Hazard Algebra for Asynchronous Circuits

E. Meijer

UU-CS-1994-06
January 1994

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

Hazard Algebra for Asynchronous Circuits

Erik Meijer
University of Utrecht
Department of Computer Science
POBox 80.089 NL-3508 TB Utrecht The Netherlands
email: erik@cs.ruu.nl

dat ene moment
dat de tijd stopt
dat ene moment
dat alles klopt
dat ene moment
niks te kort
niet te veel
dat je onderdeel bent
van het grote geheel
De Dijk

1 Introduction

Our concern is the design of provably correct asynchronous circuits. In such circuits there may occur hazards, due to the delay of signals along wires and components. Informally, a hazard is a time interval during which the output of a circuit, or circuit component, is wrong. Hazards are hardly ever defined formally in the literature. For a synchronous circuit designer this might be no problem as he assumes the outputs of all components are correct (and stabile) by the next clock edge. Asynchronous or clockless circuits may feature hazards, even though all components are correct. So it is of vital importance for the asynchronous system designer to be able to reason formally about hazards. We shall design an algebra, a formal system, for this purpose and we shall use it to design asynchronous finite-state machines. Our work grew out of an attempt to formalize Peyton Jones's technique for designing asynchronous finite-state machines [3]. A condensed version of this paper appeared in [7]

Notation

The base algebra we use is Boolean algebra with values $\mathbb{B} = \{1, 0\}$ (switching algebra) and with operations $\neg$ (negation), $+$ (disjunction), and $\cdot$ (conjunction). Equality is lifted pointwise

ISSN 0924-0275

3 Conventional algebras

We intend to design an algebra by which it is possible to prove the absence of hazards in asynchronous circuits. A formal system which takes time into account is certainly going to be too complicated to be of practical use, hence a formalism in which time has been abstracted is to be preferred. However, as shown above switching algebra over $\mathbb{B}$ is too simplistic an abstraction of logic circuits.

Another possibility is the *ternary* algebra as used by Eichelberger [1] and others. Here the value set $\mathbb{B} = \{1, 0\}$ is extended with a third value $\frac{1}{2}$ to become $\mathbb{T} = \{0, \frac{1}{2}, 1\}$. This new value is intended to describe the transient behaviour of logic gates. The operation of gates with respect to the transient value $\frac{1}{2}$ is determined by changing back and forth the required input of the gate and noting whether its output changes or remains fixed. Thus for example

$$\begin{aligned} \tfrac{1}{2} \cdot 1 &= \tfrac{1}{2} \quad \text{since } 0 \cdot 1 \neq 1 \cdot 1, \text{ and} \\ \tfrac{1}{2} \cdot 0 &= 0 \quad \text{since } 0 \cdot 0 = 0 = 1 \cdot 0. \end{aligned}$$

In ternary algebra we may define that an expression $A[p]$ has a *static hazard* on variable p if $A = \frac{1}{2}$ when p changes, but A yields the same $\mathbb{B}$-value for p either constant 1 or 0.

$$\text{static hazard } (p, A) \equiv A[p := 1] = A[p := 0] \wedge A[p := \tfrac{1}{2}] = \tfrac{1}{2}$$

Similarly we define that A has a *dynamic hazard* on p when $A = \frac{1}{2}$ if $p = \frac{1}{2}$ but A yields two different boolean values for $p = 1$ resp. $p = 0$.

$$\text{dynamic hazard } (p, A) \equiv A[p := 1] \neq A[p := 0] \wedge A[p := \tfrac{1}{2}] = \tfrac{1}{2}$$

According to these definitions the 2-input data selector $\overline{p} \cdot r + p \cdot q$ has a static hazard on p when q and r are both 1.

Not so obvious perhaps is the fact that in ternary algebra the notion of dynamic hazard is hardly useful since we can prove that for each A

$$A[p := 0] \neq A[p := 1] \Rightarrow \text{dynamic hazard } (p, A)$$

That is, the notion of dynamic hazard does not distinguish between truly occurring dynamic hazards and merely correct changes of the output. A more refined notion of dynamic hazard is wanted but this seems impossible within ternary algebra.

4 Hazard algebra

The solution to the problem of giving formal definitions of static and dynamic hazards is a quinary algebra, to be called *hazard algebra*. The domain of values is a five element set $\mathbb{H} = \mathbb{B} \cup \{\uparrow, \downarrow, \bot\}$, structured as a flat cpo with $\bot$ as least element.

$$\bot \sqsubset a \equiv a \neq \bot$$

As usual the ordering on values is lifted pointwise to terms

$$A \sqsubseteq_{\mathbb{H}} B \quad \equiv \quad (\forall\, x \in \mathbb{H} :: A[p := x] \sqsubseteq B[p := x]) \qquad \text{(LIFT)}$$

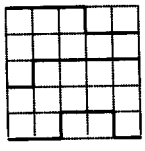
(Again the extension to more than one variable should be obvious). When no confusion can arise the $\mathbb{H}$ subscript is omitted.

Informally speaking,

- 1 (0) denotes a *constant high* (*constant low*) signal during an interval of time.
- A clean *rising* signal is denoted by $\uparrow$.
- A clean *falling* signal is denoted by $\downarrow$.
- $\perp$ denotes an *unknown* signal, possibly a hazard.

Fantuauzzi [2] describes a nine-valued algebra for the analysis of logical circuits that distinguishes between five different kinds of hazards instead of only one. This seems to be a little overkill. In his paper Fantuauzzi refers to a quinary algebra proposed by Lewis [5]. It might very well be the case that this quinary algebra is very similar to ours. This certainly is the case for the five-valued transitional logic of Thompson [9]. However he does not distinguish between the full ordering $\sqsubseteq$ and its refinements $\sqsubseteq_p$ and $\sqsubseteq_\forall$.

The operations of ternary algebra are extended to quinary algebra in conformity with the operational interpretation of $\uparrow, \downarrow$, and $\perp$. For example $\uparrow \cdot \downarrow = \perp$ reflects the possibility of a hazard when the inputs of an and-gate change in the opposite direction within the same interval of time.



A complete characterization of the quinary operators is given below. Appendix A contains the corresponding truth tables.

$$\begin{aligned} x \cdot y &= y \cdot x \\ x \cdot x &= x \\ 0 \cdot x &= 0 \\ 1 \cdot x &= x \\ \perp \cdot x &= \perp \text{, if } x \neq 0 \\ \uparrow \cdot \downarrow &= \perp \end{aligned}$$

$$\begin{aligned} \overline{0} &= 1 \\ \overline{1} &= 0 \\ \overline{\perp} &= \perp \\ \overline{\uparrow} &= \downarrow \\ \overline{\downarrow} &= \uparrow \end{aligned}$$

$$\begin{aligned} x + y &= y + x \\ x + x &= x \\ 0 + x &= x \\ 1 + x &= 1 \\ \perp + x &= \perp \text{, if } x \neq 1 \\ \uparrow + \downarrow &= \perp \end{aligned}$$

The static hazard in the data selector can be computed within the new algebra as follows

$$(\overline{p} \cdot r + p \cdot q)[p := \uparrow, r := 1, q := 1] = \overline{\uparrow} \cdot 1 + \uparrow \cdot 1 = \downarrow \cdot 1 + \uparrow \cdot 1 = \downarrow + \uparrow = \perp$$

As a corollary we get that the hazard behaviour of a term never gets worse when replacing it by a bigger one.

$$\begin{aligned} safe\,(p, A) \wedge A \sqsubseteq_p B &\Rightarrow safe\,(p, B) \\ safe\,(A) \wedge A \sqsubseteq_\forall B &\Rightarrow safe\,(B) \end{aligned}$$

The above monotonicity laws are useful since many laws for $\sqsubseteq, \sqsubseteq_\forall$ and $\sqsubseteq_p$ have a simpler lhs operand than rhs operand. The laws enable us to replace an expression by a simpler one without losing safety.

5 Hazard removal

This section shows how arbitrary expressions can be made hazard-free by expanding and subsequent covering. Data selectors play a prominent role in this process.

The by far most important property of 2-input data selectors is *the fundamental mode abides law*: For all p, q not occurring in A, B, C, and D

$$(A \lhd p \rhd B) \lhd q \rhd (C \lhd p \rhd D) \;=_\forall\; (A \lhd q \rhd C) \lhd p \rhd (B \lhd q \rhd D)$$

Its proof follows from the fundamental mode distributive laws. Another law which follows from the same distribution rules is fundamental mode distribution of $\lhd p \rhd$ over $+$ for all p not in either A, B, C, or D.

$$(A \lhd p \rhd B) + (C \lhd p \rhd D) \;=_\forall\; (A + C) \lhd p \rhd (B + D)$$

Distribution of $\lhd p \rhd$ over $\cdot$ improves the hazard behaviour.

$$(A \lhd p \rhd B) \cdot (C \lhd p \rhd D) \;\sqsubseteq\; (A \cdot C) \lhd p \rhd (B \cdot D)$$

The *Shannon expansion* $p \textcircled{S} A$ on variable p realizes A using a 2-input data selector controlled by p with $A[p := 1]$ and $A[p := 0]$ at the respective data inputs.

$$p \textcircled{S} A \;=\; A[p := 0] \lhd p \rhd A[p := 1] \qquad \text{(SHANNON)}$$

Shannon expansion preserves meaning, $A =_{\mathbb{B}} p \textcircled{S} A$, a fact which is easily verified by a case analysis on p. The abides law and the distribution law for data selectors are inherited by expansion. Idempotence of $\textcircled{S}$ follows from the equality $(p \textcircled{S} A)[p := b] = A[p := b]$ if $b \in \mathbb{B}$.

$$\begin{aligned} p \textcircled{S} (A + B) &=_\forall (p \textcircled{S} A) + (p \textcircled{S} B) \\ p \textcircled{S} (p \textcircled{S} A) &= p \textcircled{S} A \\ p \textcircled{S} (q \textcircled{S} A) &=_\forall q \textcircled{S} (p \textcircled{S} A) \end{aligned}$$

Because of the last two laws, it is safe to overload the notation for expansion over a set of variables in fundamental mode.

$$\begin{aligned} \{\} \textcircled{S} A &= A \\ (\{p\} \cup ps) \textcircled{S} A &=_\forall p \textcircled{S} (ps \textcircled{S} A) \end{aligned}$$

Expanding an expression A for each variable in A is denoted by $\boxed{A}$

$$\boxed{A} \;=\; \{p \mid p \in A\}ⓢA$$

In practical terms this means that A is realized using only 2-input data selectors. Total expansion expansion is idempotent and distributes over +, both in fundamental mode.

$$\begin{aligned} \boxed{\boxed{A}} &=_\forall \boxed{A} \\ \boxed{A+B} &=_\forall \boxed{A}+\boxed{B} \end{aligned}$$

An expression A is called *expanded* if it is equal in fundamental mode to the expansion on any variable occurring in A.

$$\text{expanded}\ (A) \;\equiv\; (\forall\, p \in A :: A =_\forall pⓢA)$$

Total expansion yields an expanded term because additional expansion on variables in $\boxed{A}$ have no effect.

$$(\forall\, p : p \in \boxed{A} : \boxed{A} =_\forall pⓢ\boxed{A})$$

Expressions in *sum-of-product*-form are also expanded if products do not contain a variable and its negation, e.g. $p \cdot \overline{p}$ is forbidden. In fact any totally expanded expression is fundamental mode equivalent to an expression in sum-of-product form. For example let A be an expression in terms of p and q then

$$\begin{aligned} \boxed{A} \;=_\forall\; & \overline{p} \cdot \overline{q} \cdot A[p := 0, q := 0] + \\ & \overline{p} \cdot q \cdot A[p := 0, q := 1] + \\ & p \cdot \overline{q} \cdot A[p := 1, q := 0] + \\ & p \cdot q \cdot A[p := 1, q := 1] \end{aligned}$$

Expanded variables can be distributed n fundamental mode over the remaining expression.

Shannons expansion theorem can be used to provide hazard-cover for transitions of variable p. The *cover on* p *of* A is defined as

$$pⓒA = A[p := 0] \cdot A[p := 1]$$

Adding a cover term preserves meaning, $A =_B A + pⓒA$, and removes hazards on p for terms expanded on p.

$$\text{safe}\ (p, pⓢA + pⓒA) \qquad \text{(Hazard-Cover)}$$

A quick check of the truth-table for $p = \downarrow$ confirms this claim.

A[p := 0]	A[p := 1]	pⓢA + pⓒA
0	0	$\uparrow \cdot 0 + 0 \cdot 0 + \downarrow \cdot 0 = 0$
0	1	$\uparrow \cdot 0 + 0 \cdot 1 + \downarrow \cdot 1 = \downarrow$
1	0	$\uparrow \cdot 1 + 1 \cdot 0 + \downarrow \cdot 0 = \uparrow$
1	1	$\uparrow \cdot 1 + 1 \cdot 1 + \downarrow \cdot 1 = 1$

$$
\begin{array}{ll}
 & safe\ (p, A^*) \\
\equiv & \\
 & safe\ (p,\ (\sum\ ps \subseteq A :: \boxed{ps \copyright A}))\ ) \\
\equiv & \\
 & safe\ (p, \boxed{A} + \boxed{p \copyright A} + (\sum\ ps \subseteq A : p \notin ps : \boxed{(\{p\} \cup ps) \copyright A}) + \\
 & \quad (\sum\ ps \subseteq A : p \notin ps : \boxed{ps \copyright A})\ ) \\
\equiv & \\
 & safe\ (p, \boxed{A} + \boxed{p \copyright A})
\end{array}
$$

When A is in sum-of-product form, the formula A^* really hides a simple procedure that may be used when hazard covering is applied manually: as long as $A = B + \overline{p} \cdot C + p \cdot D$ with $C \cdot D \notin B$ and $C \cdot D \neq 0$, add $C \cdot D$ to A.

6 Asynchronous Finite-state Machines

This section shows an application of our hazard algebra to the implementation of asynchronous finite state machines. An *asynchronous finite state machine* consists of

- A set of inputs $\mathcal{I} = \{a, b, c, \ldots\}$.
- A set of outputs $\mathcal{O} = \{m, n, o, \ldots\}$.
- A set of states $\mathcal{S} = \{R, S, T, \ldots\}$.
- An output specification m_S which specifies for each state S and each output m a boolean expression over $\mathcal{I}$, whose value m should take when in state S.
- A transition specification K_{ST} over $\mathcal{I}$ for pairs of states S and T denoting that when the machine is in state S and K_{ST} holds a transition should be made to state T.

The transitions should be *complete* and *deterministic*. Completeness means that each state S has at least one successor state T (which may be S itself)

$$(\sum\ T :: K_{ST}) \ = \ 1 \qquad \text{(COMPL)}$$

Determinacy means that in any state S at most one transition is enabled

$$K_{ST} \cdot K_{ST'} \ = \ 0, \text{ if } T \neq T' \qquad \text{(DET)}$$

From the two conditions (COMPL) and (DET) on transitions it follows that

$$K_{SR} \ = \ (\prod\ T : T \neq R : \overline{K_{ST}}) \qquad \text{(SELF)}$$

Without loss of generality we therefore assume $K_{SS} = (\prod\ T : T \neq S : \overline{K_{ST}})$. There are two possible cases to consider when proving (SELF); transition K_{SR} is enabled and all others are disabled, or $K_{SR} = 0$ and some other transition $K_{SR'}$ is set.

state has been reached.

$$m \;=\; (\sum\; S :: S \cdot m_S)^* \qquad \text{(Moor)}$$

Next-state variables are treated the same as outputs,

$$y' \;=\; (\sum\; S,T :: S \cdot K_{ST} \cdot y_T)^* \qquad \text{(State)}$$

where y_T is the value of state variable y in state T.

7 Comparison with Peyton Jones

Peyton Jones [3] uses as equation for outputs

$$m \;=\; (\sum\; S,T : K_{ST} \neq 0 : (S+T) \cdot m_S \cdot m_T) \qquad (\text{OUT}_{\text{SPJ}})$$

This is essentially the result of pushing $\_^*$ inside our Moore-machine equation. Formally this means we have to show

$$(\sum\; S,T : K_{ST} \neq 0 : (S+T) \cdot m_S \cdot m_T) \;=_\forall\; (\sum\; S :: S \cdot m_S)^*$$

At this moment we lack nice algebraic laws for the $\_^*$-operator to prove this. Similarly his equation for state variables

$$y' \;=\; \sum\; S,T : K_{ST} \neq 0 : (S+T) \cdot y_S^* \cdot y_T^* \qquad (\text{State}_{\text{SPJ}})$$

where y_S^* is defined for arbitrary S as

$$y_S^* \;=\; y_S \cdot \overline{(\sum\; T : T \neq S : \overline{y_T} \cdot K_{ST})} \cdot (\sum\; T : T \neq S : y_T \cdot K_{ST})$$

corresponds to our equation (OUT). We believe however that our formulation is much simpler.

The following machine shows that the equations for outputs given by Peyton Jones are wrong for Mealy-type machines. The machine has a single input a, one output m and two states S and T encoded by y respectively $\overline{y}$.

$$\begin{aligned} m_S &= a \\ m_T &= 0 \\ K_{ST} &= a \\ K_{TT} &= 1 \end{aligned}$$

Expanding the expression for m suggested by Peyton Jones gives

$$m \;=\; y \cdot a$$

which is makes the output m temporarily high just before a transition from S to T. Our equation hardwires m to 0.