

# Abstract Interpretation of Combinational Asynchronous Circuits

Sarah Thompson and Alan Mycroft

Computer Laboratory, University of Cambridge  
William Gates Building, JJ Thompson Avenue,  
Cambridge, CB3 0FD, UK  
{Sarah.Thompson,Alan.Mycroft}@cl.cam.ac.uk

**Abstract.** A technique, based upon abstract interpretation, is presented that allows general gate-level combinational asynchronous circuits with uncertain delay characteristics to be reasoned about. Our approach is particularly suited to the simulation and model checking of circuits where the identification of possible glitch states (static and dynamic hazards) is required.

A hierarchy of alternative abstractions linked by Galois connections is presented, each offering varying tradeoffs between accuracy and complexity. Many of these abstract domains resemble extended, multi-value logics: *transitional logics* that include extra values representing transitions as well as steady states, and *static/clean* logics that include the values S and C representing ‘unknown but fixed for all time’ and ‘can never glitch’ respectively.

## 1 Introduction

Most contemporary design approaches assume an underlying *synchronous* paradigm, where a single global signal drives the clock inputs of every flip flop in the circuit. As a consequence, nearly all synthesis, simulation and model checking tools assume synchronous semantics. Designs in which this rule is relaxed are generally termed *asynchronous circuits*.

In a synchronous model, glitches (also known as static and dynamic hazards) do not cause problems unless they occur on a wire used as a clock input; with purely synchronous design rules<sup>1</sup> this can not occur. However, such safety restrictions are *not* enforced by the semantics of either Verilog or VHDL – it is quite easy, deliberately or otherwise, to introduce unsafe logic into a clock path.

We present a technique, based upon abstract interpretation [1, 2], that allows the glitch states of asynchronous circuits to be identified and reasoned about. The approach taken involves a family of extended, multi-value *transitional* logics with an underlying dense continuous time model, and has applications in synthesis, simulation and model checking.

Our logics are extended with extra values that capture transitions as well as steady states, with an ability to distinguish *clean*, glitch-free signals from

---

<sup>1</sup> Exactly one global clock net driving the clock inputs of all flip flops in the circuit.

*dirty*, potentially glitchy ones. As a motivating example, consider the circuits represented by the expressions  $(a \wedge c) \vee (\neg a \wedge b) \vee (b \wedge c)$  and  $(a \wedge c) \vee (\neg a \wedge b)$ . With respect to steady-state values for  $a$ ,  $b$  and  $c$ , both circuits would appear to be identical, with the latter representing a circuit that might result from naïve optimisation of the former. Our technique can straightforwardly illustrate differences in their dynamic behaviour, however. Consider the critical case  $a = \uparrow_0$  and  $b = c = \top_0$ , representing  $b$  and  $c$  being wired to *true* for all time, and a clean transition from *false* to *true* on  $a$  (this notation is defined fully in Section 3):

$(a \wedge c) \vee (\neg a \wedge b) \vee (b \wedge c)$	$(a \wedge c) \vee (\neg a \wedge b)$
$= (\uparrow_0 \wedge \top_0) \vee (\neg \uparrow_0 \wedge \top_0) \vee (\top_0 \wedge \top_0)$	$= (\uparrow_0 \wedge \top_0) \vee (\neg \uparrow_0 \wedge \top_0)$
$= \uparrow_0 \vee \downarrow_0 \vee \top_0$	$= \uparrow_0 \vee \downarrow_0$
$= \top_0$	$= \top_{0..1}$

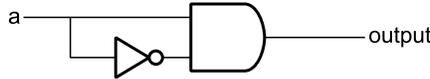
The result  $\top_0$  may be interpreted as ‘*true* for all time, with no glitches’. However,  $\top_{0..1}$  represents ‘*true* with zero or one glitches’, clearly demonstrating the poorer dynamic behaviour of the smaller circuit.

### 1.1 Hardware Components

In this paper we consider four<sup>2</sup> basic building blocks: (perfect—zero delay) AND-gates, (perfect) NOT-gates, delay elements (whose delays may depend on time, and environmental factors like temperature, and thus are non-deterministic in a formal sense), and *inertial delay* elements. The difference between an ordinary delay and an inertial delay is that in the former the number of transitions on its input and output are equal, but in the latter a short-duration pulse from high-to-low and back (or vice versa) may be removed entirely from the output.

Of course, real circuits are not so general, in particular no practically realisable circuit of non-zero size can have zero-delay. Hence real-life circuits all correspond to combinations of the above gates with some form of delay element. For the point of designing synchronous hardware all that matters is the maximum delay which can occur from a circuit, so the exact positioning of the delays is often of little importance. When circuits are used asynchronously (e.g. for designing self-timed circuits without a global clock or, more prosaically, when their output is being used to gate a clock signal locally) then their glitch behaviour is often critically important. This leads to two models (the *delay-insensitive* (DI) and *speed-independent* (SI) models) of real hardware. In the SI model logic elements may have delays, but wires do not; in the DI model both logic elements and wires have associated delay. One well-known fact about DI models is that it is impossible to have an *isochronic fork*, whereby the transitions in output from

<sup>2</sup> A perfect OR-gate can be constructed from perfect AND- and NOT-gates using de Morgan’s law.



**Fig. 1.** The circuit  $a \wedge \neg a$

a given gate will arrive delayed contemporaneously at two other gates. Reasoning in the DI model has become much more important recently as wire delays (e.g. due to routing) have become dominant over single-gate element delays in modern VLSI technologies [11].

Ordinary circuits may be embedded in our model as follows. In the SI model each physical logic gate at the hardware level is seen as a perfect logic gate whose *output* is then passed through a delay element. In the DI model, each physical logic gate is seen as a perfect logic gate whose *input(s)* first pass through separate delays. In essence, the SI and DI models of a circuit are translations of a physical circuit into idealised circuits composed solely of our four perfect elements.

Now consider the circuit in Fig. 1. Seen as a perfect logic element, its output is always *false* regardless of the value of its input signal. Seen as an SI circuit (i.e. delays on the output of the AND and NOT), given an input  $F_1$  which starts at *false* then transitions to *true* and back, the circuit will be *false* at all times except (possibly) for a small period just after the rising edge of the input, when the upper AND-input will already be *true*, but before the delayed NOT-output has yet become *false*. Thus the output is  $F_{0|1}$  if we assume an inertial delay and  $F_1$  if we assume a non-inertial delay<sup>3</sup>.

In contrast, in the DI model, the separate delays on both inputs to the AND-gate mean that the same input signal  $F_1$  may result in small positive pulses on both the rising and falling edge of the input; thus the output is described as  $F_{0|1|2}$ . It is important to note that any of these three possible outputs may occur; delays may vary with time, and can also differ on whether an input signal is rising or falling.

Our abstract interpretation framework enables us to formally deduce the above behaviours of the circuit shown in Fig. 1. Our reasoning is *correct*, because of the abstract interpretation framework. In some situations our reasoning is also *complete* in that all abstractly-predicted behaviours may be made to happen by choosing suitable delay functions for the delay elements. For example, in the DI model, our abstraction of the above circuit maps abstract signal  $F_1$  onto  $F_{0|1|2}$ , but the SI model cannot produce  $F_2$  however (positive) delay intervals are chosen.

---

<sup>3</sup> This argument assumes positive delays; at times later in the paper we also allow (non-physically realisable) delays by negative time.

## 1.2 Paper Structure

In Section 2 we define a concrete domain that models signals as (possibly non-deterministic) functions from time to the Boolean values. Section 3 describes the most accurate (though complex) of our abstract domains; Sections 5 and 6 show how this can be further abstracted. Section 4 defines the operators necessary to model circuits, Section 4.1 discusses soundness and completeness of these operators. Refinement and equivalence relations are discussed in Section 7.

## 2 Concrete Domain

**Definition 1.** Concrete time  $\mathbb{R}$  is continuous, linear and dense, having no beginning or end.

**Definition 2.** A signal is a total function in  $\mathbb{S} : \mathbb{R} \rightarrow \{0, 1\}$  from concrete time to the Boolean values. More precisely, we restrict  $\mathbb{S}$  to those functions that are finitely piecewise constant<sup>4</sup>, i.e. there exists  $\{k_1, \dots, k_n\}$  which uniquely determines and is determined by a signal  $s \in \mathbb{S}$  such that

$$\begin{aligned} s(k_i) &= \neg s(k_{i+1}) & \forall 1 \leq i < n; \\ s(x) &= s(k_i) & \forall k_i \leq x < k_{i+1}; \\ s(-\infty) &= s(x) = \neg s(k_1) & \forall x < k_1; \\ s(+\infty) &= s(x) = s(k_n) & \forall x \geq k_n. \end{aligned}$$

The function  $\Psi s \stackrel{\text{def}}{=} \{k_1, \dots, k_n\}$  represents the bijection which returns the set of times at which signal  $s$  has transitions;  $|\Psi s|$  represents the total number,  $n$ , of transitions made by  $s$ . As a further notational convenience, we denote the values of  $s$  at the beginning and end of time respectively as  $s(-\infty)$  and  $s(+\infty)$ .

We model nondeterministic signals as members of the set  $\wp(\mathbb{S})$ ; e.g. delaying signal  $s$  by time  $\delta$ , where  $\delta_{\min} \leq \delta \leq \delta_{\max}$ , gives  $\{\lambda\tau.s(\tau-\delta) \mid \delta_{\min} \leq \delta \leq \delta_{\max}\}$ .

## 3 Abstract Domain

### 3.1 Deterministic Traces

**Definition 3.** A deterministic trace  $t \in \mathbb{T}$  characterises a deterministic signal  $s \in \mathbb{S}$ , retaining the transitions but abstracting away the times at which they occur. Traces are denoted as finite lists of Boolean values bounded by angle brackets ( $\langle \dots \rangle$ ), and must contain at least one element – the empty trace  $\langle \rangle$  is not syntactically valid.

<sup>4</sup> Note that we do not consider signals that contain an infinite number of transitions, e.g. clocks that oscillate for all time. We can, however, reason about such signals by ‘windowing’ them within finite intervals (windows)  $[p, q]$  of  $\mathbb{R}$ , resulting in signals that are themselves finitely piecewise constant.

---

$F_0$	The trace $\langle 0 \rangle$ that is 0 for all time.
$F_1$	The trace $\langle 0, 1, 0 \rangle$ that has 0 at the beginning and end, containing exactly one pulse.
$F_2$	The trace $\langle 0, 1, 0, 1, 0 \rangle$ that begins and ends with 0, containing exactly two pulses.
$F_n$	The trace $\langle 0, 1_1, 0, 1_2, 0, \dots, 0, 1_n, 0 \rangle$ that begins and ends with 0, containing exactly $n$ positive-going pulses.
$T_0$	The trace $\langle 1 \rangle$ that is 1 for all time.
$T_n$	The trace $\langle 1, 0_1, 1, 0_2, 1, \dots, 1, 0_n, 1 \rangle$ that begins and ends with 1, containing exactly $n$ negative-going pulses.
$\uparrow_0$	The trace $\langle 0, 1 \rangle$ that cleanly transitions from 0 to 1.
$\uparrow_n$	The trace $\langle 0, 1_1, 0, \dots, 0, 1_n, 0, 1 \rangle$ that transitions from 0 to 1 through exactly $n$ intervening cycles.
$\downarrow_0$	The trace $\langle 1, 0 \rangle$ that cleanly transitions from 1 to 0.
$\downarrow_n$	The trace $\langle 1, 0_1, 1, \dots, 1, 0_n, 1, 0 \rangle$ that transitions from 1 to 0 through exactly $n$ intervening cycles.

---

**Table 1.** Shorthand Notation: Deterministic Traces

A *singleton trace*, denoted  $\langle 0 \rangle$  or  $\langle 1 \rangle$ , represents a signal that remains at 0 or 1 respectively for all time. For traces with two or more elements, e.g.  $\langle a, \dots, b \rangle$ ,  $a$  is the value at the beginning of time and  $b$  is the value at the end of time.

The trace  $\langle 0, 1, 0 \rangle$  represents a signal that at the start of time takes the value 0, then at some later time switches cleanly to 1, then back to 0 again before the end of time. The instants at which these transitions occur are undefined, although their time order must be preserved.

Values within traces may be discriminated only by their transitions. Therefore, the trace  $\langle 0, 0, 0, 0, 1, 1, 1 \rangle$  is equivalent to the trace  $\langle 0, 1 \rangle$ . It follows from this that all traces may be reduced to a form that resembles an alternating sequence  $\langle \dots, 0, 1, 0, 1, 0, 1, \dots \rangle$ . Any such sequence can be completely characterised by its start and end values, along with the number of intervening full cycles<sup>5</sup>. A convenient shorthand notation that takes advantage of this is defined in Table 1.

### 3.2 Nondeterministic Traces

Following the approach taken in Section 2, we represent nondeterministic traces  $\hat{t} \in \wp(\mathbb{T})$  as sets of deterministic traces<sup>6</sup>.

<sup>5</sup> It is of course also possible to represent traces completely in terms of their first (or last) element and their length. However, the representation chosen here turns out to be more convenient, e.g. comparing  $\uparrow_0$  with  $\uparrow_4$  makes it immediately obvious that both represent traces that eventually transition from 0 to 1, with  $\uparrow_0$  being ‘cleaner’ than  $\uparrow_4$ . The utility of this approach will become clear later.

<sup>6</sup> We adopt the convention that  $t$  and  $\hat{t}$  are separate variables that range over  $\mathbb{T}$  and  $\wp(\mathbb{T})$  respectively.

The need for this extra structure is demonstrated by the following example. Let us attempt to specify the meaning of the expression  $\langle 0, 1 \rangle \wedge \neg \langle 0, 1 \rangle$ , which represents the effect of feeding a clean transition from 0 to 1 to the  $a$  input of the circuit shown in Fig. 1. The  $\neg$  can be evaluated trivially, giving  $\langle 0, 1 \rangle \wedge \langle 1, 0 \rangle$ . At first sight, it may appear that the resulting trace should be  $\langle 0, 0 \rangle$  or just  $\langle 0 \rangle$ . This would be the case if certain constraints on the exact times of the transitions of the  $\langle 1, 0 \rangle$  and  $\langle 0, 1 \rangle$  traces were met, but it is not sufficient to cope with all possibilities. If  $\langle 1, 0 \rangle$  transitions before  $\langle 0, 1 \rangle$ , then the result is indeed  $\langle 0 \rangle$ . Should the transitions occur in the opposite order, the result is  $\langle 0, 1, 0 \rangle$ . Formally,

$$\{\langle 0, 1 \rangle\} \wedge \neg \{\langle 0, 1 \rangle\} = \{\langle 0, 1 \rangle\} \wedge \{\langle 1, 0 \rangle\} = \{\langle 0 \rangle\} \cup \{\langle 0, 1, 0 \rangle\} = \{\langle 0 \rangle, \langle 0, 1, 0 \rangle\}$$

**Definition 4.** Where  $\hat{t} \in \wp(\mathbb{T})$  and  $\hat{u} \in \wp(\mathbb{T})$ , the nondeterministic choice  $\hat{t} \mid \hat{u}$  is synonymous with  $\hat{t} \cup \hat{u}$ . For notational compactness, we alternatively allow either or both of the arguments of  $\mid$  to range over  $\mathbb{T}$ , e.g. where  $t \in \mathbb{T}$ , the expression  $t \mid \hat{u}$  is equivalent to  $\{t\} \mid \hat{u}$ .

The ‘ $\mid$ ’ operator allows the above equation to be expressed more compactly as follows:

$$\langle 0, 1 \rangle \wedge \neg \langle 0, 1 \rangle = \langle 0, 1 \rangle \wedge \langle 1, 0 \rangle = \langle 0 \rangle \mid \langle 0, 1, 0 \rangle$$

Using the shorthand notation, this may equivalently be written as:

$$\downarrow_0 \wedge \neg \downarrow_0 = \downarrow_0 \wedge \uparrow_0 = F_0 \mid F_1$$

**Definition 5.** Letting  $X$  range over  $\{\mathbb{T}, F, \uparrow, \downarrow\}$ ,

$$X_{m..n} \stackrel{\text{def}}{=} \bigcup_{i=m}^n \{X_i\} \quad X_{a_1 \mid \dots \mid a_n} \stackrel{\text{def}}{=} X_{a_1} \mid \dots \mid X_{a_n}$$

For example,  $F_0 \mid F_1$  may equivalently be written as  $F_{0 \mid 1}$ , and rather than fully enumerating a long list of alternate pulse counts of the form  $F_{m \mid m+1 \mid \dots \mid n-1 \mid n}$ , the preferred notation  $F_{m..n}$  may be used instead. These notations may be combined, e.g.  $F_{0 \mid 3 \mid 5..7 \mid 10..12}$ .

Nondeterministic choice obeys all the laws of set union, e.g.

$$a \mid a = a \quad a \mid b = b \mid a \quad a \mid (b \mid c) = (a \mid b) \mid c = a \mid b \mid c$$

From this, various subscript laws follow, e.g.

$$X_{a \mid a} = X_a \quad X_{a..a} = X_a$$

$$X_{a..b} \mid X_{c..d} = \begin{cases} X_{\min(a,c).. \max(b,d)} & \text{if } c \leq b \wedge a \leq d; \\ X_{a..b \mid c..d} & \text{otherwise.} \end{cases}$$

**Definition 6.** It is convenient to name the following least upper bounds w.r.t.  $\langle \wp(\mathbb{T}), \subseteq \rangle$ :

$$F_\star \stackrel{\text{def}}{=} \bigcup_{n \in \mathbb{N}} \{F_n\} \quad T_\star \stackrel{\text{def}}{=} \bigcup_{n \in \mathbb{N}} \{T_n\} \quad \uparrow_\star \stackrel{\text{def}}{=} \bigcup_{n \in \mathbb{N}} \{\uparrow_n\} \quad \downarrow_\star \stackrel{\text{def}}{=} \bigcup_{n \in \mathbb{N}} \{\downarrow_n\}$$

$$\star \stackrel{\text{def}}{=} F_\star \cup T_\star \cup \uparrow_\star \cup \downarrow_\star$$

### 3.3 Galois Connection

**Definition 7.** Given a deterministic concrete signal  $s \in \mathbb{S}$ , the abstraction function  $\beta : \mathbb{S} \rightarrow \mathbb{T}$  returns the corresponding deterministic trace:

$$\begin{aligned} \beta s &\stackrel{\text{def}}{=} \langle s(-\infty), s(k_1), \dots, s(k_n) \rangle && \text{where } \{k_1, \dots, k_n\} = \Psi s \\ &= \langle s(-\infty), \neg s(-\infty), s(-\infty), \neg s(-\infty), \dots \rangle \end{aligned}$$

Note that  $\beta s$  has exactly  $1 + |\Psi s|$  elements.

**Definition 8.** The abstraction function  $\alpha : \wp(\mathbb{S}) \rightarrow \wp(\mathbb{T})$  and concretisation function  $\gamma : \wp(\mathbb{T}) \rightarrow \wp(\mathbb{S})$  are defined as follows:

$$\alpha \hat{s} \stackrel{\text{def}}{=} \{\beta s \mid s \in \hat{s}\} \qquad \gamma \hat{t} \stackrel{\text{def}}{=} \{s \in \mathbb{S} \mid \beta s \in \hat{t}\}$$

**Definition 9.** Letting  $\sim : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{B}$  represent the equivalence relation  $s_1 \sim s_2 \Leftrightarrow \beta s_1 = \beta s_2$ , the set  $\mathbb{S}^\# \stackrel{\text{def}}{=} \mathbb{S}/\sim$  is the set of equivalence classes in  $\mathbb{S}$  with respect to  $\sim$ . The set  $[s] \stackrel{\text{def}}{=} \{s' \in \mathbb{S} \mid \beta s = \beta s'\}$  represents, for any  $s \in \mathbb{S}$ , the equivalence class containing that element.

Note that  $\mathbb{S}^\#$  is isomorphic with  $\mathbb{T}$ .

**Theorem 1.** Together, the adjoint functions  $\langle \alpha, \gamma \rangle$  form a Galois connection between  $\wp(\mathbb{S})$  and  $\wp(\mathbb{T})$ . Following Cousot & Cousot [2], Theorem 5.3.0.4 and Corollary 5.3.0.5, pp. 273, it is sufficient to show that  $\alpha \circ \gamma(\hat{x}) \supseteq \hat{x}$  and  $\gamma \circ \alpha(\hat{x}) \supseteq \hat{x}$ . We choose to prove instead the slightly stronger  $\alpha \circ \gamma(\hat{x}) = \hat{x}$ , and since the ordering relations on  $\wp(\mathbb{S})$  and  $\wp(\mathbb{T})$  are subset inclusion, we write  $\supseteq$  rather than  $\supseteq$ . Proof; letting  $\hat{x} = \{x_1, \dots, x_n\}$

1.  $\alpha \circ \gamma(\hat{x}) = \alpha\{s \in \mathbb{S} \mid \beta s \in \hat{x}\} = \{\beta s' \mid s' \in \{s \in \mathbb{S} \mid \beta s \in \hat{x}\}\} = \{\beta s' \mid \beta s \in \hat{x}\} = \hat{x}$ .
2.  $\gamma \circ \alpha(\hat{x}) = \gamma \circ \alpha\{x_1, \dots, x_n\} = \gamma\{\beta x_1, \dots, \beta x_n\} = \gamma\{\beta x_1\} \cup \dots \cup \gamma\{\beta x_n\} = \{s \in \mathbb{S} \mid \beta s = \beta\{x_1\}\} \cup \dots \cup \{s \in \mathbb{S} \mid \beta s = \beta\{x_n\}\} = [x_1] \cup \dots \cup [x_n] \supseteq \{x_1, \dots, x_n\} = \hat{x}$ .

## 4 Circuits

**Definition 10.** Circuits are modeled by composing four basic operators: zero delay ‘and’  $\wedge$ , zero delay ‘not’  $\neg$ , transmission line delay  $\Delta$  and inertial delay  $\square$ , which are defined on the concrete domain as follows:

$$\begin{aligned} \wedge &\stackrel{\text{def}}{=} \lambda(\hat{s}_1, \hat{s}_2). \{\lambda \tau. s_1(\tau) \wedge s_2(\tau) \mid s_1 \in \hat{s}_1 \wedge s_2 \in \hat{s}_2\} \\ \neg &\stackrel{\text{def}}{=} \lambda \hat{s}. \{\lambda \tau. \neg s(\tau) \mid s \in \hat{s}\} \\ \Delta &\stackrel{\text{def}}{=} \gamma \circ \alpha \end{aligned}$$

$\neg^\#$		$\wedge^\#$	$F_0$	$F_n$	$T_0$	$T_n$	$\uparrow_0$	$\uparrow_n$	$\downarrow_0$	$\downarrow_n$
$F_0$	$T_0$	$F_0$	$F_0$	$F_0$	$F_0$	$F_0$	$F_0$	$F_0$	$F_0$	$F_0$
$F_n$	$T_n$	$F_m$	$F_0$	$F_{0..m+n-1}$	$F_m$	$F_{0..m+n}$	$F_{0..m}$	$F_{0..m+n}$	$F_{0..m}$	$F_{0..m+n}$
$T_0$	$F_0$	$T_0$	$F_0$	$F_n$	$T_0$	$T_n$	$\uparrow_0$	$\uparrow_n$	$\downarrow_0$	$\downarrow_n$
$T_n$	$F_n$	$T_m$	$F_0$	$F_{0..m+n}$	$T_m$	$T_{1..m+n}$	$\uparrow_{0..m}$	$\uparrow_{0..m+n}$	$\downarrow_m$	$\downarrow_{0..m+n}$
$\uparrow_0$	$\downarrow_0$	$\uparrow_0$	$F_0$	$F_{0..n}$	$\uparrow_0$	$\uparrow_{0..n}$	$\uparrow_0$	$\uparrow_{0..n}$	$F_{0..1}$	$F_{0..n+1}$
$\uparrow_n$	$\downarrow_n$	$\uparrow_m$	$F_0$	$F_{0..m+n}$	$\uparrow_m$	$\uparrow_{0..m+n}$	$\uparrow_{0..m}$	$\uparrow_{0..m+n}$	$F_{0..m+1}$	$F_{0..m+n+1}$
$\downarrow_0$	$\uparrow_0$	$\downarrow_0$	$F_0$	$F_{0..n}$	$\downarrow_0$	$\downarrow_n$	$F_{0..1}$	$F_{0..n+1}$	$\downarrow_0$	$\downarrow_{0..n}$
$\downarrow_n$	$\uparrow_n$	$\downarrow_m$	$F_0$	$F_{0..m+n}$	$\downarrow_m$	$\downarrow_{0..m+n}$	$F_{0..m+1}$	$F_{0..m+n+1}$	$\downarrow_{0..m}$	$\downarrow_{0..m+n}$

where  $m > 0, n > 0$ .

**Table 2.** Boolean Functions on Traces

$$\square \stackrel{\text{def}}{=} \gamma \circ \square^\# \circ \alpha$$

Their abstract counterparts are defined as follows:

$$\wedge^\# \stackrel{\text{def}}{=} \alpha \circ \wedge \circ \langle \gamma, \gamma \rangle$$

$$\neg^\# \stackrel{\text{def}}{=} \alpha \circ \neg \circ \gamma$$

$$\Delta^\# \stackrel{\text{def}}{=} \lambda x.x$$

$$\square^\# \stackrel{\text{def}}{=} \lambda \hat{t}. \{t \in \mathbb{T} \mid \exists t' \in \hat{t}. \text{Val}(t) = \text{Val}(t') \wedge \text{Subs}(t) \leq \text{Subs}(t')\}$$

where  $\text{Val} : \mathbb{T} \rightarrow \{\mathbf{F}, \mathbf{T}, \uparrow, \downarrow\}$  and  $\text{Subs} : \mathbb{T} \rightarrow \mathbb{N}$  are defined as follows:

$$\text{Val}(X_n) \stackrel{\text{def}}{=} X \qquad \text{Subs}(X_n) \stackrel{\text{def}}{=} n$$

*And.* The function  $\wedge : \wp(\mathbb{S}) \times \wp(\mathbb{S}) \rightarrow \wp(\mathbb{S})$  represents a perfect zero-delay AND gate. Its abstract counterpart,  $\wedge^\# : \wp(\mathbb{T}) \times \wp(\mathbb{T}) \rightarrow \wp(\mathbb{T})$ , is defined in terms of  $\wedge$  by composition with  $\alpha$  and  $\gamma$ ; note that our semantics is based upon an independent attribute model [10].

*Not.* The bijective function  $\neg : \wp(\mathbb{S}) \rightarrow \wp(\mathbb{S})$  represents a perfect zero delay NOT gate. As with  $\wedge$ , we define  $\neg^\# : \wp(\mathbb{T}) \rightarrow \wp(\mathbb{T})$  by composition of the concrete operator  $\neg$  with  $\alpha$  and  $\gamma$ . When tabulated,  $\wedge^\#$  and  $\neg^\#$  behave as shown in Table 2.

*Transmission line (non-inertial) delay.* Our definition of transmission line delay is essentially a superset of all possible delay functions that preserve the underlying trace structure of the signal. The definition,  $\gamma \circ \alpha$ , captures this behaviour straightforwardly; the  $\alpha$  function abstracts away all details of time, though preserves transitions and the values at the beginning and end of time, then  $\gamma$  concretises this, resulting in the set of all possible traces with similar structure. This definition is more permissive than more typical notions of delay in that it includes negative as well as positive time shifts as well as transformations that can stretch or compress (though not remove or reorder) pulses.

*Inertial delay.* Inertial delay is broadly similar to transmission line delay, in that, as well as changing the time at which transitions may occur, one or more complete pulses (i.e. pairs of adjacent transitions) may be removed. This models a common property of some physical components, whereby very short pulses are ‘soaked up’ by internal capacitance and/or inductance and thereby not passed on. We model inertial delay in the abstract domain – in effect, nondeterministic traces are mapped to convex hulls of the form  $F_{0..a} \mid T_{0..b} \mid \uparrow_{0..c} \mid \downarrow_{0..d}$ . The concrete inertial delay operator  $\square$  is defined in terms of  $\square^\sharp$  by composition with  $\gamma$  and  $\alpha$ , so as with transmission line delay, it encompasses all possible (correct) inertial delay functions. It can be noted that, for all  $\hat{s} \in \wp(\mathbb{S})$ ,  $\Delta\hat{s} \subseteq \square\hat{s}$ .

#### 4.1 Soundness and Completeness

An abstract function  $f^\sharp$  may be described as *sound* with respect to a concrete function  $f$  if all behaviours exhibited by  $f$  are within the set of possible behaviours predicted by  $f^\sharp$ . Where these sets are identical (i.e. where  $f^\sharp$  predicts all possible behaviours of  $f$ ), *completeness* holds [6–8, 5, 12], two forms of which are defined below.

**Definition 11.** *Given a concrete domain  $D$  and an abstract domain  $D^\sharp$ , related by adjoint functions  $\langle \alpha, \gamma \rangle$  that form a Galois connection (i.e.  $\alpha \circ \gamma(x) \sqsupseteq x$  and  $\gamma \circ \alpha(x) \sqsupseteq x$ ), a pair of functions  $f : D \rightarrow D$  and  $f^\sharp : D^\sharp \rightarrow D^\sharp$  may be said to be sound iff the following (equivalent) relations hold:*

$$\alpha \circ f \sqsubseteq f^\sharp \circ \alpha \qquad f \circ \gamma \sqsubseteq \gamma \circ f^\sharp$$

**Definition 12.** *Let  $f_{best}^\sharp \stackrel{\text{def}}{=} \alpha \circ f \circ \gamma$ .*

**Definition 13.** *Where  $f^\sharp = f_{best}^\sharp$  and  $f \circ \gamma = \gamma \circ f^\sharp$ , the property  $\gamma$ -completeness holds.*

**Definition 14.** *Where  $f^\sharp = f_{best}^\sharp$  and  $\alpha \circ f = f^\sharp \circ \alpha$ , the property  $\alpha$ -completeness holds.*

Note that  $\alpha$ -completeness and  $\gamma$ -completeness are orthogonal properties; neither implies the other, though if either or both kinds of completeness hold, soundness must also hold.

**Theorem 2.** *The transmission line delay operator  $(\Delta, \Delta^\sharp)$  is sound,  $\alpha$ -complete and  $\gamma$ -complete. Proof:*

1.  $\Delta_{best}^\sharp = \alpha \circ \Delta \circ \gamma = \alpha \circ \gamma \circ \alpha \circ \gamma = \alpha \circ \gamma = (\lambda x.x) = \Delta^\sharp$ .
2.  $\alpha$ -completeness:  $\alpha \circ \Delta = \alpha \circ \gamma \circ \alpha = \alpha = (\lambda x.x) \circ \alpha = \Delta^\sharp \circ \alpha$ .
3.  $\gamma$ -completeness:  $\Delta \circ \gamma = \gamma \circ \alpha \circ \gamma = \gamma = \gamma \circ (\lambda x.x) = \gamma \circ \Delta^\sharp$ .

**Theorem 3.** *The inertial delay operator  $(\square, \square^\sharp)$  is sound,  $\alpha$ -complete and  $\gamma$ -complete. Proof:*

1.  $\Box_{best}^\# = \alpha \circ \Box \circ \gamma = \alpha \circ \gamma \circ \Box^\# \circ \alpha \circ \gamma = \Box^\#$ .
2.  $\alpha$ -completeness:  $\alpha \circ \Box = \alpha \circ \gamma \circ \Box^\# \circ \alpha = \Box^\# \circ \alpha$ .
3.  $\gamma$ -completeness:  $\Box \circ \gamma = \gamma \circ \Box^\# \circ \alpha \circ \gamma = \gamma \circ \Box^\#$ .

**Theorem 4.** *The perfect NOT operator  $(\neg, \neg^\#)$  is sound,  $\alpha$ -complete and  $\gamma$ -complete. Proof:*

1.  $\neg_{best}^\# = \alpha \circ \neg \circ \gamma = \neg^\#$ .
2. Since  $\neg$  is a bijection,  $\gamma \circ \alpha \circ \neg = \neg \circ \gamma \circ \alpha$ .
3.  $\alpha$ -completeness:  $\alpha \circ \neg = \alpha \circ \gamma \circ \alpha \circ \neg = \alpha \circ \neg \circ \gamma \circ \alpha = \neg^\# \circ \alpha$ .
4.  $\gamma$ -completeness:  $\neg \circ \gamma = \neg \circ \gamma \circ \alpha \circ \gamma = \gamma \circ \alpha \circ \neg \circ \gamma = \gamma \circ \neg^\#$ .

**Theorem 5.** *The perfect AND operator  $(\wedge, \wedge^\#)$  is sound<sup>7</sup>. Proof:*

1.  $\wedge \circ \langle \gamma, \gamma \rangle \subseteq \gamma \circ \wedge^\# = \gamma \circ \alpha \circ \wedge \circ \langle \gamma, \gamma \rangle$ .

Note that whilst perfect, zero delay AND is sound but not complete, a composite speed-insensitive AND ( $\wedge_{SI} \stackrel{\text{def}}{=} \Delta \circ \wedge, \wedge_{SI}^\# \stackrel{\text{def}}{=} \Delta^\# \circ \wedge^\#$ ) can be straightforwardly be shown to be  $\gamma$ -complete, but not  $\alpha$ -complete. Dually, delay-independent AND ( $\wedge_{DI} \stackrel{\text{def}}{=} \wedge \circ \langle \Delta, \Delta \rangle, \wedge_{DI}^\# \stackrel{\text{def}}{=} \wedge^\# \circ \langle \Delta^\#, \Delta^\# \rangle$ ) is  $\alpha$ - but not  $\gamma$ -complete. We find, however, that ( $\wedge_{complete} \stackrel{\text{def}}{=} \Delta \circ \wedge \circ \langle \Delta, \Delta \rangle, \wedge_{complete}^\# \stackrel{\text{def}}{=} \Delta^\# \circ \wedge^\# \circ \langle \Delta^\#, \Delta^\# \rangle$ ) is both  $\alpha$ - and  $\gamma$ -complete.

## 5 Finite Versions of the Abstract Domain

The abstract domain defined in Section 3 allows arbitrary asynchronous combinational circuits to be reasoned about. In this section we present a number of simplifications of this basic model which allow accuracy to be traded off against levels of abstraction. The model presented in Section 3 is useful in identifying possible glitches within circuits, though in this case generally one is interested in whether a particular signal *can* glitch, rather than the *number* of possible glitches – this requires less information than that captured by our original abstraction. It follows that further abstraction should be possible, which is indeed the case.

### 5.1 Collapsing Non-Zero Subscripts: the 256-value Transitional Logic $\mathbb{T}_{256}$

Mapping all non-zero subscript traces  $t \in X_{1..∞}$  to the single abstract value  $X_+$ , for  $X$  ranging over  $\{F, T, \uparrow, \downarrow\}$ , makes it possible to define a finite abstract domain with a Galois connection to  $\mathbb{T}$ . This domain has the desirable property of abstracting away details of ‘how glitchy’ a trace may be, whilst retaining the ability to distinguish *clean* traces from *dirty* traces.

<sup>7</sup> Note that we adopt an independent attribute model when considering the dyadic nature of AND.

$\neg_c$		$\Delta_c$		$\square_c$		$\wedge_c$	$F_0$	$F_+$	$T_0$	$T_+$	$\uparrow_0$	$\uparrow_+$	$\downarrow_0$	$\downarrow_+$
$F_0$	$T_0$	$F_0$	$F_0$	$F_0$	$F_0$	$F_0$	$F_0$	$F_0$	$F_0$	$F_0$	$F_0$	$F_0$	$F_0$	$F_0$
$F_+$	$T_+$	$F_+$	$F_+$	$F_+$	$F_?$	$F_+$	$F_0$	$F_?$	$F_+$	$F_?$	$F_?$	$F_?$	$F_?$	$F_?$
$T_0$	$F_0$	$T_0$	$T_0$	$T_0$	$T_0$	$T_0$	$F_0$	$F_+$	$T_0$	$T_+$	$\uparrow_0$	$\uparrow_+$	$\downarrow_0$	$\downarrow_+$
$T_+$	$F_+$	$T_+$	$T_+$	$T_+$	$T_?$	$T_+$	$F_0$	$F_?$	$T_+$	$T_+$	$F_?$	$\uparrow_?$	$\downarrow_+$	$\downarrow_?$
$\uparrow_0$	$\downarrow_0$	$\uparrow_0$	$\uparrow_0$	$\uparrow_0$	$\uparrow_0$	$\uparrow_0$	$F_0$	$F_?$	$\uparrow_0$	$\uparrow_?$	$\uparrow_0$	$\uparrow_?$	$F_?$	$F_?$
$\uparrow_+$	$\downarrow_+$	$\uparrow_+$	$\uparrow_+$	$\uparrow_+$	$\uparrow_?$	$\uparrow_+$	$F_0$	$F_?$	$\uparrow_+$	$\uparrow_?$	$\uparrow_?$	$\uparrow_?$	$F_?$	$F_?$
$\downarrow_0$	$\uparrow_0$	$\downarrow_0$	$\downarrow_0$	$\downarrow_0$	$\downarrow_0$	$\downarrow_0$	$F_0$	$F_?$	$\downarrow_0$	$\downarrow_+$	$F_?$	$F_?$	$\downarrow_0$	$\downarrow_?$
$\downarrow_+$	$\uparrow_+$	$\downarrow_+$	$\downarrow_+$	$\downarrow_+$	$\downarrow_?$	$\downarrow_+$	$F_0$	$F_?$	$\downarrow_+$	$\downarrow_?$	$F_?$	$F_?$	$\downarrow_?$	$\downarrow_?$

where  $F_? \stackrel{\text{def}}{=} F_0 \mid F_+$ ,  $T_? \stackrel{\text{def}}{=} T_0 \mid T_+$ ,  $\downarrow_? \stackrel{\text{def}}{=} \downarrow_0 \mid \downarrow_+$ ,  $\uparrow_? \stackrel{\text{def}}{=} \uparrow_0 \mid \uparrow_+$

**Table 3.** Operators on  $\mathbb{T}_c$

**Definition 15.** *The abstract domain of subscript-collapsed deterministic traces is the set  $\mathbb{T}_c \stackrel{\text{def}}{=} \{F_0, F_+, T_0, T_+, \uparrow_0, \uparrow_+, \downarrow_0, \downarrow_+\}$ . Following the usual convention, the corresponding abstract domain of subscript-collapsed nondeterministic traces is the set  $\mathbb{T}_{256} \stackrel{\text{def}}{=} \wp(\mathbb{T}_c)$ .*

Note that unlike  $\mathbb{T}$  and  $\wp(\mathbb{T})$ , both  $\mathbb{T}_c$  and  $\wp(\mathbb{T}_c)$  are finite sets, with 8 and 256 members respectively.

**Definition 16.** *The Galois connection  $\alpha_c : \wp(\mathbb{T}) \rightarrow \wp(\mathbb{T}_c)$ ,  $\gamma_c : \wp(\mathbb{T}_c) \rightarrow \wp(\mathbb{T})$  is defined as follows:*

$$\beta_c X_n \stackrel{\text{def}}{=} \begin{cases} X_0 & \text{iff } n = 0; \\ X_+ & \text{otherwise.} \end{cases}$$

$$\alpha_c \hat{t} \stackrel{\text{def}}{=} \{\beta_c t \mid t \in \hat{t}\} \quad \gamma_c \hat{t} \stackrel{\text{def}}{=} \{t \in \mathbb{T} \mid \beta_c t \in \hat{t}\}$$

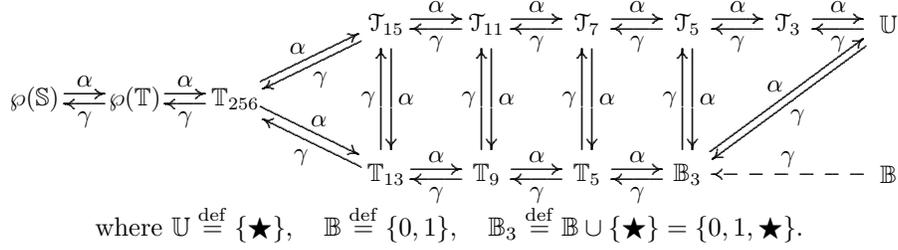
It is possible to tabulate  $256 \times 256$  truth tables that fully enumerate all members of  $\mathbb{T}_{256}$  along their edges, but they are too large to reproduce here in full. For brevity, Table 3 defines the operators  $\neg_c : \mathbb{T}_c \rightarrow \wp(\mathbb{T}_c)$ ,  $\Delta_c : \mathbb{T}_c \rightarrow \wp(\mathbb{T}_c)$ ,  $\square_c : \mathbb{T}_c \rightarrow \wp(\mathbb{T}_c)$  and  $\wedge_c : \mathbb{T}_c \times \mathbb{T}_c \rightarrow \wp(\mathbb{T}_c)$  on  $\mathbb{T}_c$ . Their fully nondeterministic versions, defined on  $\wp(\mathbb{T}_c)$ , are as follows:

$$\neg_c \hat{t} \stackrel{\text{def}}{=} \bigcup_{t \in \hat{t}} \{\neg_c t\} \quad \Delta_c \hat{t} \stackrel{\text{def}}{=} \bigcup_{t \in \hat{t}} \{\Delta_c t\} \quad \square_c \hat{t} \stackrel{\text{def}}{=} \bigcup_{t \in \hat{t}} \{\square_c t\} \quad \hat{t} \wedge_c \hat{u} \stackrel{\text{def}}{=} \bigcup_{\substack{t \in \hat{t} \\ u \in \hat{u}}} \{t \wedge_c u\}$$

Note that, as with  $\Delta^\sharp$ , the  $\Delta_c$  operator is merely an identity function.

## 6 Further Simplification of the Abstract Domain

A fully tabulated version of the  $\neg_c$ ,  $\Delta_c$ ,  $\square_c$  and  $\wedge_c$  operators defined in Section 5.1 can be regarded as a 256-value *transitional logic*, where the values are the



**Fig. 2.** Hierarchy of Domains

members of  $\wp(\mathbb{T}_c)$ . Such an approach still captures more nondeterminism than is useful in for many applications. It is possible to further reduce the abstract domain, replacing some nondeterministic choices with appropriate least upper bound elements with respect to  $\langle \wp(\mathbb{T}_c), \subseteq \rangle$ . The hierarchy of domains that results is shown in Fig. 2 – the relationship to 2-value Boolean logic  $\mathbb{B}$  and 3-value ternary logic  $\mathbb{B}_3$  is shown<sup>8</sup>. Note that since  $\mathbb{B}$  lacks an upper bound that corresponds with  $\star$ , it is not possible to define  $\alpha : \mathbb{B}_3 \rightarrow \mathbb{B}$  (though  $\gamma : \mathbb{B} \rightarrow \mathbb{B}_3$  can be trivially defined), so a Galois connection does not exist in that particular case. Following Cousot & Cousot [1, 2], the domain  $\mathbb{U}$ , *useless logic*, containing only  $\star$ , completes the lattice.

Finding the smallest lattice including  $\{F_0, F_+, T_0, T_+, \uparrow_0, \uparrow_+, \downarrow_0, \downarrow_+\}$  that is closed under  $\wedge_c$ , and  $\neg_c$  results in the 13-value transitional logic,

$$\mathbb{T}_{13} \stackrel{\text{def}}{=} \{F_0, F_+, F_?, T_0, T_+, T_?, \uparrow_0, \uparrow_+, \uparrow_?, \downarrow_0, \downarrow_+, \downarrow_?, \star\}$$

Though much smaller than  $\wp(\mathbb{T}_c)$ , this logic is equivalently useful for most purposes—note that a special element needs to be explicitly included,  $\star$ , representing the least upper bound (top element) of the lattice.

In cases where it is important to know that a trace is definitely clean, but where it is not necessary to distinguish between ‘definitely dirty’ and ‘possibly dirty’, further reducing the domain by folding  $F_+, T_+, \uparrow_+$  and  $\downarrow_+$  into their respective least upper bounds  $F_?, T_?, \uparrow_?$  and  $\downarrow_?$  results in a 9-value transitional logic,  $\mathbb{T}_9 \stackrel{\text{def}}{=} \{F_0, F_?, T_0, T_?, \uparrow_0, \uparrow_?, \downarrow_0, \downarrow_?, \star\}$ . An even simpler 5-value transitional logic  $\mathbb{T}_5 \stackrel{\text{def}}{=} \{F, T, \uparrow, \downarrow, \star\}$  results from folding all remaining nondeterminism into  $\star$ .  $\mathbb{T}_{13}$  and  $\mathbb{T}_9$  are well suited to logic simulation, refinement and model checking, whereas  $\mathbb{T}_5$  is only recommended for glitch checking.

## 6.1 Static/Clean Logics

The  $\mathbb{T}_{13}$ ,  $\mathbb{T}_9$  and  $\mathbb{T}_5$  logics can be usefully extended by introducing two extra upper bounds:  $S$ , the least upper bound of traces whose values are fixed for all

<sup>8</sup> As with our other logics, we assume that  $F \subseteq \star$  and  $T \subseteq \star$  – some ternary logics in the literature (notably Kleene’s) lack this formal requirement.

time, and  $C$ , the least upper bound of traces that may transition, but that *never* glitch.

**Definition 17.** *With respect to  $\wp(\mathbb{T}_c)$ , the least upper bounds  $S$ ,  $C$  and  $\star$  are defined as follows:*

$$\begin{aligned} S &\stackrel{\text{def}}{=} \{F_0, T_0\} & C &\stackrel{\text{def}}{=} \{F_0, T_0, \uparrow_0, \downarrow_0\} \\ \star &\stackrel{\text{def}}{=} \{F_0, F_+, T_0, T_+, \uparrow_0, \uparrow_+, \downarrow_0, \downarrow_+\} \end{aligned}$$

The resulting *static/clean* transitional logics  $\mathcal{T}_{15} \stackrel{\text{def}}{=} \mathbb{T}_{13} \cup \{S, C\}$ ,  $\mathcal{T}_{11} \stackrel{\text{def}}{=} \mathbb{T}_9 \cup \{S, C\}$  and  $\mathcal{T}_7 \stackrel{\text{def}}{=} \mathbb{T}_5 \cup \{S, C\}$  have applications in the design rule checking of ‘impure’ synchronous circuits. For example, a gated clock input represented by  $S \wedge C = C$  might be accepted by a model checker, but  $C \wedge C = \star$  would not.

Removing  $\uparrow$  and  $\downarrow$  from  $\mathcal{T}_7$  results in a 5-value static/clean logic  $\mathcal{T}_5 \stackrel{\text{def}}{=} \{F, T, S, C, \star\}$  capable of reasoning about gated clock synchronous circuits; an even simpler (though less accurate) 3-value static/clean logic  $\mathcal{T}_3 \stackrel{\text{def}}{=} \{S, C, \star\}$  results from also removing  $F$  and  $T$ .

## 7 Refinement and Equivalence in Transitional Logics

Hardware engineers frequently concern themselves with modification and optimisation of existing circuits, so it is appropriate to support this by defining equivalence and refinement with respect to our abstract domains. Refinement relationships between circuits are analogous to concepts of refinement in process calculi, and may similarly be used to aid provably correct design. For example, the Boolean equivalence  $a \wedge \neg a = F$  is not a strong equivalence in our model, nor is it a weak equivalence – it actually turns out to be a (left-to-right) refinement, i.e.  $a \wedge \neg a \succcurlyeq F_0$ , reflecting the ‘engineer’s intuition’ that it is safe to replace  $a \wedge \neg a$  with  $F_0$ , but that the converse could damage the functionality of the circuit by introducing new glitch states that were not present in the original design.

Informally, if the deterministic trace  $u \in \mathbb{T}$  *refines* (i.e. retains the steady state behaviour of, but is no more glitchy than) trace  $t \in \mathbb{T}$ , this may be denoted  $t \succcurlyeq u$ .

**Definition 18.** *Given a pair of traces  $t \in \mathbb{T}$  and  $u \in \mathbb{T}$ ,*

$$t \succcurlyeq u \stackrel{\text{def}}{=} \text{Val}(t) = \text{Val}(u) \wedge \text{Subs}(t) \geq \text{Subs}(u)$$

For example,  $F_1 \succcurlyeq F_0$ ,  $T_3 \succcurlyeq T_2$ ,  $\uparrow_5 \succcurlyeq \uparrow_5$ , but  $\downarrow_0$  and  $\uparrow_1$  are incomparable. Where  $t \in \mathbb{T}$  and  $u \in \mathbb{T}$ , if  $t \succcurlyeq u$  and  $u \succcurlyeq t$ , it follows that  $t = u$ .

Refinement and equivalence for nondeterministic traces is slightly less straightforward, in that it is necessary to handle cases like  $\downarrow_{1|3|5} \succcurlyeq \downarrow_{0|2|4}$ . To make these comparable, we construct *convex hulls* of the form  $X_{0..n}$  enclosing the nondeterministic choices, so the above case becomes equivalent to  $\downarrow_{0..5} \succcurlyeq \downarrow_{0..4}$ . In

effect, this approach compares worst-case behaviour, disregarding finer detail; in practice, since  $\wedge$ ,  $\Delta$ ,  $\square$  and  $\neg$  typically return results of the general form  $X_{0..n}$  anyway, this tends not to cause any practical difficulties. Less permissive definitions of refinement, e.g.  $\hat{t} \succ_{strict} \hat{u} \equiv \forall t \in \hat{t} . \forall u \in \hat{u} . t \succ u$ , often disallow too many possible optimisations that in practice are quite acceptable—our model better reflects the engineer’s intuition that ‘less glitchy is better,’ but that very detailed information about the structure of possible glitches is generally not important.

**Definition 19.** Where  $\hat{t} \in \wp(\mathbb{T})$  and  $\hat{u} \in \wp(\mathbb{T})$ ,

$$\hat{t} \succ \hat{u} \stackrel{\text{def}}{=} (\forall t \in \hat{t}, u \in \hat{u} . Val(t) = Val(u) \wedge MaxSubs(\hat{t}) \geq MaxSubs(\hat{u}))$$

where  $MaxSubs(\hat{t}) \stackrel{\text{def}}{=} \max_{t \in \hat{t}} Subs(t)$  is a function returning the largest subscript of a nondeterministic trace.

*Equivalence of Nondeterministic Traces.* Where  $\hat{t} \in \wp(\mathbb{T})$  and  $\hat{u} \in \wp(\mathbb{T})$ , if  $\hat{t} = \hat{u}$  then the traces are *strongly equivalent*, i.e. they represent exactly the same sets of nondeterministic choices. If the convex hulls surrounding  $\hat{t}$  and  $\hat{u}$  are identical, as is the case when  $\hat{t} \succ \hat{u} \wedge \hat{u} \succ \hat{t}$ , the traces may be said to be *weakly equivalent*, denoted  $\hat{t} \simeq \hat{u}$ . Where  $\hat{t} \succ \hat{u} \vee \hat{u} \succ \hat{t}$ , the traces are *comparable*, denoted  $\hat{t} \simeq \hat{u}$ .

*Finite Abstract Domains Refinement and equivalence* can also be defined for the finite abstract domain  $\mathbb{T}_{256}$  and some of its simplified forms. Since  $\mathbb{T}_{256}$  is implicitly nondeterministic, we do not need to consider the deterministic case.

**Definition 20.** Given traces  $t \in \mathbb{T}_{256}$  and  $u \in \mathbb{T}_{256}$ ,

$$\begin{aligned} t \succ u &\stackrel{\text{def}}{=} Val(t) = Val(u) \wedge (Subs(t) = Subs(u) \vee Subs(u) = 0) \\ t \simeq u &\stackrel{\text{def}}{=} t \succ u \wedge u \succ t \equiv t = u \\ t \simeq u &\stackrel{\text{def}}{=} t \succ u \vee u \succ t \equiv Val(t) = Val(u) \end{aligned}$$

## 8 Related Work

There seems to be relatively little work reported in the literature regarding the application of modern program analysis techniques to hardware.

Don Gaubatz [4] proposes a 4-value ‘quaternary’ logic that bears some resemblance to our 5-value transitional logic.

Paul Cunningham [3] extends Gaubatz’s work in many respects, though his formalism is based on a conventional 2-value logic with transitions handled explicitly as events rather than as values in an extended logic.

Charles Hymans [9] uses abstract interpretation to present a safety property checking technique based upon abstract interpretation of (synchronous) behavioural VHDL specifications.

## 9 Conclusions

In this paper, we have presented a technique based upon the solid foundation of abstract interpretation [1, 2] that allows properties of a wide class of digital circuits to be reasoned about. We describe what is essentially a first attempt at applying abstract interpretation to asynchronous hardware—clearly more can be done, particularly in exploring completeness.

### 9.1 Future Work

In Section 7, we define refinement and equivalence relations on circuits. It appears to be possible to generalise this definition of refinement and equivalence to any abstract domain that is itself amenable to abstract interpretation. We have already demonstrated that our technique is potentially useful for logic simulation [13] – implementing a demonstrable simulator is a logical next step.

An experimental proof system exists for the 11-value clean/static transitional logic, and we hope to extend this to cover the more general case,  $\wp(\mathbb{T})$ .

## Acknowledgments

This paper has greatly benefited from comments received on early drafts. In particular, we wish to thank the anonymous reviewers, Patrick and Radhia Cousot, Charles Hymans, the Semantics and Abstract Interpretation group at École Normale Supérieure, as well as the CPRG and Rainbow groups at Cambridge.

The first author wishes to thank Big Hand Ltd. and Senshutek Ltd. for financially supporting this work.

## References

1. COUSOT, P., AND COUSOT, R. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Los Angeles, California, 1977), ACM Press, New York, NY, pp. 238–252.
2. COUSOT, P., AND COUSOT, R. Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Antonio, Texas, 1979), ACM Press, New York, NY, pp. 269–282.
3. CUNNINGHAM, P. A. *Verification of Asynchronous Circuits*. PhD thesis, University of Cambridge, 2002.
4. GAUBATZ, D. A. *Logic Programming Analysis of Asynchronous Digital Circuits*. PhD thesis, University of Cambridge, 1991.
5. GIACOBazzi, R., AND MASTROENI, I. Domain compression for complete abstractions. In *Fourth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'03)* (2003), vol. 2575 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 146–160.

6. GIACOBAZZI, R., AND RANZATO, F. Completeness in abstract interpretation: A domain perspective. In *Proc. of the 6th International Conference on Algebraic Methodology and Software Technology (AMAST'97)* (1997), M. Johnson, Ed., vol. 1349 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 231–245.
7. GIACOBAZZI, R., AND RANZATO, F. Refining and compressing abstract domains. In *Proc. of the 24th International Colloquium on Automata, Languages, and Programming (ICALP'97)* (1997), R. G. P. Degano and A. Marchetti-Spaccamela, Eds., vol. 1256 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 771–781.
8. GIACOBAZZI, R., RANZATO, F., AND SCOZZARI, F. Making abstract interpretations complete. *Journal of the ACM* 47, 2 (2000), 361–416.
9. HYMANS, C. Checking safety properties of behavioral VHDL descriptions by abstract interpretation. In *9th International Static Analysis Symposium (SAS'02)* (2002), vol. 2477 of *Lecture Notes in Computer Science*, Springer, pp. 444–460.
10. JONES, N. D., AND MUCHNICK, S. Complexity of flow analysis, inductive assertion synthesis, and a language due to Dijkstra. In *21st Symposium on Foundations of Computer Science* (1980), IEEE, pp. 185–190.
11. MORELLI, G. Coralled: Get hold of wire delays. *Electronic Design News, September 25, 2003*, pp. 37–46.
12. MYCROFT, A. Completeness and predicate-based abstract interpretation. In *Proc. ACM conf. on Partial Evaluation and Program Manipulation* (2003), pp. 179–185.
13. THOMPSON, S., AND MYCROFT, A. Sliding window logic simulation. In *15th UK Asynchronous Forum* (2004), Cambridge.