The Eventual C-Element Theorem for Delay-Insensitive Asynchronous Circuits

Rajit Manohar*
Yale University
New Haven, CT 06520, USA
rajit.manohar@yale.edu

Yoram Moses†*
Technion-Israel Institute of Technology
Haifa 32000, Israel
moses@ee.technion.ac.il

***Abstract*—In a seminal 1990 paper [9], Martin presents the C-element Theorem which implies, roughly, that the class of purely delay insensitive (DI) circuits is fundamentally limited in the set of functions it can implement. We provide circuit examples, both DI and not DI, that violate assumptions or results from [9], showing that the set of circuits considered by [9] is more limited than one might expect—and hence, the results from [9] are weaker than previously thought. In this paper we address this issue and amend matters. We use a model of circuits that naturally captures DI properties, and in particular admits all of our circuit examples. We prove the *Eventual C-element Theorem* for DI circuits. This is a variant of the C-element Theorem that similarly implies that the class of DI circuits is very limited. The proof of the new theorem uses novel techniques, and in particular makes use of a combinatorial property not present in previous work. With the new result, Martin's original insight and its profound implications are restored and provided with a new mathematical justification.**

I. Introduction

Delay-insensitive (DI) circuits are asynchronous circuits that operate correctly in the presence of arbitrary delays on gates and wires. DI circuits are an appealing notion, because they are extremely robust to the physical characteristics of gates and wires that impact timing. In a highly influential paper, Martin considered the properties of delay-insensitive circuits [9]. Its main result was the "C-element theorem," which implies that the class of DI circuits is extremely limited. Intuitively, the C-element theorem says that if a circuit is DI and all the variables in the circuit change at least three times, then every gate in the circuit can be replaced with a C-element without affecting the computations of the circuit! The focus of this paper is on extending the applicability of this remarkable result to a much wider class of asynchronous circuits.

We examine the restrictions on the class of circuits to which the results from [9] apply. Most of these restrictions arise from the set of assumptions made in the analysis. The first assumption comes from the definition of a "gate" as a structure that only has a single logical output. (Multi-output gates are considered, but they correspond to a single output gate combined with fanout on a wire—i.e., a wire fork.) This is a natural assumption when using a CMOS gate as the basic building block for asynchronous logic. Note that this is an assumption, and alternatives do exist (see, for example [11]). The second assumption arises from requiring that the system is *closed*—i.e., the description of the DI circuit of interest includes the environment in which it operates. The rationale for this assumption is that one may correct a delay-insensitivity problem in a circuit only to realize that it will re-appear in the circuit's environment. Thus, by considering the entire system as a whole, we can analyze the *global* implications of delay-insensitivity. In this paper, we also adopt both these assumptions.

A third assumption made in the C-element theorem analysis has to do with how the set of *computations* of a circuit is defined. The analysis in [9] defines a computation as a partial order of transitions, where a transition corresponds to a signal changing from low to high or high to low. One assumption underlying the C-element theorem is that the circuit of interest has only one computation.

This third assumption automatically removes from consideration any circuit whose partial order of signal transitions depends on timing. As a simple example, consider a standard edge-to-pulse generator circuit shown in Figure 1. The circuit converts a rising edge on x into a pulse on y, and the pulse width is determined by the relative delay of the two inputs to the AND gate. This circuit is clearly not DI; given a short enough delay on the inverter and a long enough delay on the wire from x to the input of the AND gate the circuit generates no pulses, hence no transitions on y.

We should expect that the C-element theorem easily shows that this circuit is not DI. However, by changing the relative delays of gates and wires in the circuit, the circuit can have different partial orders of transitions. Hence, the C-element theorem from [9] does not apply to this non-DI circuit. This paper provides an analysis of DI circuits that does not constrain the set of circuits by their types of computation up front.

In any computation, a non-final signal transition is immediately followed (in the partial order of signal transitions)

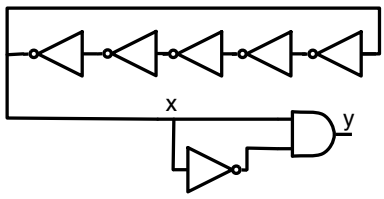


Fig. 1: A simple pulse-generator circuit.

† Yoram Moses is the Israel Pollak academic chair at the Technion, and was supported in part by the ISF grant 1520/11.
* Both authors' research was supported in part by a Ruch grant from the Jacobs Institute at Cornell Tech, and by the NSF-BSF grant CCF 1617945.

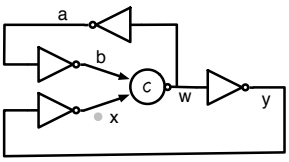


Fig. 2: Circuit example to illustrate violation of the argument used to establish the Projection Theorem. Initially $w = b = 1$, $a = 0$, $x = 0$, and $y = 0$.

by a set of other signal transitions. As shown in [9], we can associate a successor set of variables with each signal transition, which simply lists the variable changes immediately following a signal transition. [9] states that in a DI circuit every non-final transition of the same variable has the same successor set—this is called the *unique successor set* (USS) property. Furthermore, the Projection Theorem of [9] states that if a computation has the USS property, then its projection on a subset of variables also has the USS property. We now show an example of a DI circuit that exhibits the USS property, but where the Projection Theorem does not hold—in other words, we have an example of a DI circuit that cannot be analyzed by [9].

Consider the circuit in Figure 2. This circuit can have the following sequence of firings:

$$x\uparrow; w\downarrow; y\uparrow; x\downarrow; a\uparrow; b\downarrow; w\uparrow; y\downarrow; x\uparrow; a\downarrow; b\uparrow; w\downarrow; \ldots$$

In this circuit, the unique successor set of x is $\{w\}$, the unique successor set of w is $\{a, y\}$, the unique successor set of a is $\{b\}$, and the unique successor set of y is $\{x\}$.[1] The circuit can also have the following sequence of firings:

$$x\uparrow; w\downarrow; y\uparrow; a\uparrow; x\downarrow; b\downarrow; w\uparrow; y\downarrow; a\downarrow; x\uparrow; b\uparrow; w\downarrow; \ldots$$

and this firing sequence is also consistent with the successor sets described earlier. However, projecting the sequence of firings on the set $\{x, a, y\}$ leads to the following two sequences:

$$x\uparrow; y\uparrow; x\downarrow; a\uparrow; y\downarrow; x\uparrow; a\downarrow; \ldots$$
$$x\uparrow; y\uparrow; a\uparrow; x\downarrow; y\downarrow; a\downarrow; x\uparrow; \ldots$$

The Projection Theorem argues that the unique successor set for x should now be $\{a, y\}$. However, it is clear from the two sequences of firings that this is not the case since a does not fire between the first two transitions on x in the first execution sequence. However, the circuit in Figure 2 is in fact a DI circuit. Hence, the results in [9] cannot be applied to the circuit depicted in Figure 2.

The circuits from Figure 1 (a non-DI circuit) and Figure 2 (a DI circuit) are not amenable to the analysis presented in [9]. This implies that the set of circuits to which the results from [9] can be applied is more constrained than previously thought. In this paper, we present an alternate approach to the analysis of DI circuits that does not suffer from these limitations.

[1] If we were to model the fork on variable w explicitly by introducing two additional variables w_1 and w_2, then the unique successor set of w is $\{w_1, w_2\}$, of w_1 is $\{a\}$, and of w_2 is $\{y\}$. Instead of simply hiding w by projection, we would hide w, w_1, and w_2.

We adopt the framework of [8], which adapts concepts from the distributed systems literature to the domain of asynchronous circuits. We view asynchronous circuits as a particular class of distributed systems, in which Boolean gates serve as the processes, and wires correspond to communication links. Roughly speaking, "messages" are sent by toggling the value of a wire, "up" from 0 to 1 or "down" from 1 to 0. We are able to establish characterizations of the feedback loops needed in DI systems. Using these tools, the main contribution of this paper is the ***Eventual C-element Theorem***, providing a rigorous proof that justifies the conclusion that the class of purely asynchronous (i.e., DI) circuits are too weak in practice—without any *a priori* constraints on the class of computations that are possible for the circuit being analyzed.

For the sake of completeness, we begin by reviewing the execution model for asynchronous circuits from [8], as well as the standard terminology used in the literature. We also review results on potential causality applied to asynchronous circuits (Section II). We examine the consequences of delay-insensitivity, and our formal approach enables us to analyze arbitrary circuits rather than those limited to having only one computation. We establish the eventual C-element theorem, and show how it can be used to argue that the set of DI circuits is limited (Section III). Finally, we discuss related work in the asynchronous design community as well as in the distributed systems literature (Section IV), and conclude in Section V.

II. Background: Asynchronous Circuits as Distributed Systems

We summarize an existing formal model for describing asynchronous circuits, and present key results in that model. We assume the basic terminology used in describing asynchronous computations from [9]. Much of this section is a summary/re-statement of material from [8].

A. Definitions and Model

We model wires by binary variables and a gate is modelled by a pair of guarded commands called production rules. More formally, a ***production rule*** (PR) over a set V of binary variables has the form $B \mapsto z\uparrow$ or $B \mapsto z\downarrow$, where $z \in V$ is a variable, and B is a propositional formula over a subset of the variables of V. A ***gate*** is a pair of production rules $B_u \mapsto z\uparrow$ and $B_d \mapsto z\downarrow$ for the same variable z. A variable y that is used in B_u or B_d is said to be an ***input*** to the gate for variable z. A ***circuit*** over a set of variables V is a set of $|V|$ gates, one per variable $z \in V$. It follows that our model describes a *closed* circuit, and not a component or module; any components connected to a circuit's variable is part of the circuit.

Consider a circuit A over a set V_A of variables. A ***configuration*** of A is an assignment $c : V_A \to \{0, 1\}$, associating a binary value with each variable of V_A. A production rule with guard B is ***enabled*** in a configuration c if its guard is true there. (This is denoted by $c \models B$.) The execution of an enabled production rule for variable x is called a ***firing***. The firing assigns a value to the output x as specified by the right hand side of the PR. If this causes a change in value it is called an ***effective firing***.

A ***computation*** of the circuit A is an infinite sequence $s : (\mathbb{N} \cup \{0\}) \to \mathcal{C}$ of configurations, in which $s(t+1)$ is obtained from $s(t)$ by firing zero or more PRs that are enabled at $s(t)$, for all $t \geq 0$.[2] A computation is finite in this setting if there exists a finite N_0 such that $s(t)$ is constant for all $t > N_0$. In natural scenarios there may be a set $\mathcal{C}_0$ of legal initial configurations for the circuit. Computations will only be allowed to start in one of these.[3] A general configuration is then ***legal*** if it is reachable in a computation that starts in an initial configuration from $\mathcal{C}_0$. The value of a variable x at time t in computation s is denoted by $s_x(t)$. We say that ***the value of*** x ***changes at time*** t ***in*** s if $s_x(t+1) \neq s_x(t)$.

Note that our definition of computations allows steps in which no variable values change, i.e., there can be times t such that $s(t+1) = s(t)$. In this case we say that there is a ***skip step*** at time t in s. Given a general computation s, we define its ***stuttering-free*** variant, denoted by $\underline{s}$, to be the execution s' obtained from s by removing all skip steps. In the Appendix we give a formal definition of $\underline{s}$ and show that if s is a computation of a circuit A, then so is $\underline{s}$. Clearly, inserting a finite number of skip steps at any point in a valid computation of A results in another valid computation of A.

The sequence defined by a computation is indexed by an integer, which plays the role of an external notion of time. (We use both t's and m's for these indexes.) These indices are used for our modeling purposes; the circuit elements do not have access to this, and it does not affect their operation. In our analysis, it will be convenient (reasoning from the outside) to distinguish the values of a variable x_i at different times t. To this end, a pair $\langle x_i, t\rangle$ is called a ***node***.

B. Potential Causality and the Past Theorem

This section, again, recalls the framework of [8]. Given a computation s, we write $\langle y, m\rangle \hookrightarrow_s \langle z, m+1\rangle$ if a PR with output z performs an effective firing at $s(m)$, and the guard of this PR contains either y or $\neg y$. For each computation s, the partial order $\preceq_s$ over variable-time nodes called *potential causality* in s is defined as the unique minimal relation satisfying the following three conditions:

Locality:	$\langle y,t\rangle \preceq_s \langle y,t'\rangle$	if $t \leq t'$;
Successor:	$\langle y,t\rangle \preceq_s \langle z,t+1\rangle$	if $\langle y,t\rangle \hookrightarrow_s \langle z,t+1\rangle$;
Transitivity:	$\langle y,t\rangle \preceq_s \langle z,t'\rangle$	if, for some $\langle x,m\rangle$, both $\langle y,t\rangle \preceq_s \langle x,m\rangle$ and $\langle x,m\rangle \preceq_s \langle z,t'\rangle$.

When $\langle y,t\rangle \preceq_s \langle z,t'\rangle$ where $y \neq z$, the definition of potential causality implies that there must be a finite sequence of variable changes from y to z that are linked by the successor relationship. This is captured by the notion of *firing chains*.

Definition 1: We say that ***there is a chain of firings from*** $\langle y,t\rangle$ ***to*** $\langle z,t'\rangle$ ***in the computation*** s if there is a sequence of variables $\langle x_1,\ldots,x_k\rangle$ with $x_k = z$, and a sequence of monotonically increasing times $\langle t_1,\ldots,t_k\rangle$ with $t \leq t_1$ and $t_k < t'$, such that $\langle y,t_1\rangle \hookrightarrow_s \langle x_1, t_1+1\rangle$ and such that $\langle x_{i-1},t_i\rangle \hookrightarrow_s \langle x_i, t_i+1\rangle$ holds for all $2 \leq i \leq k$.

The notion of firing chain is analogous to a message chain in a distributed computation. A rather straightforward consequence of the definition of $\preceq_s$ is the following lemma from [8].

Lemma 1: For all variables $y \neq z$ and times t and t', if $\langle y,t\rangle \preceq_s \langle z,t'\rangle$ then both $t < t'$ and there is a chain of firings from $\langle y,t\rangle$ to $\langle z,t'\rangle$ in s.

Potential causality gives rise to a useful notion of a node's *past*. The *past* of a node consists of the set of nodes that precede it in the current execution, according to the potential causality relation.

Definition 2 (Past): Given a computation s and a set U of variable-time nodes, we define:

$$\mathtt{past}_s(U) \triangleq \bigcup_{\langle y',m'\rangle \in U} \left\{\langle x,m\rangle : \langle x,m\rangle \preceq_s \langle y',m'\rangle\right\}.$$

A central property that we will leverage throughout this paper is the *Past Theorem* of [8].

*Theorem 1 (**Past Theorem**):* Fix an asynchronous circuit A, times $m < m'$, a computation s of A and a set U of nodes $\langle y,m'\rangle$ at time m'. Then there is a computation s' of A such that $s'(t) = s(t)$ for all times $t \leq m$, and for all variables x and times t in the range $m < t \leq m'$, we have that

(a) $s'_x(t) = s_x(t)$ if $\langle x,t\rangle \in \mathtt{past}_s(U)$ (and so, in particular, $s'_x(m') = s_x(m')$ if $\langle x,m'\rangle \in U$), and

(b) $s'_x(t) = s_x(m)$ if $\langle x,m+1\rangle \notin \mathtt{past}_s(U)$.

Theorem 1 states that given a computation s of an asynchronous circuit, we can always construct a computation of the circuit where the only changes that take place between times m and m' are those recorded in $\mathtt{past}_s(U)$.

Finally there are two properties of production rules that are used to ensure hazard-free operation. The property of non-interference captures the notion that there is no short circuit. The property of stability ensures that once a signal starts changing, it cannot be disabled—i.e. the signal transition is hazard-free. These are defined using the terminology of [8] as follows.

Definition 3 (non-interference): A pair $B_u \mapsto z\uparrow$ and $B_d \mapsto z\downarrow$ of production rules in a circuit A is ***non-interfering*** if, for every computation s of A, there is no state $s(t)$ of the computation where both $s(t) \models B_u$ and $s(t) \models B_d$ hold.

It is often possible to verify non-interference simply by checking that $B_u \wedge B_d$ is an unsatisfiable Boolean formula. The next condition is inherently a property of the circuit, and it imposes a nontrivial constraint on the circuit's design.

Definition 4 (stability): A PR $B \mapsto z\uparrow$ (respectively, $B \mapsto z\downarrow$) in a circuit A is said to be ***stable*** if for all computations s of A and for all times $t \geq 0$, if $s(t) \models B$ and $s(t+1) \models \neg B$, then $s_z(t+1) = 1$ (respectively, $s_z(t+1) = 0$).

[2] For ease of exposition, we do not assume an interleaving model; several firing events can occur at once.

[3] We can also define notions of fairness of computations, e.g., that if a PR is enabled continuously then it fires eventually, but fairness will not play a role in the particular analysis that we perform in this paper.

An immediate consequence of this definition in a form that is useful in our proofs is the following:

Lemma 2: Let x be an input to the gate for y. Suppose there is a computation s and time $t \geq 0$ such that effective firings of both x and y are enabled at $s(t)$, and firing x there disables the firing for y. Then the gate for y is unstable.

Proof: The proof is immediate from Definition 4, because firing x at $s(t)$ demonstrates that y is not stable. ■

We now embark on the main contribution of our work, namely an alternate path to characterizing the implications of delay-insensitivity in asynchronous circuits.

III. Analysis of Delay-Insensitivity

In a speed-independent (SI) circuit, all gates must be stable and non-interfering. This means that if a production rule has an enabled effective firing, then this firing stays enabled independent of the speed of any of the other gates in the system; only when the firing completes can the corresponding production rule be disabled. Hence, a SI circuit operation is robust to variations in the delays of gates. However, gates are not the only source of delay in an asynchronous circuit; wires that connect the output of a gate to the input of another gate are an important component of delay.

In a circuit, there can be a non-trivial delay in propagating a signal from one end of a wire segment to the other. The basic model from [8] cannot capture this, because it uses one variable to refer to the entire wire. To discuss propagation delays in wire segments, we introduce a name for each end of a wire segment. Suppose we name one end of the wire x and the other end x', where $x = x'$ in the initial state. Then the question arises: how do these variables relate to each other in a computation? The first observation we make is that if x changes and then remains unchanged, eventually x' will change to re-establish $x' = x$—this is the normal behavior of a wire that is actively driven. The second observation concerns the behavior when x changes more than once before x' changes. In this case, if we assume that wires can have arbitrary delay, the value of x' could change from 0 to 1 and back to 0 (or from 1 to 0 to 1, depending on the value of x) at *an arbitrary time*. This means that if a gate relies on the value of x' after the first change of x, we could construct a scenario where the gate that uses x' is unstable, since we can place a 0 to 1 or a 1 to 0 transition at an (in)appropriate time.

To capture this intuition, we use the following wire model. If the two ends of the wire are x and x', we introduce a pair of production rules $x \mapsto x'\uparrow$ and $\neg x \mapsto x'\downarrow$ to model the wire. This accounts for the first observation, (that $x' = x$ eventually), and the second observation is handled by requiring that these new production rules are *stable* if the circuit is to operate correctly in the presence of arbitrary delay on the wire segment. A speed-independent circuit is said to be ***delay-insensitive*** if modeling any set of wire segments with explicit production rules as described above preserves speed-independence, and does not materially change the computation of the circuit. We now introduce definitions that make this intuition precise. This choice of the wire model precisely matches the approach taken in [9], as well as the approach taken to model arbitrary delays on a wire fork in [8]. After a change that introduces explicit production rules to model wire segments, we relate computations in the original circuit to those in the modified circuit using the same approach as [8]. This is summarized below.

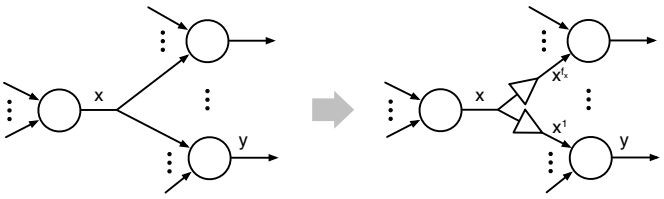


Fig. 3: Modification of the original circuit A (left) to $\tilde{A}$ (right) by the introduction of explicit buffers to model wire segments. The variable x is input to f_x gates in A.

Let s' be a computation of a circuit A' and suppose that V is a subset of the variables in A'. We define the ***restriction of*** s' ***to*** the variables in V, denoted by $s'|_V$, to be the sequence of configurations over V given by $(s'|_V)_x(t) = s'_x(t)$ for every $x \in V$ and $t \geq 0$. If V is a strict subset of the variables in A', then $s'|_V$ might have new skip steps. Hence, when comparing circuit behaviors, [8] studies how stuttering-free variants of computations of A relate to stuttering-free variants of restricted computations of A'.

Definition 5: Suppose that circuit A has variables V and circuit A' is obtained by adding variables to V and modifying the production rules in A. Let s be a computation of A and s' a computation of A'. Then s ***is*** said to be ***consistent with*** s', denoted by $s \approx s'$, if $\underline{s} = \underline{s'|_V}$.

If the circuit A' is obtained from A by adding variables, s is a computation of A and s' a computation of A', then $s \approx s'$ implies that both computations perform the same changes to all variables of A, and do so in the exact same order. If every computation of A is consistent with one of A' and every computation of A' is consistent with one of A then the two circuits are equivalent [8].

A. Delay-Insensitivity and a Successor Property

In a purely delay-insensitive circuit, the introduction of explicit production rules to model the delay along an individual wire should not impact the set of computations that are possible or the stability of the production rules used to model the wire. We start the analysis by examining the impact of introducing buffers to model wire segments.

Consider a variant $\tilde{A}$ of A where there are buffers on all branches of variable x for all variables x in A. Formally, $\tilde{A}$ is a new asynchronous circuit that is identical to A except for the following modification: for every variable x in A and for each variable v_i whose guard uses x,

1) We add a new variable x^i and two new production rules $x \mapsto x^i\uparrow$ and $\neg x \mapsto x^i\downarrow$, and
2) In the guards for v_i, we replace all occurrences of x by x^i.

This modification is illustrated in Figure 3. Since A is purely delay-insensitive, all production rules in $\tilde{A}$ are stable.

$\tilde{A}$ has all the same variables as A, plus a number of extra variables corresponding to the newly introduced buffers. The question we ask is: under what circumstances can we view computations of $\tilde{A}$ as implementing computations of the original circuit A? It is straightforward to establish that all original computations of A still could occur in $\tilde{A}$.

Lemma 3: For every computation s of A, there exists a computation s' of $\tilde{A}$ where $s \approx s'$.

Proof: To construct s', consider a firing of the gate for a variable x in s. We replicate this firing in s', and then immediately follow it with a firing of the gates for all variables x^i that correspond to the buffers introduced in $\tilde{A}$ that use x as an input. ■

Recall that if A is purely delay-insensitive, then introducing the buffers in $\tilde{A}$ preserves stability of all the gates of the circuit A in the new circuit $\tilde{A}$. We now show the consequence of requiring that the newly introduced buffers in $\tilde{A}$ are stable.

*Definition 6 (**successor property**):* Circuit A exhibits the successor property iff for every computation s and variables x and y where x is an input to the gate for y in A, if x changes at times t and $t' > t$ in s, then y changes in s at some time $t^\dagger$ that satisfies $t < t^\dagger < t'$.

Lemma 4: If A is a DI circuit then the circuit $\tilde{A}$ defined above satisfies the successor property.

Proof: Let s' be a computation of $\tilde{A}$ in which x changes at times t and $t' > t$. By Theorem 1, setting $m = t$, $m' = t'+1$ and $U = \{\langle x, t'+1\rangle\}$ we have that there is a computation s'' of $\tilde{A}$ in which x changes value at times t and t' and in which all changes of variables that occur between times t and t' are in $\mathtt{past}_{s''}(\langle x, t'+1\rangle) = \mathtt{past}_{s'}(\langle x, t'+1\rangle)$. Moreover, all changes in values in s'' between times t and t' occur in s' as well, and so it suffices to establish the existence of the desired firing chain in s''.

By construction, there is some buffer with x as input and x^k as output, where x^k is an input to the gate for y. By stability of the gate for x^k, the fact that x changed twice at times t and $t' > t$ implies that x^k must change at some intermediate time $t < t'' < t'$. Thus, by construction of s'' we have that $\langle x^k, t''+1\rangle \preceq_{s''} \langle x, t'+1\rangle$. Since $x \neq x^k$ we have by Lemma 1 that there must be a firing chain c from $\langle x^k, t''+1\rangle$ to $\langle x, t'+1\rangle$ in s''. Moreover, since x^k is the output of a buffer whose sole input is x, the fact that x^k changes at t'' means that $\langle x, t''\rangle \hookrightarrow_{s''} \langle x^k, t''+1\rangle$. Adding this firing of x^k at time t'' to the chain c, we obtain a firing chain in s'' from $\langle x, t''\rangle$ (and since $t < t''$ also from $\langle x, t+1\rangle$) to $\langle x, t'+1\rangle$ that includes a change in x^k. Since x^k is an input only to the gate for y, a firing chain from x^k to x must include a change in y. The claim follows. ■

If we define the successor set of x in circuit A as the set of variables y that have x as input in A, then Lemma 4 implies that every variable in that set must change between two changes of x. We remark that this establishes the *unique successor set* (USS) theorem from [9].

B. Feedback in Asynchronous Circuits

Requiring production rules to be stable has significant consequences. We use Theorem 1 to show that if a production rule could have an effective firing in a given configuration, and later in time the guard of the opposing production rule for the variable becomes true, then there must be a causal connection between the output of the production rule and an input to the opposing guard. More formally, we show that:

*Theorem 2 (**feedback**):* Let $B_u \mapsto z\uparrow$ and $B_d \mapsto z\downarrow$ be stable and non-interfering production rules in the circuit A, let s be a computation of A and let $t' > t \geq 0$.

1) If $s_z(t) = 0$, $s(t) \models B_u$, and $s(t') \models B_d$, then $\langle z, t+1\rangle \preceq_s \langle y, t'\rangle$ must hold for at least one variable y in the guard of B_d; and symmetrically
2) If $s_z(t) = 1$, $s(t) \models B_d$, and $s(t') \models B_u$, then $\langle z, t+1\rangle \preceq_s \langle y, t'\rangle$ must hold for at least one variable y in the guard of B_u.

Proof: We prove the first claim; the proof of the symmetric case is identical. The claim is concerned with the computation s up to, and including, time t'. Hence, we can assume without loss of generality, that the production rule $B_d \mapsto z\downarrow$ fires at time t' and so $s_z(t'+1) = 0$. Define the set

$$U = \{\langle x, t'\rangle : x \text{ is in the guard of } B_d\}.$$

If $\langle z, t+1\rangle \preceq_s \langle y, t'\rangle$ holds for no $\langle y, t'\rangle \in U$, then $\langle z, t+1\rangle \notin \mathtt{past}_s(U)$; assume this is the case. By Theorem 1 applied to U and times t and t', there is a computation s' of A in which $s'_z(m) = 0$ for all $t \leq m \leq t'$, while $s'(t) \models B_u$ and $s'(t') \models B_d$. Since $s'(t') \models B_d$ and the two production rules are non-interfering we have that $s'(t') \models \neg B_u$. Given that $s'(t) \models B_u$, there must be some time $t < m \leq t'$ such that $s'(m-1) \models B_u$ and $s'(m) \models \neg B_u$. Since we have shown that $s'(m) = 0$ at this time m, it follows by definition of stability that the production rule $B_u \mapsto z\uparrow$ is not stable in A, contradicting the assumption. The claim follows. ■

In terms of firing chains, we thus have

*Corollary 1 (**firing chain feedback**):* Let $B_u \mapsto z\uparrow$ and $B_d \mapsto z\downarrow$ be stable and non-interfering production rules. Let s be a computation of A in which the value of z changes at time t and then changes again at time $t' > t$. Then there is a firing chain from $\langle z, t+1\rangle$ to $\langle z, t'+1\rangle$ in s.

Proof: This is a direct consequence of Theorem 2 and Lemma 1. ■

In a precise sense, Theorem 2 and Corollary 1 formalize the fact that before the identity of the enabled guard for a variable can change, the newly enabled guard must receive an acknowledgment that the earlier enabled guard fired successfully. Moreover, this acknowledgement must involve a chain of firings originating after the first change in the variable. Corollary 1 illustrates the value of using potential causality to reason about circuits.

C. The Eventual C-element Theorem

A gate is a C-element if between successive effective firings, all of its *inputs* must flip value. The main technical claim of [9] roughly states that if a circuit A has a computation in which every gate fires at least 3 times, then all of A's gates are C-elements. Since DI circuits satisfy the successor property by Lemma 4, showing that their gates are C-elements amounts

to showing that they also satisfy a corresponding *predecessor property*, defined below.

*Definition 7 (**predecessor property**):* Circuit A exhibits the predecessor property iff for every computation s and variables x and y where x is an input to gate y in A, if y changes at times t and $t' > t$ in s, then there exists a firing chain from $\langle y, t+1\rangle$ to $\langle y, t'+1\rangle$ in s that includes a change in x.

It is standard to consider a circuit A as inducing a directed graph whose nodes are the variables of A and where there is an edge between x and y precisely if x is an input to y's gate G_y in A. We shall use $\delta(x, w)$ to denote the shortest distance (in number of edges) from x to w in this graph.

The feedback property captured by Corollary 1 can be used to show that many inputs of a gate in a DI circuit must be on a feedback loop from the gate's variable y:

Lemma 5: Let x and y be variables of a DI asynchronous circuit A, with x being an input to the gate for y. If there is a computation s of A in which x changes value at least twice, then there must be a directed path from y to x in the circuit A.

Proof: Fix a DI asynchronous circuit A. Let A' be the circuit obtained from A by introducing an explicit buffer from x to x' in order to model the wire segment from x to y. Since A is DI, A' is speed-independent. Given the computation s, we know by Lemma 3 that there is a computation s' of A' such that $s \approx s'$, and where x' always fires immediately after x. Since x changes twice in computation s, both x and x' must change twice in computation s'. Assume that x' changes at times t and $t' > t$ in s'. Since the newly introduced buffer in A' is stable and non-interfering, Corollary 1 implies there is a firing chain from $\langle x', t+1\rangle$ to $\langle x', t'+1\rangle$ in computation s'. This firing chain occurs along a directed path from x' to y to x and back to x' in A', since the only input to x' is x, and x' is only used as an input by the gate for y. Hence, the claimed directed path from y to x must exist in A. ■

We define the ***feedback inputs*** of a gate G_x as the inputs that are on a feedback loop (a directed path) from its output x. By Lemma 5, every input of x that changes value at least twice in some computation of a DI circuit A is a feedback input of G_x. By Lemma 4, over an interval in which x changes twice, then every variable z such that $\delta(x, z) = 1$ must change at least once. If x changes three times, then every z such that $\delta(x, z) = 1$ must change twice, which means every variable w for which $\delta(z, w) = 1$ (and hence $\delta(x, w) = 2$) must change once. Extending this argument, it is easy to see that if, for some computation s of A the gate G_x fires at least $\delta(x, y) + 2$ times, then y fires at least twice in s.

In the rest of this section, A is assumed to be a DI circuit. Let y be a feedback input of G_x in A. We now show that in every computation of A, from some time on G_x and G_y must fire in an alternating order, if they fire at all.

Lemma 6: Suppose that y is a feedback input of a gate G_x in the DI asynchronous circuit A, and let s be a computation of A. Then there is a time $t_{s,y}$ after which the gates G_y and G_x can only fire in alternating order. Furthermore, G_y and G_x cannot fail to fire in alternating order more than $\delta(x, y)-1$ many times in s.

Proof: Let y be a feedback variable of G_x in A. Since y is a feedback input of G_x, there is a cycle $\langle x, x_{i_1}, \ldots, x_{i_d}, y, x\rangle$ in A, with $d = \delta(x, y) - 1$. Denote $W = \{x, x_{i_1}, \ldots, x_{i_d}, y\}$. For every $z \in W$, we denote by z^+ the successor of z in the cycle, and by z^- its predecessor. Fix a computation s of A. The successor property captured by Lemma 4 implies that a necessary condition for $z \in W$ to change value at $s(m)$ is that z^+ changed value since the last time $m' < m$ at which z changed value in s (provided that such an earlier time m' exists).

For every time $m \geq 0$, define

$$F(m) = \big\{ z \in W : \text{ for every } m' < m, \text{ if } z \text{ changed value at } s(m'), \text{ then } z^+ \text{ changed value in } s \text{ at least once between } m'+1 \text{ and } m-1. \big\}$$

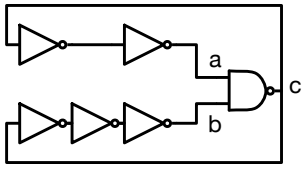
Thus, $F(m)$ keeps track of the variables in the cycle that are allowed to fire at time m, as far as the successor property is concerned. Clearly, $F(0) = W$. If $z \in W$ changes value at a given time m, then its predecessor in the cycle will be added to F, unless it is in $F(m)$ already. Let $\mathsf{changing}(m)$ be the set of variables in W that change value at $s(m)$. The rule by which F is updated is:

$$F(m+1) = \Big(F(m) \cup \{z^- : z \in \mathsf{changing}(m)\}\Big) \setminus \mathsf{changing}(m). \tag{1}$$

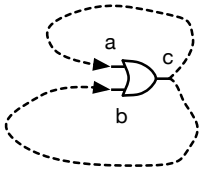
The variables z that change are removed from F and their predecessors z^- are added. (Except that if the predecessor z^- also changes at m, it will need to wait for a later change in the original variable z before z^- can change again.) Equation (1) implies that F cannot grow in size: $|F(m+1)| \leq |F(m)|$ for all $m \geq 0$. Indeed, $|F(\cdot)|$ remains the same only if none of the predecessors z^- are in $F(m)$. Moreover, if $F(m)$ is a singleton, then $F(m')$ is a singleton for all $m' > m$, since z can change value only if $F(m) = \{z\}$, but then we have that $F(m+1) = \{z^-\}$, since $z^- \notin F(m)$.

So suppose that some $z \in W$ changes value twice, at $s(m)$ and at $s(m')$ with $m < m'$, and its predecessor z^- does not change value in the interim. We will show that $|F(m'+1)| < |F(m)|$. If its predecessor z^- also changes value at $s(m)$, then both z and z^- are removed from $F(m)$ and at best $(z^-)^-$ is added on their account, so $|F(m+1)| \leq |F(m)| - 1 < |F(m)|$. By monotonicity of $|F(\cdot)|$ we have that $|F(m'+1)| \leq |F(m+1)| < |F(m)|$, as claimed, since $m' > m$. Otherwise z^- does not change at $s(m)$ and by Equation (1) we have that $z^- \in F(m+1)$ because z changed at $s(m)$. Moreover, $z^- \in F(m')$ since z^- does not change value between $m+1$ and m'. Indeed, both $z^-, z \in F(m')$ since $z \in \mathsf{changing}(m')$. Equation (1) now implies that $|F(m'+1)| < |F(m')|$, and by monotonicity of $|F(\cdot)|$ we have that $|F(m'+1)| < |F(m')| \leq |F(m)|$, establishing the claim.

It follows that the total number of times at which a variable in W changes value twice in s without its predecessor changing value is no greater than $d = |W|-2$, and hence finite. In particular, since $x \in W$, the number of times at which y may fail to change value between two consecutive changes in x is at most $d = \delta(x, y) - 1$. Lemma 4 implies that between every two changes in y there must be a change in its successor x. Thus, in the infinite computation s, the gates G_x and G_y fail to fire in alternating order at most $\delta(x, y) - 1$ times. Hence,



(a) A circuit that displays both finite and infinite computations. Initially a=b=1, c=1, and the inverters leading to b have values that are consistent with b=1.



(b) A circuit that contains a two-input OR gate for c, where both inputs a and b are feedback inputs of the gate.

Fig. 4: Applying the eventual C-element theorem to some example circuits.

there is a time $t_{s,y}$ after which gates G_x and G_y can only fire in alternating order. ■

Given a gate for x and an input variable y to the gate, if changes in x and y alternate, then the gate for x must logically behave in a manner that is equivalent to production rules of the form $y \wedge B \mapsto x\uparrow$ and $\neg y \wedge B' \mapsto x\downarrow$, or $\neg y \wedge B \mapsto x\uparrow$ and $y \wedge B' \mapsto x\downarrow$ for appropriate guards B and B'. In this case we say that the gate for x acts as a C-element with respect to the input y. The reason for using this terminology is that if this were true for every input to the gate for x, then the entire gate behaves as if it were a C-element. Given Lemma 6, we can prove the following result.

*Theorem 3 (**Eventual C-element**):* Let A be a DI asynchronous circuit.

(i) For every computation s of A, there is a time T after which every gate that fires acts as a C-element with respect to its feedback inputs; and

(ii) There is a uniform finite bound $b = b(A) < |V_A|^2$ on the number of violations of the predecessor property that can occur in any computation of A. After at most b violations, every gate in A acts as a C-element with respect to its feedback inputs.

Proof: We first show that, for every variable y of A there is a time T_y after which gate G_y acts as a C-element in s as claimed. Fix a variable y of A, and let X be the set of feedback inputs of G_y. Clearly, X is finite. For every $x \in X$, we have by Lemma 6 that there is a time $T_{s,x}$ after which gates G_y and G_x can only fire in alternating order. Define $T_y \triangleq \max_{x \in X} T_{s,x}$. It follows that after time T_y every firing of G_y must be preceded by a single change in value of each of the variables in X. Thus, G_y acts as a C-element with respect to its feedback variables X. Hence, by setting $T \triangleq \max_{y \in V} T_y$, we have that every gate that fires after time T in the computation s of A acts as a C-element with respect to its feedback inputs.

By Lemma 6, number of violations of the predecessor property between an input and output of a gate is less that the shortest distance between the two in the circuit graph, and this, in turn is bounded by the number of gates V_A. Since each gate could exhibit such violations, the total number of violations in the circuit is bounded by V_A^2. ■

Notice that if a computation s of a DI circuit A repeats a configuration c, then all firings in s between the two repetitions are C-element firings. Otherwise the circuit also has a computation s' in which configuration c can repeat more than $b = b(A)$ times, resulting in more than b violations, contradicting Theorem 3.

Theorem 3 also severely restricts the gates that can appear in a DI circuit. As an example, suppose a standard two-input OR gate (production rules $a \vee b \mapsto c\uparrow$ and $\neg a \wedge \neg b \mapsto c\downarrow$) is part of a DI circuit, as shown in Figure 4b. If a and b both change twice, then both a and b must be feedback inputs to c (Lemma 5). If both a and b are feedback inputs, then in any computation, eventually the gate must behave as a C-element with respect to both inputs. This means that eventually a and b changes must alternate with c—which is impossible since if a and b are both 0, then a change in either one of them can cause c to change. Hence, such an OR gate cannot appear in a DI circuit. Alternatively, only one of the inputs could be a feedback input, in which case the other input can change at most once in *any* computation. A similar argument can be used to show an AND gate cannot appear in a DI circuit unless only one of its inputs can change more than once in a computation.

Figure 4a shows an example circuit whose computation can be either finite or infinite, depending on the relative order in which a and b change in response to a change in c. If c becomes 1 leading to b becoming 0 *before* c changes again, then the circuit will enter a stable configuration. If b never becomes 0 (a possible execution), then c will oscillate. This circuit is obviously not DI, and is ruled out by Theorem 3 as well since the gate for c is not a C-element with respect to its feedback inputs a and b.

Finally, we revisit our two initial examples (Figures 1 and 2). We can conclude that Figure 1 is not DI directly by Lemma 5 since y has no feedback to x even though x can change twice. In Figure 2, all the gates are C-elements with respect to all their inputs, and hence it is consistent with DI behavior.

IV. Related Work

Satisfaction of a set of delay requirements is an essential part of the correct operation of general asynchronous circuits. Previous work in trace theory viewed delay insensitivity as a constraint on the set of possible execution traces [10], but this was not translated into any constraints on the gates used to implement the circuits. A variable that is used as the input to multiple gates has a wire fork on its output, where each fork corresponds to the wire segment connecting the variable to a separate gate. The fact that researchers who were designing

purely delay insensitive circuits were in fact making a timing assumption on wire forks was first highlighted in [9], which introduced several important notions such as the acknowledgment theorem and the unique successor set property. That paper also introduced the notion of isochronic forks, which are wire forks where the delays on different wire segments of the fork are bounded relative to each other. The isochronic fork assumption is a mild timing assumption, and it was a way to increase the computational power of the class of purely delay-insensitive circuits. [7] showed that the introduction of the isochronic fork is sufficient to enable the construction of circuits that can implement any computable function (modulo finite memory) while respecting the constraints of CMOS technology. The formalism used in this paper as well as the Past Theorem for asynchronous circuits was introduced by [8], which also precisely characterized the isochronic fork timing assumption. An outline of a different proof of the same result based on the notion of transition causality had previously been provided by Keller et al. [5].

As far as timing and coordination are concerned, there is a close connection between asynchronous circuits and asynchronous distributed systems. The potential causality relation $\preceq_s$ is a variant of Lamport's *happened before* relation from his seminal paper [6]. The two differ in the Successor step, which in Lamport's case relates the sending of a message over an asynchronous channel to its delivery. While $\preceq_s$ corresponds to the existence of chains of firings as captured in Lemma 1, Lamport's relation corresponds to the existence of message chains between events at different sites. Message chains are the essential tool for information flow and coordinating actions in asynchronous distributed systems, just as firing chains are for asynchronous circuits. Indeed, Chandy and Misra showed in [2] that the only way a site can know about a change that occurs at another site of an asynchronous system is by way of a message chain. In a precise sense, firing chains can be shown to play the analogous role in asynchronous circuits. While we do not introduce a formal definition of what a node $\langle x, t\rangle$ ***knows*** about the circuit (this can be done using the framework of [4], [3] by viewing the circuit as an asynchronous system), it is interesting to consider our analysis in these terms. The fact that a firing chain is necessary in order to inform a node about the change of value of a different variable is essentially captured by Theorem 1. In a precise sense, a gate's stability implies that if an effective firing is enabled, then its guard can not be disabled unless the firing itself has occurred. In order to ensure this, however, the gate's input variables must know about the firing. In particular, the firing must be in the past of the input variables. Since the output variable is distinct from the input variables, Lemma 1 implies that this requires a chain of firings. Theorem 2 can be seen as formalizing this intuition. We remark that the Past Theorem (Theorem 1) is a simplified version of a theorem proven in [1] for a message-based distributed systems context with time bounds.

The notion of potential causality ($\preceq_s$) used in our analysis, is adopted without change from [8]. It is used in an essential way by [8] in the proof of the Past Theorem (Theorem 1), which plays a significant role in our analysis of DI circuits. Our notion of potential causality resembles Martin's pre-order relation over transitions [9]. They differ when production rules have disjunctions, because potential causality relates all inputs to a production rule to its output, whereas Martin's pre-order relation only relates the output to those inputs that cause the output's value to change. The proof of the Past Theorem from [8] does not go through for Martin's transition causality. Another related paper that uses a transition causality model similar to the one in [9] is work by Keller et al. [5].

V. Conclusions

Martin's C-element theorem has been extremely influential in the development of asynchronous circuit design. We showed both DI and non-DI circuit examples that fall outside the class of circuits that can be analyzed by that result. Given its significance, establishing the practical consequences of the C-element theorem for a much wider class of asynchronous circuits is called for.

We developed an alternate strategy for analyzing delay-insensitive asynchronous circuits. Using our approach, we stated and rigorously established the Eventual C-element Theorem, which applies to any closed circuit where gates have single outputs, and that similarly implies that the class of delay insensitive circuits is extremely limited.

Following in the footsteps of [8], our approach leverages concurrency-based tools and techniques inspired by distributed systems research. In addition to the formalism, the proof of our main result involves a novel combinatorial argument that shows that the successor property implies that a predecessor property will eventually hold (see Lemma 6). This is the key to establishing our Eventual C-element Theorem, and there is no analogue to this line of reasoning in the asynchronous circuits literature on delay-insensitivity.

References

[1] Ido Ben-Zvi and Yoram Moses. Beyond lamport's *Happened-before*: On time bounds and the ordering of events in distributed systems. *Journal of the ACM*, 61(2):13:1–13:26, 2014.

[2] K. Mani Chandy and Jay Misra. How processes learn. *Distributed Computing*, 1(1):40–52, 1986.

[3] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about Knowledge*. MIT Press, 2003.

[4] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990.

[5] Sean Keller, Michael Katelman, and Alain J. Martin. A necessary and sufficient timing assumption for speed-independent circuits. In *Proceedings of the 15th IEEE International Symposium on Asynchronous Circuits & Systems (ASYNC '09)*, pages 65–76, 2009.

[6] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[7] Rajit Manohar and Alain J. Martin. Quasi delay-insensitive circuits are Turing-complete. In *Proceedings of the 2nd IEEE International Symposium on Asynchronous Circuits & Systems (ASYNC '96)*, 1996.

[8] Rajit Manohar and Yoram Moses. Analyzing isochronic forks with potential causality. In *Proceedings of the 21st IEEE International Symposium on Asynchronous Circuits & Systems (ASYNC '15)*, 2015.

[9] Alain J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In W. J. Dally, editor, *Proceedings of the 6th MIT conference on VLSI*, pages 263–278. MIT Press, 1990.

[10] Huub M. J. L. Schols. A formalisation of the foam rubber wrapper principle. Master's thesis, Dept. of Math. and C.S., Eindhoven Univ. of Technology, 1985.

[11] Nikolai Starodoubtsev, Sergei Bystrov, and Alex Yakovlev. Monotonic circuits with complete acknowledgement. In *Proceedings of the 9th IEEE International Symposium on Asynchronous Circuits & Systems (ASYNC '03)*, pages 98–108, 2003.