes \text{ to signals} \\
 \forall n : \mathbb{Z} \cdot s(n) \otimes t(n) &= s(n-1) \otimes t(n-1) \\
 \Leftarrow &&& \text{calculus with Leibniz's rule} \\
 \forall n : \mathbb{Z} \cdot s(n) = s(n-1) \wedge t(n) = t(n-1) & \\
 &= && \text{Law (10) again} \\
 Ds = \mathbf{0} = Dt. &
 \end{aligned}$$

Finally the result follows by induction over circuits. □

It is not assumed there that the circuit is combinational. In view of the nature of the statement being proved, it would suffice to establish it in the binary model of signals which captures one change of value (*cf.* Section 4.3); however the proofs are surprisingly similar given the laws in Figure 2, so the more general proof has been presented. The assumption that the gate delay k is positive avoids circularity in the application of induction.

5.2 Single input change

A circuit is called *monotone* iff it responds to input having at most one change in value with output having at most one change of value.

Theorem 3 A circuit composed of only *and*, *or* and *fork* gates is monotone; in particular, when subject to a single input change it is hazard free.

Proof By way of proof we establish a little more: if one input undergoes a single change then each output is either constant or undergoes the same kind of change. Again, we reason in the signal model, and for simplicity consider the single change in input to be an increase.

We begin by defining the behaviour we are assuming of each input and which we must establish for each output: $\uparrow s$ holds iff signal s is either constant or an increasing step function (that is, the characteristic function of some interval (m, ∞))²

$$\begin{aligned} \uparrow : \mathbb{S} &\rightarrow \mathbb{B} \\ \uparrow s &:= \text{const}(s) \\ &\vee \\ &\exists m : \mathbb{Z} \cdot s = \lambda n : \mathbb{Z} \cdot (n > m). \end{aligned}$$

Then by Law (8),

$$(30) \quad \uparrow \delta s = \uparrow s.$$

By assumption, the desired property holds for each input subcircuit: $\forall j \cdot \uparrow inj$. Suppose that *gate* is as in the proof of Theorem 2, but monotone:

$$(31) \quad (\uparrow s \wedge \uparrow t) \Rightarrow \uparrow (s \otimes t).$$

To show $\uparrow out0(\text{gate})$ we calculate

²An alternative, perhaps slightly nicer, definition is $\uparrow s = \exists m : \mathbb{Z} \cup \{\pm\infty\} \cdot s = \lambda n : \mathbb{Z} \cdot (n > m)$. A more sophisticated alternative, which does not seem to help here, is $\delta s \leq s$. Notation which is more expressive and of wider use will appear in the next section. But this simple notation suffices for the present proof.

$$\begin{aligned}
& \uparrow_{\text{out}} 0(\text{gate}) \\
& = \text{definition of } \text{gate} \\
& \uparrow \delta_k (s \otimes t) \\
& = \text{Laws (30) and (3)} \\
& \uparrow (s \otimes t) \\
& \Leftarrow (31) \\
& \uparrow s \wedge \uparrow t.
\end{aligned}$$

The result follows by induction over circuits. \square

5.3 Transients

We capture transient behaviour by signals that exhibit a rising or falling step for a given length but are otherwise constant. That restrictive definition suits our current purposes; a more general one would allow arbitrary behaviour in the distant past and future. Furthermore by separating the definition of transient from that of a hazard, the latter term can be defined according to context—of what length transient is considered hazardous—and of the discipline that produces it—single or multiple input change.

Definition of transients. Indicating the length parameter k as a subscript, the *rising transient* signal up_k is defined to be high on just the interval $[0, k)$ and the *falling hazard* signal dn_k to be low on just that interval.

$$\begin{aligned}
up, dn & : \mathbb{N} \rightarrow \mathbb{S} \\
up_k & := \lambda n : \mathbb{Z} \cdot 0 \leq n < k \\
dn_k & := \neg up_k
\end{aligned}$$

A signal is said to exhibit a rising transient of length k iff it equals up_k and a falling transient iff it equals dn_k . For symbolic purposes that is recorded by predicates \sqsupset_k and \sqsubset_k on signals.

$$\begin{aligned}
\sqsupset, \sqsubset & : \mathbb{N} \rightarrow \mathbb{S} \rightarrow \mathbb{B} \\
\sqsupset_k s & := (s = up_k) \\
\sqsubset_k s & := (s = dn_k)
\end{aligned}$$

That notation is more restrictive than might at first be regarded as reasonable, since it requires a transient to begin at time 0 (rather than at an arbitrary time). We have chosen it for simplicity; as a result we need this definition.

Definition of transient freedom. To say that signal s *never* exhibits a rising transient is to say that no translation of s equals a rising step of positive length at the origin; the definition of never exhibiting a falling transient is analogous.

$$\begin{aligned}
\text{never} \sqsupset & : \mathbb{S} \rightarrow \mathbb{B} \\
\text{never} \sqsupset (s) & := \neg \exists k, l : \mathbb{Z} \cdot \begin{pmatrix} k > 0 \\ \sqsupset_k \delta_l s \end{pmatrix}
\end{aligned}$$

Laws for transients are not easy to come by, but some are given in Figure 3. The first says that a rising transient of length 0 reduces to the constant low signal and the second that the signal constant at high exhibits a transient only of length 0. The third relates rising and falling transients via inversion, whilst the fourth identifies a rising transient in terms of the locations of its rising and falling derivatives. Laws (36) and (37) show that a binary conditional distributes transients provided the condition does not refer to time. The final law identifies the way in which a (rising) transient is affected by delay, so in particular is important for reasoning about *never*.

$$\begin{aligned}
(32) \quad & up_0 = \mathbf{0} \\
(33) \quad & \sqcap_k \mathbf{1} = k = 0 \\
(34) \quad & \sqcap_k s = \sqcup_k \neg s \\
(35) \quad & = \left(\begin{array}{l} D^+ s = \lambda n : \mathbb{Z} \cdot n = 0 \\ D^- s = \lambda n : \mathbb{Z} \cdot n = k \end{array} \right) \\
(36) \quad & \sqcap_k (s \triangleleft c \triangleright t) = (\sqcap_k s) \triangleleft c \triangleright (\sqcap_k t) \quad \text{if time variable } n \text{ is not free in } c \\
(37) \quad & \sqcup_k (s \triangleleft c \triangleright t) = (\sqcup_k s) \triangleleft c \triangleright (\sqcup_k t) \quad \text{if time variable } n \text{ is not free in } c \\
(38) \quad & \sqcap_k \delta_l s = (s = \lambda n : \mathbb{Z} \cdot n \in [-l, k-l])
\end{aligned}$$

Figure 3: Some laws for transients, with $k : \mathbb{N}$ and $s : \mathbb{S}$.

5.4 Restricted case of a conditional gate

Consider the simplest circuit capable of a race (in which unnecessary labels have been omitted):

$$out(or(in, not(in))),$$

where the *not* gate has delay $d : \mathbb{Z}$ but, for simplicity, the *or* gate has no delay (since it would only shift the position of a transient, not affect its occurrence; see Corollary 5):

$$\begin{aligned}
(39) \quad & in, out : \mathbb{S} \\
& d : \mathbb{Z} \\
& out = in \vee \delta_d \neg in.
\end{aligned}$$

Elementary considerations show that with a single input change, *out* never exhibits a rising transient, but exhibits a falling transient if the delay is positive and input falls, or delay is negative and input rises. More precisely:

Theorem 4 For a single input change in the circuit (39), the predicate $never\sqcap(out)$ holds, and if $d \neq 0$ then the existence of a falling transient on output is determined by a conditional:

$$\forall k : \mathbb{N}^+ \cdot \sqcup_k out \Leftrightarrow \left(\begin{array}{l} k = d \\ in = \neg heavi \end{array} \right) \triangleleft d > 0 \triangleright \left(\begin{array}{l} k = -d \\ in = \delta_{-d} heavi \end{array} \right).$$

Proof A signal with a single change of value is a translation of either the Heaviside function or its negation (Laws (9), (11) and (15)). Considering the case $in = heavi$, we reason

$$\begin{aligned} & out \\ = & \hspace{15em} \text{definition of circuit (39)} \\ & heavi \vee \neg \delta_d heavi \\ = & \hspace{10em} \text{definition of } heavi \text{ and Law (6) for } \neg \\ & (n \geq 0) \vee (n - d < 0) \\ = & \hspace{15em} \text{calculus} \\ & \mathbf{1} \triangleleft d \geq 0 \triangleright n \notin [d, 0) \\ = & \hspace{15em} \text{definition of } \delta_d \\ & (\delta_d \mathbf{1}) \triangleleft d \geq 0 \triangleright \delta_d (n \notin [0, -d)) \\ = & \hspace{15em} \text{Law (6)} \\ & \delta_d (\mathbf{1} \triangleleft d \geq 0 \triangleright n \notin [0, -d)). \end{aligned}$$

In other words,

$$in = heavi \Rightarrow out = \delta_d (\mathbf{1} \triangleleft d \geq 0 \triangleright n \notin [0, -d)).$$

Thus, by translation invariance of the connectives defining out (i.e. disjunction and inversion), Law (6), and by commutativity of delays, Law (2),

$$in = \delta_{-d} heavi \Rightarrow out = (\mathbf{1} \triangleleft d \geq 0 \triangleright n \notin [0, -d)).$$

Hence for any $k, l : \mathbb{Z}$ with $k > 0$,

$$\begin{aligned} & \sqcap_k \delta_l out \\ = & \hspace{15em} \text{above} \\ & \sqcap_k (\delta_l \mathbf{1} \triangleleft d \geq 0 \triangleright \delta_l (n \notin [0, -d))) \\ = & \hspace{15em} \text{Law (36)} \\ & (\sqcap_k \delta_l \mathbf{1}) \triangleleft d \geq 0 \triangleright (\sqcap_k \delta_l (n \notin [0, -d))) \\ = & \hspace{15em} \text{Law (5) and definition of } \delta_l \\ & (\sqcap_k \mathbf{1}) \triangleleft d \geq 0 \triangleright (\sqcap_k n \notin [-l, k - l)) \\ = & \hspace{15em} \text{Laws (33) and (38)} \\ & 0 \triangleleft d \geq 0 \triangleright 0 \\ = & \hspace{15em} \text{calculus} \\ & 0 \end{aligned}$$

so that the predicate $never \sqcap (out)$ holds. Furthermore,

$$\begin{aligned}
& \sqcup_k out \\
& = \\
& \sqcup_k (\mathbf{1} \triangleleft d \geq 0 \triangleright n \notin [0, -d]) \\
& = \qquad \qquad \qquad \text{Law (36) and definition of } \delta \\
& (\sqcup_k \mathbf{1}) \triangleleft d \geq 0 \triangleright (\sqcup_k n \notin [0, -d]) \\
& = \qquad \qquad \qquad \text{Laws (34) and (38)} \\
& 0 \triangleleft d \geq 0 \triangleright (k = -d) \\
& = \qquad \qquad \qquad \text{calculus} \\
& k = -d > 0.
\end{aligned}$$

In this case we conclude

$$in = \delta_{-d} heavi \Rightarrow \forall k : \mathbb{N}^+ \cdot \sqcup_k out = (k = -d > 0).$$

The remaining case, $in = \neg heavi$, follows symmetrically: again $never \sqcap (out)$ holds, but now

$$in = \neg heavi \Rightarrow \forall k : \mathbb{N}^+ \cdot \sqcup_k out = (k = d > 0).$$

which completes the proof. \square

In order to apply Theorem 4 in a more realistic context, in the next section it will be convenient to use the following intermediate result concerning circuit (39) augmented by a further delay:

$$\begin{aligned}
in, out & : \mathbb{S} \\
a, d & : \mathbb{Z}
\end{aligned}$$

$$(40) \quad out = \delta_a (in \vee \delta_d \neg in).$$

Corollary 5 The output of circuit (40) never exhibits a rising transient but, for $d \neq 0$, exhibits a falling transient (by definition, it must be recalled, starting at the origin) of length $k > 0$ iff

$$\left(\begin{array}{l} k = d \\ in = \neg \delta_a heavi \end{array} \right) \triangleleft d > 0 \triangleright \left(\begin{array}{l} k = -d \\ in = \delta_{a-d} heavi \end{array} \right).$$

\square

5.5 Conditional gate

The real interest in circuit (39) is that it embodies the transient behaviour of this circuit

$$(41) \quad out(or(and0(in0, not(in2)), and1(in1, in2)))$$

implementing a conditional gate whose Boolean behaviour is

$$\begin{aligned} in0, in1, in2, out &: \mathbb{B} \\ out &= (in1 \triangleleft in2 \triangleright in0) = (in0 \wedge \neg in2) \vee (in1 \wedge in2) \end{aligned}$$

and whose timed (signal) behaviour we represent

$$(42) \quad out = \delta_a(\delta_b(in0 \wedge \delta_d \neg in2) \vee \delta_c(in1 \wedge \delta_e in2)),$$

where the delays a, b, c, d come respectively from the *or*, *and0*, *and1* and *not* gates and delay e is incurred by the path of the signal $in2$.

The transient behaviour derives entirely from the simpler circuit (39):

Theorem 6 For a single input change in the circuit (42), output never exhibits a rising transient but, for $b + d - c - e \neq 0$, exhibits a falling transient of length $k > 0$ iff

$$(in0 = \mathbf{1}) \wedge (in1 = \mathbf{1}) \wedge \left(\left(\begin{array}{l} k = y \\ in2 = \neg \delta_x heavi \end{array} \right) \triangleleft y > 0 \triangleright \left(\begin{array}{l} k = -y \\ in2 = \delta_{x-y} heavi \end{array} \right) \right),$$

where $x := a + c + e$ and $y := b + d - c - e$.

Proof If the single input change occurs on either $in0$ or $in1$, so that no race occurs, then Laws (2) and (5) show that output is a translation of the changing input and hence transient free. So it remains to consider the case in which the single changing input is $in2$.

If either $in0$ or $in1$ is low then the same reasoning involving Laws (2) and (5) shows that no transient occurs. So we are left with $(in0 = \mathbf{1}) \wedge (in1 = \mathbf{1})$ in which case (42) simplifies

$$\begin{aligned} out & \\ &= \text{calculus} \\ &\delta_a(\delta_b(\delta_d \neg in2) \vee \delta_c(\delta_e in2)) \\ &= \text{Laws (7), (6), (2) and (4)} \\ &\delta_{a+c+e}((\delta_{b+d-c-e} \neg in2) \vee in2) \\ &= \text{taking } x := a + c + e \text{ and } y := b + d - c - e \\ &\delta_x((\delta_y \neg in2) \vee in2). \end{aligned}$$

Thus, by Corollary 5, $never \sqcap (out)$ holds and, continuing with those values for x and y , output exhibits a falling transient iff

$$\left(\begin{array}{l} k = y \\ in2 = \neg \delta_x heavi \end{array} \right) \triangleleft y > 0 \triangleright \left(\begin{array}{l} k = -y \\ in2 = \delta_{x-y} heavi \end{array} \right).$$

□

5.6 Iterative multiplexor

Definition of multiplexor. A multiplexor of size $n : \mathbb{N}$ contains n control inputs, 2^n Boolean data inputs and a single Boolean data output which equals the data input addressed by control. We choose to give its Boolean specification in terms of the bijection $bin : \mathbb{B}^n \rightarrow [0, 2^n)$ which converts a bit string to a natural number by binary expansion (with x_0 least significant and x_{n-1} most significant).

$$\begin{aligned} n &: \mathbb{N} \\ con &: \mathbb{B}^n \\ in &: [0, 2^n) \rightarrow \mathbb{B} \\ out &: \mathbb{B} \\ mux(n, con, in, out) &:= out = in(bin(con)) \end{aligned}$$

where

$$\begin{aligned} bin &: \mathbb{B}^n \rightarrow [0, 2^n) \\ bin(x) &:= \sum_{0 \leq i < n} x_i 2^i \end{aligned}$$

A multiplexor of size 0 has no controls and simply outputs its single Boolean input; a multiplexor of size 1 forms a conditional.

That (iterative, rather than recursive) definition of bin leads (well, following a formal derivation it does) to the standard circuit that is based on disjunctive normal form to ‘decode’ control and hence select an input. Its description involves the parameter n and hence a variable number of components, dependent on n . It will thus be helpful to adopt the convention, proposed in Section 2.2, of subscripting each gate identifier with its arity.

The Boolean description of the iterative multiplexor circuit contains a single output from its single 2^n -input *or* gate which inputs from 2^n -many $(n+1)$ -input *and* gates

$$(43) \quad out(or_{2^n}((and_{n+1}(\vec{x}i))_{0 \leq i < 2^n}))$$

where the $n+1$ elements of vector $\vec{x}i$ contain one input, in_i , and n controls, half of which are inverted according to the usual binary pattern: if index $i : [0, 2^n)$ then

$$(44) \quad \vec{x}i := (in_i, B(i, j)_{0 \leq j < n})$$

where the ‘binary pattern’ is captured by the presence or absence of an inverter on control con_j at gate and_i :

$$B(i, j) := not(i, j)(con_j) \triangleleft i \bmod 2^j = 0 \triangleright con_j.$$

There the various inverters have been distinguished by identifier (i,j) , enabling us to assume each has its own delay.

The timed (signal) behaviour we consider incorporates delays at just the inverters (since, as we have already seen, other delays do not affect the existence of a transient, just its location). Suppose that the inverter $not(i,j)$ has delay $d(i,j)$. The timed behaviour is described by reinterpreting (43) and (44) with signals, and replacing the previous binary pattern with this timed version

$$(45) \quad B(i,j) := \delta_{d(i,j)}(\neg conj) \triangleleft i \bmod 2^j = 0 \triangleright conj.$$

The presence of transients in that circuit follows the pattern which has by now become standard: *out* never exhibits a rising transient but exhibits a falling transient as determined by Theorem 6. The two inputs that must be high to provide the race leading to a transient are determined by the values of the control bits together with the two values of the changing control bit *conj*. We calculate their indices using the list-processing functions³ *take*, *drop* and the function *bin*, and—in an attempt to abbreviate already-complex notation—leave implicit the dependency on *j*:

$$\begin{aligned} as_0 &:= (take\ j\ conj) ++ [0] ++ (drop\ (j+1)\ conj) \\ as_1 &:= (take\ j\ conj) ++ [1] ++ (drop\ (j+1)\ conj) \\ a_0 &:= bin(as_0) \\ a_1 &:= bin(as_1) \end{aligned}$$

so that the required inputs are ina_0 and ina_1 . Since exactly one of $(a_0 \bmod 2^j)$ and $(a_1 \bmod 2^j)$ is 0, exactly one of $B(a_0,j)$ and $B(a_1,j)$ results in an inverter; suppose the value of either a_0 or a_1 that gives the inverter is a . Having reduced the situation to a conditional gate, the following theorem follows from Theorem 6.

Theorem 7 For a single input change in the circuit given by (43), (44) and (45), output never exhibits a rising transient but exhibits a falling transient iff the change occurs in some control input, *conj* with $ina_0 = \mathbf{1}$ and $ina_1 = \mathbf{1}$ and $d(a,j) \neq 0$, in which case the transient is given by

$$\left(\begin{array}{l} k = d(a,j) \\ in2 = \neg heavi \end{array} \right) \triangleleft d(a,j) > 0 \triangleright \left(\begin{array}{l} k = -d(a,j) \\ in2 = \delta_{d(a,j)} heavi \end{array} \right).$$

5.7 Recursive multiplexor

A recursive recasting of the function *bin* leads instead to a recursive design based on a binary tree of conditionals. The recursive function *bins* takes an arbitrary nonempty sequence of Booleans to a natural

³The functions $take, drop : \mathbb{N} \rightarrow seq\ T \rightarrow seq\ T$ are defined so that *take n ts* returns the first *n* elements of sequence *ts*, and *drop n ts* returns the remaining tail. See [4].

number by binary expansion. Using $++$ for sequence catenation, and $\text{seq}^\bullet \mathbb{B}$ for the set of nonempty sequences of Booleans,

$$\begin{aligned} \text{bins} &: \text{seq}^\bullet \mathbb{B} \rightarrow \mathbb{N} \\ \text{bins}([0]) &:= 0 \\ \text{bins}([1]) &:= 1 \\ \text{bins}(xs ++ [x]) &:= \text{bin}(xs) + 2x. \end{aligned}$$

The restriction of bins to sequences of length n is readily checked to coincide with the function bin .

The recursive design is described, writing $\text{seq}_k \mathbb{B}$ for the set of sequences of \mathbb{B} of length k , in terms of the top and bottom half of the input sequence and a conditional gate.

$$\begin{aligned} n &: \mathbb{N} \\ \text{cons} &: \text{seq}_n \mathbb{B} \\ \text{ins} &: \text{seq}_{2^n} \mathbb{B} \\ \text{out} &: \mathbb{B} \\ \text{muxs}(0, \text{cons}, \text{ins}, \text{out}) &:= \text{out} = \text{ins} \\ \text{muxs}(n+1, [c] ++ \text{cons}, \text{ins}, \text{out}) &:= \text{muxs}(n, \text{cons}, \text{bot}(\text{ins}), \text{out}) \triangleleft c \triangleright \text{muxs}(n, \text{cons}, \text{top}(\text{ins}), \text{out}) \end{aligned}$$

where the two halves of the input list (which correspond to the restrictions $\text{in} \upharpoonright [0, 2^{n-1})$ and $\text{in} \upharpoonright [2^{n-1}, 2^n)$ respectively in the iterative design) are conveniently defined by take and drop : for list xs of (even) length $2k$,

$$\begin{aligned} \text{top}(xs) &:= \text{take } k \text{ } xs \\ \text{bot}(xs) &:= \text{drop } k \text{ } xs. \end{aligned}$$

That design is converted to a circuit by implementing the conditional as in (41) but, for simplicity as in the previous section, with delays at only the inverters. Although the resulting circuit appears markedly different from the iterative multiplexor, its transient behaviour is the same: from Theorem 7 we have

Corollary 8 The output of the recursive multiplexor exhibits the same transient behaviour as that of the iterative multiplexor.

6 Events

So far, a circuit has been modelled as a collection of observables—corresponding at some level of abstraction to values on wires (input, output or internal)—subject to a constraint reflecting the circuit’s functionality. In order to focus on that functionality we have made strong assumptions about the circuit’s environment: that it is always ready to accept output but delivers input only when convenient for the circuit. That suffices, as our results have shown, for a simple hazard analysis of circuits in isolation. But unfortunately that assumption is not ‘compositional’: a single input change may be conveyed (in

our model) by an isochronic fork to a component two of whose inputs change simultaneously as a result. Compositionality requires us to strengthen the theory to allow more general input changes. What, for instance, if the environment provides a second input change, whilst the circuit is in the process of producing output in response to the previous input change?

In this section we introduce a model for discussing such questions: the event model \mathbb{E} . We assume, as before, that the environment is always prepared to accept output from the circuit, but also that it may provide input to the circuit at any time.

6.1 Introducing events

So far we have been concerned entirely with models of signals in which the value of a signal is defined explicitly at each time. Thus if a signal s is to be low until time 0, high for the next four time units and thereafter low, it is described

$$\begin{aligned} s : \mathbb{S} \\ s(n) &:= (1 \leq n < 5). \end{aligned}$$

By contrast the event model describes only changes in value. Thus the signal s above has two changes, one at time 1 and the other at time 5, expressed by differentiation (recall the differentiation operator $D := D^+ \vee D^-$ introduced in Section 4.4):

$$\begin{aligned} s, Ds : \mathbb{S} \\ (Ds)(n) &:= n \in \{1, 5\}. \end{aligned}$$

However both s and $\neg s$ share the same derivative. So if a signal is to be determined uniquely then as well as its times of change, its value at some time must be known. For some signals it is convenient for that time to be ‘initially’, or at $-\infty$, though of course that concept is not defined for all signals (it is not defined, for instance, for a clock). For example signal s above is conveniently described as being low initially. Signals for which an initial value are defined, are described as follows.

Definition of initial value. A signal $s : \mathbb{S}$ has an *initial value* $c : \mathbb{B}$, written $\lim_{-\infty} s = c$, iff there is some integer $n_0 : \mathbb{Z}$ such that $\forall n < n_0 \cdot s(n) = c$. In other words the limit at $-\infty$ equals c in the discrete topology on \mathbb{B} (in which convergence is eventual equality). Similarly it has a *final value* c , $\lim_{\infty} s = c$, iff there is some integer $n_0 : \mathbb{Z}$ such that $\forall n > n_0 \cdot s(n) = c$.

Our final decision in defining \mathbb{E} is to restrict a signal Ds to just the times at which it is high, thus abstracting the times at which no change occurs. Now Ds is represented by its ‘compression’, s' :

$$s' := \text{compress}(Ds),$$

where the function *compress* maps a signal to the relation consisting just of those pairs whose result is high but with the high value replaced by the signal name

$$(46) \quad \begin{aligned} & \textit{compress} : \mathbb{S} \rightarrow (\mathbb{Z} \leftrightarrow A^+) \\ & \textit{compress}(s) := \{(n, s) : \mathbb{Z} \times A^+ \mid s(n)\}. \end{aligned}$$

In this example, $s' = \{(1, s), (5, s)\}$ in which the pair (n, s) is called an ‘event’ and interpreted to mean that s changes value at time n . Thus, in both the signal and event models a signal takes Boolean values. Interpreted as a member of \mathbb{S} , a signal’s values are given explicitly at each time; but interpreted as a member of the event model \mathbb{E} , only the times of a signal’s changes are given.

In relaxing the assumptions on a circuit’s environment we must first clarify whether a signal change is determined by the circuit or its environment. An *action* is a change determined by the environment, like a change in input. A *co-action* is a change determined by the circuit, like an output change or the change of an internal signal produced as an output from some subcircuit. Our assumption now becomes symmetrical: the environment always allows co-actions and the circuit always allows actions. We thus describe an environment by the way in which it schedules actions, and a circuit by the way in which it schedules co-actions in the context of an environment.

If a circuit is ‘unstable’ (*i.e.* in a state preparing an output change required by previous input changes) then an input event that causes an error (by acting before the output event to make the state stable and thus ‘losing’ the output) is called a *choke*. We model that by extending the normal model of a circuit to contain the extra signal *choke*, which is high iff one of the actions causes the circuit to choke. (The alternative, of extending signal values from \mathbb{B} to $\mathbb{B} \cup \{\textit{choke}\}$, would require elaboration of the calculus of signals.) The new signal *choke* satisfies an invariant: if *choke* changes value at time n then some action changes value at n . Of course in a particular circuit that invariant is strengthened to contain knowledge of which action causes the choke.

6.2 The event model

That discussion motivates the following definition, extended from one signal to a circuit (*i.e.* a signature declaring a set of signals, together with an invariant). It has been simplified in order to deal, in the remainder of this paper, with combinational circuits.

Definition of event model. Given a set A of signal names, let $A^+ := A \cup \{\textit{choke}\}$. The type $\mathbb{E}(A)$ consists of all relations from the time domain \mathbb{Z} to the set A^+

$$\mathbb{E}(A) := \mathbb{Z} \leftrightarrow A^+.$$

The members $t : \mathbb{E}(A)$ are called *multitraces* and the pairs $(n, s) \in t$ are called *events*. For each multitrace $t : \mathbb{E}(A)$, $(n, a) \in t$ iff $a \in A$ and at time n the signal a changes value, or

$a = \text{choke}$ and at time n some action causes a choke (notation for which will be introduced in Section 6.3).

For the study of circuits which are considered to have an initial state we let $\mathbb{E}^\circ(A)$ denote the set of multitraces t that are either empty or such that the times in t are well founded (so t contains an event whose time is minimal).

A multitrace is a relation—rather than a partial function in either direction—because different events may occur at the same time and one signal may change at different times; examples are provided by the running example of the inverter begun just below and by the *and* gate in Section 6.5. It is sufficient for a multitrace to be a *set* of events—rather than a *bag*—because each signal can change value at most once at any time. Thus the type of a multitrace implicitly provides it with that healthiness condition. We have imposed no other healthiness conditions (not even nonemptiness). Perhaps the most tempting would be that any output arises from a matching input, a property we must certainly consider. But it is not satisfied by a crystal oscillator (*i.e.* clock) and so is not imposed in general.

Let us consider an example, to be extended in the remaining subsections of this section.

Example. Recall that in the signal model an inverter with a delay of two time units functioning in a ‘benign’ environment (*i.e.* which alternates the supply of input events and the acceptance of output events, starting with an input, and hence which schedules no chokes) is specified simply

$$\begin{aligned} in, out &: \mathbb{S} \\ out &:= \neg\delta_2 in \end{aligned}$$

(and that, by Law (6), it makes no difference, whether the negation is placed before or after the delay operator). The choice of delay 2 there allows invalid ‘hasty’ input (though a unit integer delay does not) which we wish to analyse.

Taking derivatives we find,

$$\begin{aligned} &D(out)(n) \\ &= && \text{definition of circuit} \\ &D(\neg\delta_2 in)(n) \\ &= && \text{Laws (12) and (15)} \\ &\delta_2 D(in)(n). \end{aligned}$$

Thus in the event model, the inverter in a benign environment is represented by a multitrace t in which each input event engenders an output event two time units later

$$(47) \quad \begin{aligned} &t : \mathbb{E}(\{in, out\}) \\ &\forall n : \mathbb{Z} \cdot (n, in) \in t \Leftrightarrow (n+2, out) \in t. \end{aligned}$$

Notice that no inversion is apparent there, so that the description for a buffer would be the same: if at some time input is low and output high, or *vice versa*, then an inverter is described; otherwise a buffer results.

It is possible that for some time n_0 both $(n_0, in) \in t$ and $(n_0, out) \in t$: the environment produces the next input just as the circuit responds to the previous input. That is why a multitrace need not be a partial function even in this restricted circuit having just one input and one output. (That differs from the view taken in process algebra where simultaneous events can be observed only in succession in either order. We return to this point in the example of a two-input *and* gate in Section 6.5.)

This simple example is thus choke free. Its extension to environments that result in choke is considered in the next section. \square

Refinement between multitraces, $t \sqsubseteq t'$, is defined to ensure that t' implements t but perhaps in more environments.

Definition of refinement. For multitraces $t, t' : \mathbb{E}(A)$ we say that t is refined by t' , $t \sqsubseteq t'$, iff (a) t chokes when t' does, (b) t performs a co-action when t' does, and (c) if t does not choke then it performs an action when t' does. Assuming s denotes an action and s' a co-action:

$$(48) \quad t \sqsubseteq t' := \left(\begin{array}{l} (n, choke) \in t' \Rightarrow (n, choke) \in t \\ (n, s') \in t' \Rightarrow (n, s') \in t \\ (n, s) \in t' \wedge (n, choke) \notin t \Rightarrow (n, s) \in t \end{array} \right).$$

Using a standard technique, refinement is simplified to set containment at the expense of replacing each multitrace t by its ‘choke-strict expansion’, t^* , which is t when t does not choke, but arbitrary when it does:

$$t^* := t \cup \{(m, a) : \mathbb{Z} \times A^+ \mid m \geq n \wedge (n, choke) \in t\}.$$

For then

$$t \sqsubseteq t' \Leftrightarrow t^* \supseteq t'^*.$$

Either way, choke corresponds to \sqsubseteq -minimal behaviour:

$$(49) \quad \{(n, choke)\} \sqsubseteq \{(m, a) \mid m \geq n \wedge a : A^+\}.$$

In the event model a circuit is represented by a multitrace $t : \mathbb{E}(A)$. The connection with the signal model is given, as we have seen in the previous section, by differentiation and compression of each signal in A . A Galois connection is formulated as follows.

Consider a circuit having a set A of labels partitioned into its inputs, internal wires and outputs. Let S be a signal-level description, where the signature of S consists of the signals a_σ (for $a : A^+$) and the constraint of S is given by predicate s (whose free variables are the a_σ). Define the *choke*-strict constraint s^* (by analogy with t^*)

$$s^* := \forall n : \mathbb{Z} \cdot s \vee (Dchoke_\sigma)(n).$$

The representation function π from the signal model to the event model takes S to a multitrace $\pi(S)$ consisting of those events arising from changes of the signals a_σ subject to the constraint s :

$$\begin{aligned} \pi(S) &: \mathbb{E}(A) \\ \pi(S) &:= \{(n, a) : \mathbb{Z} \times A^+ \mid s \wedge (Da_\sigma)(n)\} \end{aligned}$$

(wherein Da_σ means $D(a_\sigma)$). It satisfies a Galois equivalence

$$\begin{aligned} \pi(S) &\sqsubseteq t && \text{definition of } \pi \\ \Leftrightarrow &&& \\ \{(n, a) : \mathbb{Z} \times A^+ \mid s \wedge (Da_\sigma)(n)\} &\sqsubseteq t && \text{definition of } \sqsubseteq \\ \Leftrightarrow &&& \\ \left(\begin{array}{l} (n, choke) \in t \Rightarrow s \wedge (Dchoke_\sigma)(n) \\ \forall a : A \cdot (n, a) \in t \wedge \neg(s \wedge (Dchoke_\sigma)(n)) \Rightarrow s \wedge (Da_\sigma)(n) \end{array} \right) &&& \text{by definition of } s^* \\ \Leftrightarrow &&& \\ \left(\begin{array}{l} (n, choke) \in t \Rightarrow s^* \wedge (Dchoke_\sigma)(n) \\ \forall a : A \cdot (n, a) \in t \Rightarrow s^* \wedge (Da_\sigma)(n) \end{array} \right) &&& \text{calculus} \\ \Leftrightarrow &&& \\ \forall a : A^+ \cdot (n, a) \in t \Rightarrow s^* \wedge (Da_\sigma)(n) &&& \text{definition of } \varepsilon \text{ below} \\ \Leftrightarrow &&& \\ \varepsilon(t) &\Rightarrow s^* && \end{aligned}$$

where $\varepsilon(t)$ has *signature* the signals a_σ whose compressed derivatives lie in t and *constraint* the fact that those compressed derivatives exhaust t^* ; recalling Definition (46), of *compress*, that yields

$$\varepsilon(t) := \begin{array}{l} \{a_\sigma : \mathbb{S} \mid a \in A^+\} \\ t^* = \cup \{\text{compress}(Da_\sigma) \mid a \in A^+\}. \end{array}$$

The embedding ε is injective (different multitraces differ in at least one event hence give rise to different embeddings) hence, by a simple result of Galois connections, π is surjective. However π is not injective (the inverter and buffer have the same projection), nor (therefore) is ε surjective.

6.3 Timed automata

The *timed automata* [1] used in this section to express circuits have, as a result of our assumptions about environments, a special form. From any stable state (*i.e.* one in which the circuit invariant holds), each

action is possible. From any unstable state, the co-actions that would return it to a stable state with a certain delay k are also accompanied by all actions with delays less than k , each returning it to another unstable state or resulting in choke. The former condition we call *action completeness*, the second *co-action completeness* and a timed automaton having this form we call a ‘circuit automaton’ (defined below).

However we follow the standard notation of timed automata. We consider a vector (integer) clock x which is reset after each transition (action or co-action). Each coordinate c of the vector clock constraint accompanying each transition, $a \xrightarrow{c} T$, lies in the propositional closure of constraints of the form $x \bowtie m$ for constant time $m : \mathbb{Z}$ and operator $\bowtie : \{<, \leq, =, \neq, \geq, >\}$. In fact the clock constraints we consider are severely restricted, as given by the definition below. For vector m of integers having the same length as vector x , we write $x \bowtie m := \forall i \cdot x_i \bowtie m_i$. For details of timed automata in general we refer to [1].

To indicate that action a chokes the circuit under clock constraint c we write $a \xrightarrow{c} \perp$ (which can be thought of, for semantic purposes, as $\text{choke} \xrightarrow{c} \perp$). That notation is slightly unusual because the guard, c , appears after the action, a , which it enables; it is justified by UPPAAL syntax. We adopt the convention $a \rightarrow P := a \xrightarrow{\text{true}} P$.

Example. The naive inverter considered in the previous section can be expressed by this timed automaton

$$\text{Inv} = in \rightarrow out' \xrightarrow{x=2} \text{Inv}$$

that fails to satisfy co-action completeness (although an inverter Inv_1 with unit output delay, $k = 1$, *does* satisfy it). The relevant completion is

$$\begin{aligned} \text{Inv}_2 &= in \longrightarrow J \\ J &= out' \xrightarrow{x=2} \text{Inv}_2 \square in \xrightarrow{x<2} \perp \end{aligned}$$

which can be thought of operationally as follows. The first (input) event occurs at a time determined by the environment at which point the clock x is reset (to 0). In the unstable state J , occurrence of a further input event before $x = 2$ causes a choke; otherwise when $x = 2$ an output event occurs and x is reset. \square

The simple timed automata we consider are defined as follows.

Definition of CA. A *circuit automaton* (CA), with a set A partitioned into actions a_i and co-actions a'_j , and a vector clock x , is expressed by mutual recursion from an initial state T_0

$$(50) \quad \begin{aligned} T_0 &= \square_i a_i \rightarrow T_i \\ &\vdots \\ &= (\square_j a'_j \xrightarrow{x=k} T'_j) \square (\square_i a_i \xrightarrow{x<k} T_i) \\ &\vdots \end{aligned}$$

where the state updates T_i and T'_j reflect the logic-level specification of the circuit and the delay k reflects its signal-level specification. The first block in the mutual recursion contains stable states whilst the second block contains unstable states (each containing at least one co-action). Let the set of such automata be $CA(A)$. \square

Example. The circuit Buf_2 consisting of two unit-delay inverters in sequence:

$$\begin{aligned} Inv_1 &:= in \rightarrow (out' \xrightarrow{x=1} Inv_1) \\ Buf_2 &:= \exists u : \mathbb{S} \cdot Inv_1[u/out'] \wedge Inv_1[u/in]. \end{aligned}$$

As a CA, the simplification of Buf_2 is:

$$(51) \quad \begin{aligned} Buf_2 &= in \rightarrow B \\ B &= out' \xrightarrow{x=2} Buf_2 \sqcap in \xrightarrow{x \leq 2} (out \xrightarrow{x=2} B), \end{aligned}$$

which is of interest because it exemplifies that the sequencing of two one-place ‘buffers’ is not amenable to the same kind of choking as a general two-place buffer (in which the term $(out \xrightarrow{x=2} B)$ is replaced by \perp): sequencing isolates the effect of the input action on the output co-action. \square

A combinational example more typical than the inverter (which has only one input) is the two-input *and* gate of Section 6.5.

6.4 Multitrace semantics

The regularity of circuit automata is reflected in their semantics. For our purposes the following ‘multitrace’ semantics suffices, to be compared with the semantics of general timed automata [2, 1] and general circuits [5].

The *multitrace semantics* of a circuit automaton $T : CA(A)$ is given in the context of an environment $E : \mathbb{E}^\circ(A)$ which schedules the actions of T (recall the definition of $\mathbb{E}^\circ(A)$ from Section 6.2). The semantics has type

$$mt : \mathbb{E}^\circ(A) \times CA(A) \rightarrow \mathbb{E}^\circ(A).$$

To describe it, some notation is convenient.

For nonempty environment $E : \mathbb{E}^\circ(A)$, t_0 denotes the smallest time in E :

$$t_0 := \sqcap \{t : \mathbb{Z} \mid \exists a : A \cdot (t, a) \in E\},$$

and t_{00} denotes its previous value (the environment E is updated in the following recursive definition, so this convention eliminates a parameter that would be explicit in an implementation, like a functional program).

From (50), for the case of stable states,

$$(52) \quad \begin{aligned} & mt(E, \square_i a_i \rightarrow T_i) \\ & := \{(t_0, a_i) \mid (t_0, a_i) \in E \wedge T_i \neq \perp\} \cup \bigcup \{mt(E \setminus \{(t_0, a_i)\}, T_i) \mid (t_0, a_i) \in E \wedge T_i \neq \perp\}. \end{aligned}$$

In particular, from a stable state if the environment offers no events (*i.e.* is empty), then no event occurs. But otherwise each event in E with least time matches one of the inputs of the circuit (by the assumption of action completeness); one of them is scheduled and the circuit moves to the relevant state.

The case of unstable states is more involved:

$$(53) \quad \begin{aligned} & mt(E, (\square_j a'_j \xrightarrow{x=k} T'_j) \square (\square_i a_i \xrightarrow{x < k} T_i)) \\ & := \\ & \{(t_0, a_i), (t_0, \text{choke}) \mid (t_0, a_i) \in E \wedge 0 \leq t_0 - t_{00} < k \wedge T_i = \perp\} \cup \\ & \{(t_0, a_i) \mid (t_0, a_i) \in E \wedge 0 \leq t_0 - t_{00} < k \wedge T_i \neq \perp\} \cup \\ & \bigcup \{mt(E \setminus \{(t_0, a_i)\}, T_i) \mid (t_0, a_i) \in E \wedge 0 \leq t_0 - t_{00} < k \wedge T_i \neq \perp\} \cup \\ & \{(t_0 + k, a'_j) \mid \neg \exists i \cdot (t_0, a_i) \in E \wedge 0 \leq t_0 - t_{00} < k\} \cup \\ & \bigcup \{mt(E \setminus \{(t_0, a_i)\}, T_i) \mid \neg \exists i \cdot (t_0, a_i) \in E \wedge 0 \leq t_0 - t_{00} < k\}. \end{aligned}$$

Thus an action scheduled to choke the circuit (as encoded in the automaton by $T_i = \perp$) suppresses its pending co-actions. Otherwise, a scheduled action transforms the state—as encoded by the automaton—and the environment is updated. In the last case, no action is scheduled before the pending co-actions, which occur; this case includes $E = \{\}$ in which just the pending co-action occurs.

It may be helpful to specialise that to a circuit with a single co-action and an environment that at any time schedules at most one event:

$$(54) \quad \begin{aligned} & mt(E, a' \xrightarrow{x=k} T' \square \square_i a_i \xrightarrow{x < k} T_i) \\ & = \begin{cases} \{(t_0, a_i), (t_0, \text{choke})\} & \text{if } (t_0, a_i) \in E \wedge 0 \leq t_0 - t_{00} < k \wedge T_i = \perp \\ \{(t_0, a_i)\} \cup mt(E \setminus \{(t_0, a_i)\}, T_i) & \text{if } (t_0, a_i) \in E \wedge 0 \leq t_0 - t_{00} < k \wedge T_i \neq \perp \\ \{(t_0 + k, a')\} \cup mt(E \setminus \{(t_0, a_i)\}, T_i) & \text{if } \neg \exists i \cdot (t_0, a_i) \in E \wedge 0 \leq t_0 - t_{00} < k. \end{cases} \end{aligned}$$

Example. Continuing from the previous section the example of the inverter, consider first the simple case of Inv_1 , an inverter with unit output delay. The multitraces are calculated, for any environment $E : \mathbb{E}^\circ(A)$,

$$\begin{aligned} & mt(E, Inv_1) \\ & = \hspace{20em} \text{by (52)} \\ & \{(t_0, in)\} \cup mt(E \setminus \{(t_0, in)\}, out \xrightarrow{x=1} Inv_1) \\ & = \hspace{20em} \text{by (54), last line} \end{aligned}$$

$$\{(t_0, in), (t_0 + 1, out)\} \cup mt(E \setminus \{(t_0, in)\}, Inv_1)$$

which, by calculus, has solution

$$(55) \quad \begin{aligned} mt(E, Inv_1) &= \{(t, in) \mid (t, in) \in E\} \cup \{(t+1, out) \mid (t, in) \in E\} \\ &= E \cup \{(t+1, out) \mid (t, in) \in E\}. \end{aligned}$$

But the identification of the multitraces of Inv_2 , an inverter with output delay 2, requires extra notation. For a subset $F \subseteq \mathbb{Z}$ and $m : \mathbb{Z}$, let $F^{<m} := \{j : F \mid j < m\}$. The predicate $ch(m)$ holds iff m is the time of a choke input: in this case, both $m-1$ and m belong to F

$$ch(m) := \left(\begin{array}{c} m-1 \in F \\ m \in F \end{array} \right).$$

The predicate $fstch(m)$ holds iff m is the time of the first choke input: m is the first member of F for which $ch(m)$ holds:

$$fstch(m) := \left(\begin{array}{c} ch(m) \\ \forall j : F^{<m} . \neg ch(j) \end{array} \right).$$

Then if the environment supplies input events at times $F \subseteq \mathbb{Z}$, F either contains a choke time m or it does not. If it does, the events of the inverter consist of those normal input and output events before $m-1$ with the input event at $m-1$ not matched by output and the input event at m matched by a *choke* event at m ; otherwise the events are as in (55). Thus the circuit has multitrace given by the following conditional, where F is the set of times appearing in the environment E :

$$(56) \quad \begin{aligned} t : \mathbb{E}^\circ(\{in, out\}) \\ fstch(m) \wedge t &= \{(n, in) \mid n \in F^{<m+1}\} \cup \{(n+2, out) \mid n \in F^{<m-2}\} \cup \{(m, choke)\} \\ &\triangleleft \exists m : F \cdot ch(m) \triangleright \\ t &= \{(n, in) \mid n \in F\} \cup \{(n+2, out) \mid n \in F\}. \end{aligned}$$

That, then, is the extension we seek of (55) to a general environment.

Noticing in particular that, for the environment $E := \mathbb{N} \times \{in\}$,

$$\{(0, in), (1, in), (1, choke)\} \subseteq mt(E, Inv_2) \setminus mt(E, Inv_1)$$

whilst

$$\{(0, in), (1, out)\} \subseteq mt(E, Inv_1) \setminus mt(E, Inv_2),$$

we observe that neither of Inv_1 nor Inv_2 refines the other. □

In converting a timed automaton to the form of a CA it is convenient to use laws. Examples appear in Figure 4 and an application of their use appears in Section 6.5.

- (57) \square is associative, commutative and idempotent
- (58) \square is monotone: $T \sqsubseteq T' \Rightarrow a \xrightarrow{c} T \sqsubseteq a \xrightarrow{c} T'$
- (59) \square has zero the choke action: for any co-action b' , $b' \xrightarrow{c} T \square a \xrightarrow{true} \perp = a \xrightarrow{true} \perp$
- (60) for any co-action b' , $a \xrightarrow{true} \perp \sqsubseteq b' \xrightarrow{c} T$
- (61) $a \xrightarrow{true} T = a \rightarrow T$
- (62) $a \xrightarrow{c} T \square a \xrightarrow{c'} T = a \xrightarrow{c \vee c'} T$
- (63) $a \xrightarrow{true} T \square a \xrightarrow{x=k} T' = a \xrightarrow{x \leq k} T \square a \xrightarrow{x=k} T'$

Figure 4: Laws for simple timed automata and circuit automata.

The multitrace semantics of a CA combines several threads of behaviour. Actions are combined with the result that the environment is able to choose from a selection of behaviours. But also co-actions are combined, which corresponds to nondeterminism. We do not consider here the use of priority between events; but to handle that it is convenient to replace the external choice \square with a prioritised version like the operator \triangleleft in CSP [10].

The inverter example illustrates a method for describing circuits: the signal specification is used as the basis for construction of a CA whose multitrace semantics is, by action and co-action completeness, the multitrace semantics of the signal specification extended to general environments. But furthermore the CA serves as the basis for simulation (in our case in UPPAAL [3]). However the inverter is restricted by having only a single action—its input. We now consider a two-input *and* gate, and the possibility that the environment supplies both inputs simultaneously.

6.5 An *and* gate

Recall that in the signal model an *and* gate with double unit delay, functioning in a benign environment, is specified

$$\begin{aligned} a, b, c &: \mathbb{S} \\ c &:= \delta_2(a \wedge b). \end{aligned}$$

Let us assume that the initial states of all signals a , b and c are low (which is stable because, as a consequence, the three signals are simultaneously low before some time and so the gate invariant $c = a \wedge b$ holds and so no output event is required). Determining the output events from the signal specification is slightly more involved than in the inverter example. After initialisation one input may change any number of times without changing output, provided the other input does not change (by definition of *and* gate and our supposed initialisation). However as soon as each input has changed an odd number of times, output changes. And so on.

To infer the event behaviour from the signal specification we use Law (18) (recalling the convention that the differentiation operator binds tighter than the logical connectives):

$$\begin{aligned} Dc & \\ = & \text{definition} \\ D(\delta_2(a \wedge b)) & \\ = & \text{Law (15)} \\ \delta_2 D(a \wedge b) & \\ = & \text{Law (18)} \\ \delta_2((D^+a \wedge b) \vee (a \wedge D^+b) \vee (D^-a \wedge \delta b) \vee (\delta a \wedge D^-b)). & \end{aligned}$$

From initialisation, because it is stable, the circuit is ready to respond to an input event supplied by the environment: either a or b increases after which one of the first two terms applies to place it in an

unstable state. In that state a further input event leads to choke; but otherwise an output event ensues two time units later. Then, with all three signals high, the circuit is again stable and so ready to respond to an input event; one of the last two disjuncts applies as either a or b decreases, which leads to another unstable state. A further input event leads to choke; but otherwise a further output event occurs two time units later. Then the circuit returns to its state just after initialisation.

Such behaviour is conveniently formalised as a CA, A , which is in state $A(i,j,k)$ when signals a , b and c have Boolean values i , j and k respectively. Initially all three signals are low, so the circuit is in state $A = A(0,0,0)$. A state $A(i,j,k)$ is stable iff it satisfies the invariant $k = i \wedge j$. In an unstable state a responding output event is required and so any input event leads immediately to choke, again expressed by the error state \perp . Thus:

$$\begin{aligned}
 A(0,0,0) &= a \rightarrow A(1,0,0) \sqcap b \rightarrow A(0,1,0) \\
 A(1,0,0) &= a \rightarrow A(0,0,0) \sqcap b \rightarrow A(1,1,0) \\
 A(0,1,0) &= a \rightarrow A(1,1,0) \sqcap b \rightarrow A(0,0,0) \\
 A(1,1,1) &= a \rightarrow A(0,1,1) \sqcap b \rightarrow A(1,0,1) \\
 (64) \quad A(1,1,0) &= a \xrightarrow{x<2} \perp \sqcap b \xrightarrow{x<2} \perp \sqcap c \xrightarrow{x=2} A(1,1,1) \\
 A(0,0,1) &= a \xrightarrow{x<2} A(1,0,1) \sqcap b \xrightarrow{x<2} A(0,1,1) \sqcap c \xrightarrow{x=2} A(0,0,0) \\
 A(0,1,1) &= a \xrightarrow{x<2} \perp \sqcap b \xrightarrow{x<2} A(0,0,1) \sqcap c \xrightarrow{x=2} A(0,1,0) \\
 A(1,0,1) &= a \xrightarrow{x<2} A(0,0,1) \sqcap b \xrightarrow{x<2} \perp \sqcap c \xrightarrow{x=2} A(1,0,0).
 \end{aligned}$$

The first four states are stable and the last four unstable. Hence the automaton is often depicted as a cube (which in our case has a state in the centre to represent \perp). In a benign environment, always ready to accept output events and delivering input events not simultaneously but together in alternation with the acceptance of output, A is deterministic, deadlock-free, divergence-free and choke-free.

Now let us consider the possibility that the environment schedules the two input events a and b simultaneously. For comparison, we begin by modelling that as an event ab denoting the single event consisting of the simultaneous occurrence of a and b . But that simple abstraction needs qualification, to reflect reality. In a real circuit it cannot be guaranteed that the events occur truly simultaneously (hence use of the abstraction ‘isochronic fork’ above). So we interpret it to mean that the result is the same regardless of which ordering of the two events occurred; otherwise ab is not a safe abstraction and so not well defined.

If ab occurs in either of the stable states $A(0,0,0)$ or $A(1,1,1)$, a quick check shows its effect to be the same, regardless of the ordering of its constituents; in both cases the result is stable. From the stable state $A(1,0,0)$ the events a then b drive the circuit into the stable state $A(0,1,0)$, whilst in the reverse order they yield choke; and so, in the absence of any way to distinguish the two, the result for ab must be choke. Similarly from $A(0,1,0)$ the results are either choke or $A(1,0,0)$, so the result for ab must again be choke. From the unstable state $A(1,1,0)$ both orderings result in choke, as they do from $A(0,0,1)$. From the unstable states $A(0,1,1)$ and $A(1,0,1)$ one ordering results in choke and the other in an unstable state; so in each case the result must be choke.

For the purpose of action and co-action completeness of the CA, ab is considered the same as any other

action. Thus:

$$\begin{aligned}
A_0(0,0,0) &= a \rightarrow A_0(1,0,0) \sqcap b \rightarrow A_0(0,1,0) \sqcap ab \rightarrow A_0(1,1,0) \\
A_0(1,0,0) &= a \rightarrow A_0(0,0,0) \sqcap b \rightarrow A_0(1,1,0) \sqcap ab \rightarrow \perp \\
A_0(0,1,0) &= a \rightarrow A_0(1,1,0) \sqcap b \rightarrow A_0(0,0,0) \sqcap ab \rightarrow \perp \\
A_0(1,1,1) &= a \rightarrow A_0(0,1,1) \sqcap b \rightarrow A_0(1,0,1) \sqcap ab \rightarrow A_0(0,0,1) \\
(65) \quad A_0(1,1,0) &= a \xrightarrow{x<2} \perp \sqcap b \xrightarrow{x<2} \perp \sqcap ab \xrightarrow{x<2} \perp \sqcap c \xrightarrow{x=2} A_0(1,1,1) \\
A_0(0,0,1) &= a \xrightarrow{x<2} A_0(1,0,1) \sqcap b \xrightarrow{x<2} A_0(0,1,1) \sqcap ab \xrightarrow{x<2} \perp \sqcap c \xrightarrow{x=2} A_0(0,0,0) \\
A_0(0,1,1) &= a \xrightarrow{x<2} \perp \sqcap b \xrightarrow{x<2} A_0(0,0,1) \sqcap ab \xrightarrow{x<2} \perp \sqcap c \xrightarrow{x=2} A_0(0,1,0) \\
A_0(1,0,1) &= a \xrightarrow{x<2} A_0(0,0,1) \sqcap b \xrightarrow{x<2} \perp \sqcap ab \xrightarrow{x<2} \perp \sqcap c \xrightarrow{x=2} A_0(1,0,0).
\end{aligned}$$

However a CA is able to express the more detailed effect of the various orderings of ab that have been abstracted above. So now we return to the more typical method of modelling the effect of ab with only single events, by allowing each single event to transform the circuit to a state from which one alternative is that the other occurs immediately: with guard $x = 0$. That relies on the assumption that an event with guard $x = 0$ is enabled only by the second of a simultaneous pair of events, and not by a ‘quick normal’ event (the results differ, for example, in state $A_1(1,0,0)$ for action a , as indicated in (66)); the latter are enabled via the guard $x > 0$. Of course the result must be consistent with the automaton above in the sense that its multitrace semantics refines that of the previous automaton.

The implementation, using the ‘guard $x = 0$ ’ technique mentioned above, is justified as follows. Consider the event ab in state $A(1,0,0)$. If it occurs as the result of an a event as the first component of ab then it must next allow an immediate b event, as well as a ‘normal’ b event and ‘normal’ a event. Thus:

$$\begin{aligned}
&A(1,0,0) \\
&= \text{modelling assumption} \\
&a \rightarrow A(0,0,0) \sqcap b \xrightarrow{x=0} A(1,1,0) \sqcap b \xrightarrow{x>0} A(1,1,0) \\
&= \text{Law (62)} \\
&a \rightarrow A(0,0,0) \sqcap b \xrightarrow{x \geq 0} A(1,1,0) \\
&= \text{Convention (61)} \\
&a \rightarrow A(0,0,0) \sqcap b \rightarrow A(1,1,0).
\end{aligned}$$

The result of such simplifications, with the explicit encoding of the output delay, is the following CA. We suppose that the unstable states (the last four) satisfy, in preparation for simulation in UPPAAL, the state invariant $x \leq 2$.

$$\begin{aligned}
(66) \quad A_1(0,0,0) &= a \rightarrow A_1(1,0,0) \sqcap b \rightarrow A_1(0,1,0) \\
A_1(1,0,0) &= a \xrightarrow{x=0} \perp \sqcap a \xrightarrow{x>0} A_1(0,0,0) \sqcap b \xrightarrow{x=0} \perp \sqcap b \xrightarrow{x>0} A_1(1,1,0) \\
A_1(0,1,0) &= a \xrightarrow{x=0} \perp \sqcap a \xrightarrow{x>0} A_1(1,1,0) \sqcap b \xrightarrow{x=0} \perp \sqcap b \xrightarrow{x>0} A_1(0,0,0) \\
A_1(1,1,1) &= a \rightarrow A_1(0,1,1) \sqcap b \rightarrow A_1(1,0,1) \\
A_1(1,1,0) &= a \xrightarrow{x=0} A_1(0,1,0) \sqcap a \xrightarrow{x>0} \perp \sqcap b \xrightarrow{x=0} A_1(1,0,0) \sqcap b \xrightarrow{x>0} \perp \\
&\quad \sqcap c \xrightarrow{x=2} A_1(1,1,1) \\
A_1(0,0,1) &= a \xrightarrow{x=0} \perp \sqcap a \xrightarrow{x>0} A_1(1,0,1) \sqcap b \xrightarrow{x=0} \perp \sqcap b \xrightarrow{x>0} A_1(0,1,1) \\
&\quad \sqcap c \xrightarrow{x=2} A_1(0,0,0) \\
A_1(0,1,1) &= a \rightarrow \perp \sqcap b \xrightarrow{x=0} \perp \sqcap b \xrightarrow{x>0} A_1(0,0,1) \sqcap c \xrightarrow{x=2} A_1(0,1,0) \\
A_1(1,0,1) &= a \xrightarrow{x=0} \perp \sqcap a \xrightarrow{x>0} A_1(0,0,1) \sqcap b \rightarrow \perp \sqcap c \xrightarrow{x=2} A_1(1,0,0)
\end{aligned}$$

Consider, for example, the b transition from $A_1(0,1,1)$ with guard $x = 0$. Previously an a event has occurred from state $A_1(1,1,1)$ (recall the assumption of the environment that the guard $x = 0$ is satisfied only by the second event in a concurrent pair) and now the accompanying b event is possible. Were it to succeed, it would drive the state to $A_1(0,0,1)$ which indeed preserves instability. But overall, it would mean that ab drives the circuit from state $A_1(1,1,1)$ to state $A_1(0,0,1)$ which is from a stable state to an unstable one and hence would enable an incorrect output event. So the result must be choke.

Thus by design the CA A_1 chokes in the same situations as A_0 and so, in terms of multitrace semantics, $A_0 = A_1$. However a simple modification provides a strict refinement. For example in both $A_0(0,1,1)$ and $A_1(0,1,1)$ the simultaneous event ab results in choke because one ordering of the two events does so. But the two orderings can be separated, so that just one leads to choke, by revising (66) along the lines

$$\begin{aligned}
A_2(0,1,1) &= a \rightarrow \perp \sqcap b \rightarrow A_2(0,0,1) \sqcap c \xrightarrow{x=2} A_2(0,1,0) \\
A_2(0,0,1) &= a \xrightarrow{x=0} A_2(1,0,1) \sqcap \dots
\end{aligned}$$

Then, since A_2 chokes in fewer situations than A_1 , in the multitrace semantics $A_1 \sqsubset A_2$. In fact the equality $A_0 = A_1$ and the refinement $A_1 \sqsubset A_2$ are more easily established, rather than calculating multitraces directly, by using laws sound in the multitrace semantics.

The authors have executed (66) (and various variations of it) using UPPAAL's graphical interface, replacing the state invariant $x \leq 2$ by making each unstable state *urgent*. Under the assumption that the environment supplies only one input event when the circuit is stable, but may change both simultaneously in the same direction when it is unstable, the CA is deterministic and deadlock free.

We have also used the method described above (of constructing an UPPAAL automaton starting from the signal specification and using action and co-action completeness) for the other combinational circuits in this paper (as well as several others, including sequential circuits: a C-element, T-element, arbiter and dual-rail false variable [15]) to obtain simulations in a reasonably systematic manner.

7 Conclusion

The kind of hazard analysis studied in undergraduate courses in Computer Science, and in the corresponding texts (*e.g.* [7, 13]), is restricted to informal reasoning about benign environments: they never block output and provide input only when convenient for the circuit. What are the reasons for restricting to benign environments? Can those be weakened and simple rigorous methods still used to analyse transient behaviour in more realistic environments? Can such analysis be supported by simple rigorous reasoning and, if so, to what advantage and at what expense? Is there a sound methodology that can be followed that leads stepwise from simple specification to simulation?

In this paper we have taken an approach sufficient to answer those questions. The restriction to benign environments is shown not to be compositional, and so to be useful only for very simple circuits in isolation. It must ultimately be a matter of taste how much formalism is used in teaching. But the laws of the signal calculus considered here are no more complex than those of predicate calculus. In particular they are readily automated. And the simple multitrace semantics appears to be sufficient for demonstrating soundness of the manipulation of combinational circuits as considered here.

The form of the results concerning hazardous behaviour of combinational circuits is rewarding. For example, the usual hazard analysis of a conditional gate is itself expressed as a conditional. Thus the two cases (of rising or falling input) considered when analysing hazards informally are here combined into one result, which stems from a systematic algebraic analysis.

The various models are sufficiently simple to be related via simple functional projections (rather than say more general Galois connections), here the various π 's.

However the major contribution has been the method for proceeding systematically and in a sound manner from logic to signal to circuit automaton and hence to simulation of combinational circuits in UP-PAAL. The range of examples on which we have used this method is so far limited, but encouraging. For combinational circuits the multitrace semantics suffices, but for sequential circuits a more detailed semantics reflecting input history is needed. Fortunately the timed automata which arise still have a restricted form ('elastic timed automata'), sufficient to enable them to be replaced by deterministic semantic equivalents, their 'determinisation' [17].

7.1 Further work

We have lacked time to extend the method proposed here to sequential circuits. In order to do so, our simple multitrace semantics must be extended to include action history. Fortunately that is established territory, using the process-algebra and traces approach of the Eindhoven/Dutch school (van de Snepscheut, Udding, Ebergen, Schols, Rem, Verhoof, van Birkel) and of Dill. The semantic models, each with the choke-refinement ordering, must be related (by Galois connection) thus ensuring correctness of the move between the various formalisms for circuits, timed automata and in particular UPPAAL.

Process algebra has been very much just behind the scenes in Section 6. A compelling piece of work would be to make the connection explicit by using say CSP to express and simplify a combination of CAs, using the combinator \parallel of parallel composition, and to simulate the result in the CSP model-checker, FDR. It would then be of interest to compile a table comparing the results of the UPPAAL simulations resulting from application of our method, with results obtained using other (for example FDR) simulations.

A more extended project consists of performing the step—in which the simultaneous event ab is replaced by its single constituents—by ‘action refinement’, using a formalism of which this is a particular application. Indeed this application appear to be an interesting test of any formalism of action refinement.

Finally, important remaining work lies in the clarification and the connection of the current approach with related work, including:

Dill’s success/failures structures [5]. We have chosen to express event semantics using a parameter E for the environment. But that can of course be related to a form more standard in process algebra, by considering traces that lead to choke (Dill’s ‘failures’) and those that do not (his ‘successes’);

timed process algebra [9, 12, 16, 15] and the connection between choke-refinement and mirroring [5, 18, 19] in the multitrace semantics;

Kahn processes and the reactive processes of Joesphs [6, 8] (because our ‘processes’, by being prepared always to engage in an action, are reactive);

models having (many) more than two signal values; see for example the elegant approach in [14].

References

- [1] R. Alur. Timed automata. *Proc. CAV’99*, Springer LNCS, **1633**:8–22, 1999.
- [2] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, **126**:183–235, 1994.
- [3] G. Behrmann, A. David and K. G. Larsen. <http://www.uppaal.com>.
- [4] R. S. Bird. *Introduction to Functional Programming using Haskell*, second edition, Prentice-Hall International, 1998.
- [5] D. L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed Independent Circuits*. (AMS Distinguished Dissertation.) M.I.T. Press, 1988.
- [6] S. A. Edwards and O. Tardieu. Deterministic receptive processes are Kahn processes. *Proceedings of the 3rd International Conference on Formal Methods and Models for Codesign (MEMOCODE)*. Verona, Italy, July 2005.

- [7] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2006.
- [8] M. B. Josephs and J. T. Udding. An overview of D-I algebra. In: T. N. Mudge, V. Milutinovic and L. Hunter, editors, *System Sciences (HICCS), Proceedings of the 26th annual Hawaii International Conference*, 329–338, IEEE CS Press, 1993.
- [9] J. Ouaknine and S. A. Schneider. Timed CSP: a retrospective. Proceedings of the workshop *Essays on Algebraic Process Calculi, (APC 25)*. *Electronic Notes in Theoretical Computer Science*, **162**:273–276, September 2006.
- [10] A. W. Roscoe. *The Theory and Practice of Concurrency*, Prentice-Hall, 1998.
- [11] C. Ratzko and J. W. Sanders. A calculus of signals. In *Proceedings of ICECS2k, The 7th IEEE International Conference on Electronics, Circuits and Systems*, volume 1:399–402, IEEE Computer Society, 2000. (An expanded version is available as Technical Report PRG-2000-06 of the Oxford University Computing Laboratory.)
- [12] S. A. Schneider. *Concurrent and Real-time Systems: The CSP Approach*. Wiley, 1999.
- [13] A. Tanenbaum. *Structured Computer Organization, 5th edition*. Addison-Wesley/Prentice-Hall, 2006.
- [14] S. Thompson and A. Mycroft. Abstract interpretation of combinational asynchronous circuits. *Science of Computing Programming*, **64**(1):166–183, 2007.
- [15] X. Wang, M. Z. Kwiatkowska, G. Theodoropoulos and Q. Zhang. Towards a Unifying CSP approach to Hierarchical Verification of Asynchronous Hardware. *Electronic Notes in Theoretical Computer Science*. **128**(6):231–246, 2005.
- [16] X. Wang and M. Z. Kwiatkowska. On process-algebraic verification of asynchronous circuits. *Fundamenta Informaticae*, **80**(1-3): 283–310, 2007.
- [17] Wang Xu and J. W. Sanders. Determinising elastic timed automata having silent transitions. In preparation.
- [18] B. Zhou, T. Yoneda and B-H. Schlingloff. Conformance and mirroring for timed asynchronous circuits. *Proc. of Asia and South Pacific Design Automation Conference*, 341–346, 2001.
- [19] B. Zhou, T. Yoneda and C. Myers. Framework of timed trace theoretic verification revisited. *IEICE Transactions*, **E85-D**(10):1595–1604, 2002.