University Library



A gateway to Melbourne's research publications

Minerva Access is the Institutional Repository of The University of Melbourne

Author/s:
Gange, G;Horsfall, B;Naish, L;Sondergaard, H

Title:
Four-Valued Reasoning and Cyclic Circuits

Date:
2014-07-01

Citation:
Gange, G., Horsfall, B., Naish, L. & Sondergaard, H. (2014). Four-Valued Reasoning and Cyclic Circuits. IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, 33 (7), pp.1003-1016. https://doi.org/10.1109/TCAD.2014.2304176.

Persistent Link:
https://hdl.handle.net/11343/268030

Four-Valued Reasoning and Cyclic Circuits

Graeme Gange, Benjamin Horsfall, Lee Naish and Harald Søndergaard

***Abstract*—Allowing cycles in a logic circuit can be advantageous, for example, by reducing the number of gates required to implement a given Boolean function, or set of functions. However, a cyclic circuit may easily be ill-behaved. For instance it may have some output wire oscillate instead of reaching a steady state. Propositional three-valued logic has long been used in tests for well-behaviour of cyclic circuits: a symbolic evaluation method known as ternary analysis provides one criterion for well-behaviour under certain assumptions about wire and gate delay. We revisit ternary analysis and argue for the use of four truth values. The fourth truth value allows for the distinction of "undefined" and "underspecified" behaviour. Ability to underspecify behaviour is useful, because, in a quest for smaller circuits, an implementor can capitalize on degrees of freedom offered in the specification. Moreover, a fourth truth value is attractive because, rather than complicating (ternary) circuit analysis, it introduces a pleasant symmetry, in the form of contra-duality, as well as providing a convenient framework for manipulating specifications. We use this symmetry to provide fixed point results that clarify how two-, three-, and four-valued analyses are related, and to explain some observations about ternary analysis.**

***Index Terms*—Boolean functions, circuit optimization, combinational circuits, multivalued logic.**

I. Introduction

The close correspondence between electronic circuits and Boolean formulas has been utilized since the 1930s [1]. A circuit can be represented as a set of Boolean equations. For example, we can capture the 4-bit full adder in Fig. 1 with the equations

$$\begin{array}{lcllcl} c_0 &=& \langle x_0, y_0, 0\rangle & z_0 &=& x_0 \oplus y_0 \oplus 0 \\ c_1 &=& \langle x_1, y_1, c_0\rangle & z_1 &=& x_1 \oplus y_1 \oplus c_0 \\ c_2 &=& \langle x_2, y_2, c_1\rangle & z_2 &=& x_2 \oplus y_2 \oplus c_1 \\ c_3 &=& \langle x_3, y_3, c_2\rangle & z_3 &=& x_3 \oplus y_3 \oplus c_2 \end{array} \quad (1)$$

where $\langle x, y, z\rangle$ is the "median" function $(x \wedge y) \vee (y \wedge z) \vee (x \wedge z)$ and $\oplus$ denotes exclusive or.

In systems of equations such as (1), variables that only occur on the right-hand sides represent input. Those on left-hand sides can be considered output, although in many cases we want to distinguish between variables that are truly output and those that simply serve to hold intermediate results (representing internal wires). For the adder, for example, it is natural to consider only z_i as output—the "carry" variables c_i are internal.

This paper is concerned with circuits that contain cycles. Traditionally, a circuit which is intended to implement a function (a "combinational" circuit) is acyclic. However, the utility of cyclic circuits has been recognized since the computation laboratory at Harvard produced its circuit compendium in 1951 [2]. Although the introduction of feedback loops in circuits may lead to undesirable or ill-defined behaviour, it is well known that, in some cases, it can produce well-behaved combinational circuits consisting of fewer components than equivalent acyclic circuits.

This work was supported in part by the ARC through grant DP110102579.

The authors are with the Department of Computing and Information Systems, The University of Melbourne, Victoria 3010, Australia.

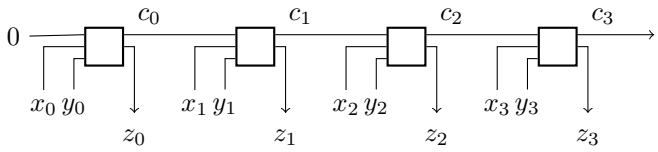


Fig. 1. A four-bit adder

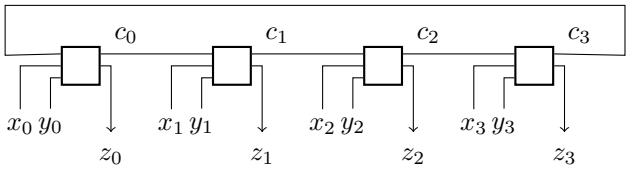


Fig. 2. An end-around adder

In terms of Boolean equations, permitting cycles simply means allowing for recursively defined Boolean functions. Fig. 2 shows a 4-bit end-around carry adder (from [2], page 159), captured by replacing the definitions of c_0 and z_0 in (1) by

$$c_0 \;=\; \langle x_0, y_0, c_3\rangle \qquad z_0 \;=\; x_0 \oplus y_0 \oplus c_3$$

resulting in a set of mutually recursive definitions. The circuit is similar to the previous one, except the carry from the most significant bit in the sum is fed back as "carry in" to the entire addition (a design which was aimed at simplifying subtraction in a one's complement number system).

The use of recursion raises questions of well-definedness, just as the introduction of a cycle in the circuit raises questions of well-behavedness. There are well-known fixed point theorems that establish conditions under which solutions to a set of equations are guaranteed to exist. The question of a cyclic circuit's behaviour is more complex, since a circuit's behaviour depends on physical properties of its components, but useful criteria for well-behavedness (under reasonable assumptions about the hardware) have been known for half a century [3].

Kautz [4] discusses the end-around carry circuit from Fig. 2 and notes that it "contains closed-loop storage of an undesirable type". Namely, if $x_i \neq y_i$ for all $i \in \{0, 1, 2, 3\}$, there is one stable state in which the four internal wires c_i all are 0, and the output $z = 1111$. At the same time, another stable state has the internal wires all 1, and $z = 0000$. Hence the output depends not only on input, but occasionally also on the initial state of the circuit's internal wires. (Of course 0000 and

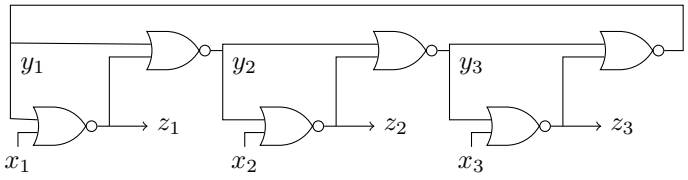


Fig. 3. Kautz's circuit

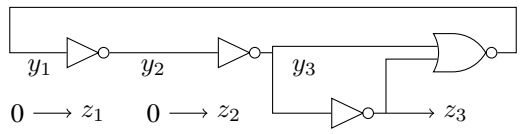


Fig. 4. Kautz's circuit specialized for input 110

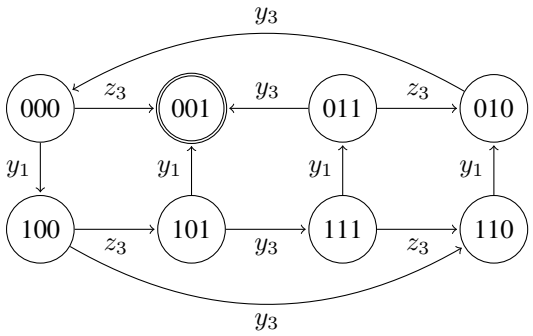


Fig. 5. Possible state transitions for the specialized circuit

1111 are two different one's complement representations of the same number, zero, so this may or may not be considered a problem, depending on the application.)

Kautz [4] proposes a small cyclic circuit, shown in Fig. 3, as a component of the adder. The circuit implements three functions, defined by

$$\begin{aligned} z_1 &= \neg x_1 \wedge (\neg x_3 \vee x_2) \\ z_2 &= \neg x_2 \wedge (\neg x_1 \vee x_3) \\ z_3 &= \neg x_3 \wedge (\neg x_2 \vee x_1) \end{aligned}$$

Kautz points out that the internal wires must oscillate in one input case, $x_1 = x_2 = x_3 = 1$, but that the outputs nevertheless are uniquely determined ($z_1 = z_2 = z_3 = 0$) in this case; for other inputs, there is exactly one stable state.

If we represent a circuit by a set of recursive Boolean function definitions and proceed to reason about solutions to that set, we implicitly assume a behavioural model in which neither gates nor wires incur any delay. The physical reality of course is different. To see how the delay model affects reasoning about circuits, consider the circuit in Fig. 3. To explore the range of possible behaviours of this circuit, let us fix some input combination, say, $x_1 = x_2 = 1$ and $x_3 = 0$. This effectively turns the circuit into the one shown in Fig. 4. The finite-state automaton in Fig. 5 shows all possible state transitions for this system, assuming that no two wires change state simultaneously. When reasoning about possible state changes, y_2 is of little interest, as it connects two inverters which together simply provide some delay. So our interest is confined to the possible states (y_1, y_3, z_3). In Fig. 5, an automaton state labeled $b_1b_2b_3$ stands for $y_1 = b_1, y_3 = b_2, z_3 = b_3$, and each transition is labeled with the name of the wire that changes value. For example, two transitions are possible from state 000. If the nor gate fires next, y_1 changes and the new state is 100. If instead the bottom-most inverter fires, z_3 changes to 1 and the stable state 001 is reached. Transitions to the same state are ignored.

From the automaton it can be seen that certain sequences of transitions lead to the stable state 001. However, as can also be seen, if the inverter from which z_3 emanates is sufficiently slow, we may see a perpetual transition sequence such as 000–100–101–111–110–010–000–…, in which y_1, y_3 and z_3 all oscillate. Thus, for a delay model which allows a range of delays for a given type of gate, Kautz's circuit is not well-behaved.

Brzozowski and Seger [5], [6] discuss delay models and their impact on methods for reasoning about circuits. Building on Brzozowski and Seger's work, as well as Berry's [7], Shiple [8] builds on this in a careful study of different notions of circuit well-behavedness and allied algorithms. A finer taxonomy for circuit classification is offered by Mendler, Shiple and Berry [9].

In this paper we discuss the use of four-valued logic for specification and analysis of cyclic circuits. Three-valued logic is helpful as a tool for analysis; so-called ternary analysis has a venerable history. However, as also pointed out by Mendler, Shiple and Berry [9], the third value has been used to denote a variety of system behaviours (oscillation, instability, transience, …), and we can add a variety of epistemic states (undefinedness, irrelevance, absence of information, …). We favour the use of four truth values because this conveniently allows the distinction between "undefined" behaviour and "underspecification" and because it leads to a simple presentation of circuit analysis. Rather than complicating analysis, the use of four values introduces a symmetry which helps explain otherwise obscure aspects of ternary analysis. Importantly, it leads to *compositional* specifications, as discussed in Section IV.

We denote the two non-classical truth values by $\top$ and $\bot$. In the context of circuit specification and implementation we associate "underspecified" with $\top$ and "undefined" with $\bot$. The former has to do with designer intention, the latter with circuit behaviour. Distinguishing input vectors that fall outside the care set from those causing bad behaviour/output is clearly desirable. Moreover, under-specification is useful in synthesis, because the (automated) minimization of circuits can capitalize on degrees of freedom offered in a specification. For example, take some set $\{f, g, h\}$ of functions whose definitions have been synthesised without regard to care sets. Say (recursive) function definitions have been synthesised as follows:

$$\begin{aligned} f &= ((u \wedge g) \vee x) \wedge z \\ g &= (u \vee h) \wedge y \\ h &= (\neg x \vee z) \wedge (y \vee f) \end{aligned}$$

by which the three functions are well-defined. However, suppose that f's care set excludes input combinations in which

both u and x are 0. The freedom offered by this information allows a set of simpler definitions (that is, smaller circuits):

$$\begin{aligned} f &= (x \vee y) \wedge z \\ g &= (u \vee h) \wedge y \\ h &= (\neg x \wedge y) \vee f \end{aligned}$$

The functions g and h thus defined are no different from those given by the previous definition, and f has changed only in that it now yields 1 on input $u\ x\ y\ z = 0011$.

We shall say more about the four truth values in Section III-B. They find uses in different areas including artificial intelligence [10], transition system analysis [11], and programming language semantics [12], [13]. The interpretation of the non-classical truth values may differ, but common across the applications is the clear distinction between, on the one hand, degrees of truth, and on the other, information content. While 0 offers the least degree of truth, $\perp$ holds the least amount of information.

Three- and four-valued logic can be encoded in Boolean logic. It is natural to ask why four-valued (or three-valued) logic would then be considered, as Boolean logic is simpler and offers identical expressive power. The answer is that multi-valued logic provides a more abstract way of representing the behaviour and specification of circuits. Boolean logic can be used to encode multi-valued logics, but committing to a particular encoding or representation is often a mistake. Generally, the key to efficiently solving a problem is finding a clever representation. There are many ways of encoding multi-valued logic using Boolean logic, along with more direct representations. Here we use multiple representations as it suits us, and though we make no contributions to finding more efficient representations, neither do we close off any options. An even more important reason to avoid encoding using Boolean logic is that it obfuscates important aspects of the domain. In particular, there is a natural "information" ordering in three-valued logic which is the key to the elegance, insights and popularity of ternary analysis. This ordering extends naturally to four values, similarly leading to elegance, insights and, we hope, eventual popularity.

The main contributions offered in this paper are:

- A simple formulation, in a single framework, of concepts such as specification, implementation, irrelevance and undefinedness (Section IV).
- A single concise formulation of 2-, 3- and 4-valued circuit analysis (Section V).
- Various propositions that relate 2-, 3- and 4-valued fixed points and explain certain aspects of circuit analysis *qua* fixed point finding (Section VI).

Section II summarises related work, most of it coming from the area of combinational circuit analysis. Section III provides some necessary background from order and fixed point theory, and we introduce four-valued logic over a bilattice. We shall only need rather basic concepts from the theories, but the aim of the section is to make the paper largely self-contained. Section IV discusses how we apply four-valued logic to specification and analysis of circuits and what it means to implement a four-valued specification. In Section V we revisit, formally, the traditional ternary analysis, rephrasing it for the four-valued setting. In Section VI we explore the relations between two-, three-, and four-valued fixed points, and we identify the reason why two-valued reasoning sometimes work when it could not be expected to. We also utilise the symmetry offered by the use of four values to explain why certain ternary analysis results come in pairs. Section VII concludes the paper.

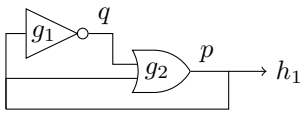


Fig. 6. This circuit fails the ternary test, but every initial state converges to $(p, q) = (1, 0)$ under the up-bounded inertial delay model [6] (assuming the signal from p to g_1 and g_2 are a single delay element; if they can be updated independently, oscillation may occur under $(p, q) = (0, 0)$).

II. Background and Related Work

The literature contains much discussion of the behaviour of cyclic circuits, and many different formulations of what it means for a circuit to be *well behaved*.

The earliest work, due to Eichelberger [3] considered detection of *static hazards* (also known as *glitches*), where a change of inputs from one value to another results in a spurious pulse for some output value. Starting from a stable state, Eichelberger's *Algorithm A* sets the changed inputs to an indeterminate value, propagating uncertainty through the circuit. His *Algorithm B* then sets the inputs to their final values, and computes the resulting final state.

This analysis is similar to the *General Multiple Winner (GMW)* analysis, which simulates the possible evaluation of a circuit given a change of one or more inputs. Brzozowski and Seger [5] show that, for a given input change from a stable total state, Eichelberger's analysis is consistent with the set of states reachable according to GMW analysis with gate and wire delays.

Eichelberger's analysis assumes that the circuit begins in a stable state; it does not address the problem of determining whether such a state exists, or will necessarily be reached. GMW analysis can handle this case, but is potentially exponential even for a fixed input assignment. A number of formulations of *combinationality* have been proposed to analyse the behaviour of cyclic circuits without suffering these drawbacks. The most common formulation is that of Malik [14], which requires the circuit to converge to a unique stable state, independent of the initial value of internal wires. Rather than simulating each input separately, Malik assumes the circuit starts in an indeterminate state, and uses a symbolic ternary simulation to reason about the set of inputs that are guaranteed to reach a final stable state. Note that this formulation of combinationality is procedural; a circuit is said to be combinational if and only if it passes the ternary analysis. There are circuits that fail the ternary analysis but nevertheless converge for all inputs (such as the circuit shown in Fig. 6). In subsequent sections, when we use the term "combinational", we have in mind Malik's notion.

Shiple [8] refers to Malik's formulation as *combinational output-stability*, and discusses improved iteration strategies for computing the ternary fixpoint. He also describes two other notions of well-behavedness, *sequential output-stability* and *constructivity*, both of which extend the notion of combinationality to circuits containing memory elements.

Ternary analysis is typically justified by assuming circuits follow the up-bounded inertial delay model [6], which assumes that (a) an unstable delay element will be updated after some bounded length of time, and (b) only unstable delay elements may change – that is, delay elements have no memory. Recently, however, Mendler *et al.* [9] observed that, under this delay model, the ternary analysis is imprecise in cases where some gates or wires do not contain delays (such as the case shown in Fig. 6); they instead define constructivity under a more conservative non-inertial delay model which eliminates the memorylessness requirement (for which ternary analysis is exact), and extend the ternary analysis to provide upper bounds on stabilization times.

Namjoshi and Kurshan [15] consider extending notions of constructivity to non-Boolean domains, and give a formulation in terms of satisfiability testing. This formulation essentially asks if there exists some three-valued fixed point which has a two-valued input assignment, but some wire evaluates to $\perp$. They prove that in the Boolean case, this is equivalent to the formulation used by Malik as well as by Shiple, Berry and Touati [16]. They note that for the domains they consider, testing constructivity is PSPACE-complete.

Alternative notions of well-behavedness, discussed by Halbwachs and Maraninchi [17], are strong and weak consistency. In this framework, a cyclic circuit is represented as a set of simultaneous equations. A circuit is considered to be *strongly consistent* iff, for any set of inputs, there exists a unique satisfying assignment to all internal and output wires. It is said to be *weakly consistent* iff, for any set of inputs, the set of outputs is uniquely determined. As observed by Namjoshi and Kurshan [15], the uniqueness criterion used for this formulation does not necessarily preclude oscillation in the corresponding circuit.

We are not aware of complexity results for strong and weak consistency. We note, however, that given an instance (X, W, φ) where φ is a (recursive) circuit definition, with inputs $X = \langle x_1, \ldots, x_k \rangle$ and set W of wires, strong consistency (SC) asks about the truth of the following formula:

$$\forall X \exists W \ \varphi(X, W)$$
$$\wedge \ \forall X \forall W \forall W' \ (\neg\varphi(X, W) \vee \neg\varphi(X, W') \vee W = W')$$

The first conjunct expresses the satisfiability of the system of equations, and the second expresses uniqueness. Deciding the truth of quantified formulae of this form is in Π_2^p [18]. Weak consistency (WC) is similarly in Π_2^p, as can be shown by a simple reduction. Briefly, we can reduce SC to WC by duplicating, in an SC instance, each internal variable as an output.

Jiang *et al.* [19] consider a function-level notion of combinationality. However, unlike the above formulations, all function definitions are assumed to be evaluated simultaneously (and instantaneously). This makes the evolution of the circuit from a given state purely deterministic. A circuit is said to be *functionally combinational* if, in this framework, the circuit will always eventually reach a state with stable output values. It is *stably* functionally combinational if both the internal wire and output values eventually stabilize. Testing if a circuit is functionally combinational in this sense is PSPACE-complete [19].

For the purposes of (cyclic) circuit synthesis, these analyses are traditionally used in a "generate and test" manner, where a circuit with cycles is generated, and then evaluated to check for combinationality [20]. Based on work of Lee *et al.* [21] addressing the acyclic case, Backes and Riedel [22] adapt interpolant-based synthesis techniques to the direct construction of combinational cyclic circuits. To this end, they develop another SAT-based formulation of combinationality, but expressed at the function level (rather than the gate level, as in the case of [15]). This function-level formulation essentially treats each recursive function definition as an atomic unit, which is evaluated completely or not at all. Unlike the formulation of Jiang *et al.* [19], however, this formulation permits delays between functional units. As observed by Jiang *et al.*, function-level combinationality (under either formulation) does not necessarily guarantee desirable behaviour of a physical implementation of the circuit.

III. Mathematical Preliminaries

For logical connectives we use $\wedge$ to denote conjunction ('and'), $\vee$ to denote disjunction ('or'), $\neg$ to denote negation, and $\Rightarrow$ to denote implication.

We use $\vec{x}$ to denote a *vector*, or finite tuple, $(x_0, x_1, \ldots, x_n)$. Let $\mathbb{N} = \{0, 1, 2 \ldots\}$. An ω-sequence (or just *sequence*) of X-elements $[x_0, x_1, \ldots] = [x_i \mid i \in \mathbb{N}]$ is a total mapping s from $\mathbb{N}$ to X with $s(i) = x_i$. The sequence is *ultimately stationary* iff $\exists k \in \mathbb{N} \ \forall n \in \mathbb{N} \ (n \geq k \Rightarrow x_n = x_k)$, and in this case we refer to x_k as the *final* element.

A. Lattices and Fixed Points

We now recall some central concepts needed from order and fixed point theory, including partial orders and lattices. We focus on binary relations on a set X, that is, on subsets of $X \times X$. A binary relation which is reflexive, transitive, and anti-symmetric is a *partial order*. That is, $\sqsubseteq$ is a partial order iff, for all $x, y, z \in X$, (1) $x \sqsubseteq x$, (2) whenever $x \sqsubseteq y$ and $y \sqsubseteq z$ then also $x \sqsubseteq z$, and (3) whenever both $x \sqsubseteq y$ and $y \sqsubseteq x$ hold then $x = y$. We say that $(X, \sqsubseteq)$, the set together with the partial order, is a *partially ordered set*. We write $x \sqsubset y$ to mean $x \sqsubseteq y \wedge x \neq y$.

A common way of depicting a partially ordered set is by letting its elements be the nodes of a directed acyclic graph, in which an edge from x to y indicates $x \sqsubseteq y$. One can simplify the graph by leaving out all edges that follow by transitivity, so that it is the "path" relation in the graph that corresponds to $\sqsubseteq$. This is done in the most common depiction, the *Hasse diagram* for $(X, \sqsubseteq)$, in which, additionally, it is understood that the direction of an edge is always "upwards". Fig. 7 shows three simple Hasse diagrams. The leftmost diagram expresses that $x \sqsubseteq y$ iff $x = 0 \vee y = 1$. The one in the middle says $x \sqsubseteq y$ iff $x = \perp \vee x = y$. It exemplifies the common situation

that X has *incomparable* elements—in this case neither $0 \sqsubseteq 1$ nor $1 \sqsubseteq 0$ holds. A sequence $[x_i \in X \mid i \in \mathbb{N}]$ is a *chain* iff $\forall j, k \in \mathbb{N}\ (x_j \sqsubseteq x_k \vee x_k \sqsubseteq x_j)$, that is, all elements are comparable.

Let Y be a (possibly empty) subset of the partially ordered set X. An element $x \in X$ is a *lower bound* for Y iff $x \sqsubseteq y$ for all $y \in Y$. It is an *upper bound* for Y iff $y \sqsubseteq x$ for all $y \in Y$. A lower bound x for Y is the *greatest lower bound* for Y iff, for every lower bound x' for Y, $x' \sqsubseteq x$. A greatest lower bound may not always exist, but when it does, it is unique, and we denote it by $\bigsqcap Y$. For a two-element set $Y = \{x, y\}$ we write the greatest lower bound simply as $x \sqcap y$ and refer to $\sqcap$ as the 'meet' of x and y. Dually, an upper bound x for Y is the *least upper bound* for Y iff, for every upper bound x' for Y, $x \sqsubseteq x'$. When it exists, it is unique, and we denote it by $\bigsqcup Y$. Again, for $Y = \{x, y\}$ we use $x \sqcup y$, referring to $\sqcup$ as the 'join' of x and y.

A (complete) lattice is a partially ordered set for which every subset has a greatest lower bound and a least upper bound. The middle diagram in Fig. 7 fails to be a lattice for the simple reason that $\{0, 1\}$ has no upper bound, let alone a least upper bound.

If $(X, \sqsubseteq)$ is a lattice then k-tuples of X-elements naturally form a lattice as well, for example if we order the elements of X^k component-wise. That is, we define the ordering thus: $(x_1, \ldots, x_k) \sqsubseteq (y_1, \ldots, y_n)$ iff $x_1 \sqsubseteq y_1 \wedge \cdots \wedge x_k \sqsubseteq y_k$. Similarly, if X is a lattice, the set $D \to X$ of functions from some set D (lattice or not) to X naturally forms a lattice under pointwise ordering. That is, for $F, G \in D \to X$ we define $F \sqsubseteq G$ iff $F(d) \sqsubseteq G(d)$ for all $d \in D$.

Let $F : X \to X$ be defined on the partially ordered set $(X, \sqsubseteq)$. F is *isotone*[1] iff F preserves order, that is, iff $(x \sqsubseteq y) \Rightarrow (F(x) \sqsubseteq F(y))$ for all $x, y \in X$. F is *antitone*[1] iff F reverts order, that is, iff $(x \sqsubseteq y) \Rightarrow (F(y) \sqsubseteq F(x))$ for all $x, y \in X$. An element $x \in X$ is a *fixed point* for F iff $F(x) = x$.

Let X be a complete lattice and let $F : X \to X$ be isotone. Let F's set of fixed points be $Y = \{x \mid F(x) = x\}$. Then $Y \neq \varnothing$ and Y itself forms a complete lattice. $\bigsqcap Y$ is the least fixed point of F and $\bigsqcup Y$ is the greatest fixed point of F. This follows from the Knaster-Tarski fixed point theorem [24]. For many applications of fixed point theory, F's two extreme fixed points are of special interest. We denote the least fixed point by $lfp(F)$ and the greatest fixed point by $gfp(F)$.

Kleene iteration is a procedure which computes the sequence $iter_F(x) = [F^j(x) \mid j \in \mathbb{N}]$ but stops as soon as a final element (a fixed point of F) is reached; if $iter_F(x)$ is not ultimately stationary, the process does not terminate. It is well-known that for an isotone F defined on a *finite* lattice[2] with least element $\bot$ and greatest element $\top$, each of the sequences $iter_F(\bot)$ and $iter_F(\top)$ is an ultimately stationary chain, and that the final element of the former is $lfp(F)$, whereas the final element of the latter is $gfp(F)$. We refer to the former as F's *lower Kleene sequence* and the latter as F's *upper Kleene sequence*, whether or not F is isotone and the lattice is finite. Finally, we say that F is *homing* iff for all $x \in X$, $iter_F(x)$ is ultimately stationary.

[1]We follow Birkhoff [23] in using this term. We note that 'monotone' is often used in place of 'isotone'.

[2]The condition can be weakened in various ways, for example, we can allow an infinite lattice as long as its longest chain is finite.

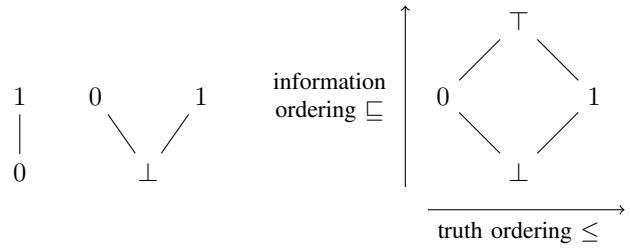


Fig. 7. Partially ordered sets $\mathbf{2}$, $\mathbf{3}$, and $\mathbf{4}$ of truth values

B. Truth Values

Consider again Fig. 7 which shows three commonly used sets of truth values. The set $\mathbf{2} = \{0, 1\}$, ordered classically, gives rise to an ordering of Boolean functions, which is simply classical entailment, $\models$. The ordering given to set $\mathbf{3} = \{\bot, 0, 1\}$ is the usual "information ordering" underlying K_3, Kleene's strong three-valued logic [25]. In this ordering, the classical truth values 0 and 1 are equally informative, but incomparable. The element $\bot$ represents "unknown", or lack of information. K_3 comes about by defining each three-valued connective as the strongest isotone extension of its two-valued cousin. This logic is well known and has found wide application in computer science. One reason is that, in computer science, *partial* functions are common, and Kleene's K_3 lends itself naturally to reasoning about partial functions. K_3 also underlies the technique of ternary analysis which is a focus of this paper.

The set $\mathbf{4} = \{\bot, 0, 1, \top\}$ has two natural orderings which together make $\mathbf{4}$ the simplest non-trivial example of a so-called interlaced bi-lattice [10]. The information ordering, $\sqsubseteq$, will be particularly important for our later discussion of fixed points[3]. Different interpretations can be attached to the elements of $\mathbf{4}$. In our case, the intuition behind the non-classical values is that $\bot$ denotes "neither false nor true" whereas $\top$ denotes "either false or true", as in Belnap's logic [26]. While $\bot$ represents a truth value *gap*, $\top$ represents a *glut*.

It is useful to introduce two alternative but equivalent ways of viewing a function which has $\mathbf{4}$ as its co-domain. Given a four-valued function $\varphi : \mathbf{2}^k \to \mathbf{4}$, we define its *positive content* $pos(\varphi) : \mathbf{2}^k \to \mathbf{2}$ as follows:

$$pos(\varphi)(\vec{v}) = \begin{cases} 1 & \text{if } 1 \sqsubseteq \varphi(\vec{v}) \\ 0 & \text{otherwise} \end{cases}$$

Similarly, the *negative content* is defined:

$$neg(\varphi)(\vec{v}) = \begin{cases} 1 & \text{if } 0 \sqsubseteq \varphi(\vec{v}) \\ 0 & \text{otherwise} \end{cases}$$

[3]Being interlaced entails numerous useful algebraic properties; for example, each meet and each join operation is isotone with respect to *either* ordering and the bilattice is distributive in the strong sense that each meet and each join operation distributes over all the other meet and join operations.

φ is *consistent* iff $\varphi(\vec{v}) \sqsubset \top$ for all $\vec{v}$ and *complete* iff $\bot \sqsubset \varphi(\vec{v})$ for all $\vec{v}$. φ is *two-valued* iff it is complete and consistent.

A pair of two-valued functions $t, f : \mathbf{2}^k \to \mathbf{2}$ can likewise be used to represent a four-valued function. Define

$$whole(t,f)(\vec{v}) = \begin{cases} \bot & \text{if } \neg t(\vec{v}) \wedge \neg f(\vec{v}) \\ 0 & \text{if } \neg t(\vec{v}) \wedge f(\vec{v}) \\ 1 & \text{if } t(\vec{v}) \wedge \neg f(\vec{v}) \\ \top & \text{if } t(\vec{v}) \wedge f(\vec{v}) \end{cases}$$

Then $whole(pos(\varphi), neg(\varphi)) = \varphi$.

Example 3.1: Consider the function φ given by this table:

x	y	φ
0	0	1
0	1	$\top$
1	0	$\bot$
1	1	0

Separating its positive and negative contents, we can write φ as $(\neg x, y)$ without any loss of information. φ is not consistent, as the classical truth assignment $\{x \mapsto 0, y \mapsto 1\}$ satisfies both components. Neither is φ complete, as $\{x \mapsto 1, y \mapsto 0\}$ satisfies neither component. ∎

The positive/negative representation used in this example can help explain the two orderings on $\mathbf{4}$. Let $\models$ be classical (two-valued) logical consequence. Then we have

$$(f,g) \sqsubseteq (f',g') \text{ iff } f \models f' \wedge g \models g'$$

That is, for *information* to increase (or remain unchanged), positive and/or negative support must increase (or remain unchanged). On the other hand,

$$(f,g) \leq (f',g') \text{ iff } f \models f' \wedge g' \models g$$

That is, for *truth* to increase (or remain unchanged), positive support must increase (or remain unchanged) while negative support must *decrease* (or remain unchanged).

We may use a subscript on a connective to indicate which logic it belongs to. For example, $\wedge_{\mathbf{4}}$ and $\vee_{\mathbf{4}}$ are the meet and join operations according to the "truth order" $\leq$ in Fig. 7, and $\neg_{\mathbf{4}}$ acts as identity on $\bot$ and $\top$, but swaps 0 and 1. $\wedge_{\mathbf{3}}$, $\vee_{\mathbf{3}}$, and $\neg_{\mathbf{3}}$ are their restrictions to $\mathbf{3}$. We will often drop the subscript $\mathbf{2}$, so that $\neg$, $\wedge$, $\vee$ are the classical connectives.

The connectives lift to functions by natural extension. Using the positive/negative content encoding explained above, we can rephrase the four-valued operations as follows:

$$\begin{aligned} \neg_{\mathbf{4}}(\varphi,\psi) &= (\psi,\varphi) \\ (\varphi_1,\psi_1) \wedge_{\mathbf{4}} (\varphi_2,\psi_2) &= (\varphi_1 \wedge \varphi_2, \psi_1 \vee \psi_2) \\ (\varphi_1,\psi_1) \vee_{\mathbf{4}} (\varphi_2,\psi_2) &= (\varphi_1 \vee \varphi_2, \psi_1 \wedge \psi_2) \end{aligned} \quad (2)$$

The operations corresponding to other connectives (or types of gate) are easily derived from this.

Given a four-valued function (u,v), its *complement* $\complement(u,v)$ is the function $(\neg u, \neg v)$. The *dual* $\jmath(u,v)$ of the four-valued function (u,v) is (v,u). Both $\complement$ and $\jmath$ are involutions, and they commute. That is, for all four-valued functions φ, $\complement(\complement(\varphi)) = \varphi$, $\jmath(\jmath(\varphi)) = \varphi$, and $\complement \circ \jmath = \jmath \circ \complement$. We define $\bigcirc(\varphi) = \complement(\jmath(\varphi))$ and note that this is also an involution. Two functions φ_1 and φ_2 are *contraduals*[4] iff $\varphi_1 = \bigcirc(\varphi_2)$, and we denote this relation by $\varphi_1 \between \varphi_2$. Note that $\between$ is symmetric, that $\jmath$ is isotone, and that $\complement$ and $\bigcirc$ are antitone. Also note that $\bigcirc$ essentially just swaps $\top$ and $\bot$ in a function but keeps 0 and 1 unchanged — it is an information order analogue of $\neg_{\mathbf{4}}$.

Contraduality will be used in Section IV to define an "implements" relation, and in Section VI for certain fixed point theorems.

[4]We follow Halmos [27] in using this term.

C. Representation of Four-Valued Functions

For clarity we sometimes present functions and their specifications as truth tables. Of course, this is impractical for large specifications. We use the pos/neg representation (discussed above) in many places. However, there are many other ways in which a four-valued function can be encoded using exactly two Boolean functions. For human-readable specifications we may favour a representation for which the care set is more explicit, whereas for circuit minimization a different representation may be preferred, possibly not even relying on Boolean functions at all. For example, we could use Multi-terminal Binary Decision Diagrams [28].

IV. Specifying Circuit Behaviour

We take the view that a k-input, n-output circuit implements n functions, each of type $\mathbf{2}^k \to \mathbf{2}$. As discussed, the possibility of feedback in the circuit means these functions are, in general, partial: When circuits are cyclic or Boolean functions are defined recursively, some input combinations may result in output that is not well-defined; for functions that capture circuit behaviour, we use $\bot$ to denote undefinedness.

An implementor may be prepared to *tolerate* that some output is not well-defined for certain input, but the undefinedness of output is not usually an explicit implementation objective. The intention behind a specification is to circumscribe functions that *may*, rather than *must*, be partial.

It is natural to think of a specification σ (of a single function $h : \mathbf{2}^k \to \mathbf{2}$) as an input-output relation, that is, as a subset of $\mathbf{2}^k \times \mathbf{2}$ (or, equivalently, as a total function $\sigma : \mathbf{2}^k \to \mathbf{4}$). We say that σ is *input-complete* iff for all $\vec{v} \in \mathbf{2}^k$, $(\vec{v},0) \in \sigma$, or $(\vec{v},1) \in \sigma$, or both. We say that σ is *deterministic* iff for all $\vec{v} \in \mathbf{2}^k$, $(\vec{v},0) \notin \sigma$, or $(\vec{v},1) \notin \sigma$, or both. Clearly an input-complete deterministic specification is (isomorphic to) a total function $\mathbf{2}^k \to \mathbf{2}$. Lack of determinism in a specification corresponds to "under-specification" of circuit behaviour and provides the implementor with degrees of freedom.

Some frameworks allow the user to specify that certain input combinations are guaranteed (or required) not to occur. Such requirements correspond to "controllability don't cares" in the terminology of Damiani and De Michelis [29], and "illegal inputs" in the terminology of Tripakis [30]. These are a precondition on the *use* of the function; this is in contrast to the classic notion of don't cares ("observability don't cares" in [29]), where any output is permitted for a given input combination. In some contexts, it is essential to be able to reason about both in one framework (such as interface theories and applications [30], [31]); however, in terms of implementation, the corresponding outputs are unconstrained for both classes of don't cares. We treat both of these cases

as underspecification, rather than input-incompleteness. In our interpretation, an input-incomplete specification would *require* the circuit to take no value for a given input—this is clearly impossible to implement in practice (for a well-structured circuit). However, as we shall see later, it may be useful to permit input-incomplete specifications when comparing partial specifications.

Four-valued logic allows us to talk about (and distinguish) undefinedness and underspecification in one and the same framework. We use $\top$ in specifications to indicate that any behaviour is allowed for a given input vector—whether this is because we really do not care, or it is because we know that the detail does not matter, as the particular input combination should never occur. Underspecification has to do with the attitudes and intentions of a circuit designer. Undefined has to do with the actual behaviour of a circuit. Underspecified and undefined are dual concepts that should be clearly distinguished, but ideally also fit into a single logical framework which allows analysis of the relationship between specification and implementation.

Suppose a circuit is modelled by a (possibly recursively defined) Boolean function h. As there is no limitation on the form of recursive definition, we have to consider h at least three-valued. By considering it four-valued (even though it never uses the value $\top$) allows simpler comparison with specifications. Also suppose a specification has been given in the form of a four-valued function σ, where $\sigma_t = pos(\sigma)$, the cases where the function is allowed to be true and $\sigma_f = neg(\sigma)$, cases where the function is allowed to be false. We define when a function h *implements* a specification σ. Intuitively, the specification puts an upper and lower bound on what can be true and what can be false in the function. For all inputs and outputs, the following must hold:

(i-1) If σ says an output must not be false ($\neg\sigma_f$), that output in h must be true.
(i-2) If an output in h is true, σ must say that output can be true (σ_t).
(i-3) If σ says an output must not be true ($\neg\sigma_t$), that output in h must be false.
(i-4) If an output in h is false, σ must say that output can be false (σ_f).

This is summed up concisely using contraduals. We say that h *implements* σ iff $\bigcirc(h) \sqsubseteq \sigma$. (Owing to the antitony of $\bigcirc$, an equivalent formulation is $\bigcirc(\sigma) \sqsubseteq h$). That is, the implementation must sit above the contradual of the specification, according to the information ordering. Wherever the specification has $\top$ for some input combination, the implementation may deliver any value whereas a 0 or 1 in the specification must result in the same value in the implementation.

Consider the following examples. We can use a table to give a specification for a two-place function σ over inputs x and y, and derive the corresponding positive and negative content:

x	y	σ	σ_t	σ_f
0	0	1	1	0
0	1	$\top$	1	1
1	0	$\top$	1	1
1	1	0	0	1

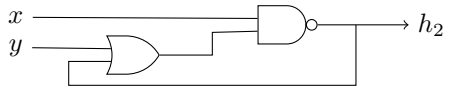


Fig. 8. A cyclic circuit which oscillates when $x = 1$ and $y = 0$

Alternatively, we could use a different representation such as defining the positive and negative content symbolically, leaving σ implicit:

$$\sigma_t = \neg(x \wedge y), \qquad \sigma_f = x \vee y$$

It is easy to check that the function h_1 defined by

$$h_1 = \neg y$$

implements σ, that is, $\neg(x \vee y) \models \neg y \models \neg(x \wedge y)$, and also $\neg\neg(x \wedge y) \models \neg\neg y \models x \vee y$.

Next consider the recursive definition

$$h_2 = \neg(x \wedge (y \vee h_2))$$

which corresponds to the cyclic circuit in Fig. 8. When $x = 1$ and $y = 0$ there is no Boolean solution to the equation but there is a solution using three-valued logic, $h_2 = \bot$, which indicates the circuit is not well behaved for these inputs; it oscillates. This three-valued solution naturally extends to four-valued logic. Because σ allows any behaviour when $x \neq y$, we can say h_2 implements σ, in spite of being a partial function over Booleans. In contrast, if we consider the (also partial) function h_3 defined by

$$h_3 = \neg x \wedge \neg h_3$$

we note that it never yields 1, and so it fails to satisfy item (i-1) above. Similarly, the function h_4, defined by

$$h_4 = \neg x \vee (y \wedge \neg h_4)$$

is undefined for $x = 1, y = 1$, and so fails to satisfy item (i-3) above. Hence neither h_3 nor h_4 implements σ. The following table compares the specification, its contradual and the functions h_1–h_4.

x	y	σ	$\bigcirc(\sigma)$	h_1	h_2	h_3	h_4
0	0	1	1	1	1	$\bot$	1
0	1	$\top$	$\bot$	0	1	$\bot$	1
1	0	$\top$	$\bot$	1	$\bot$	0	0
1	1	0	0	0	0	0	$\bot$

Note the dual roles of $\bot$ and $\top$, as well as the tolerance of undefined behaviour in case of h_2. The functions h_3 and h_4 are rejected, not because they are partial, but because they are incorrect as implementations of σ. Also note that the "implements" relation is not simply the information ordering $\sqsubseteq$ between the implementation and specification: Applying $\sqsubseteq$ pointwise, we have $h_3 \sqsubseteq \sigma$, yet h_3 does not implement the specification σ.

We now turn to a more realistic example, namely the specification of a seven-segment display decoder. With the seven segments labeled as shown in Fig. 9, the specification of the seven functions a–g is given by Table I. Each of a–g is

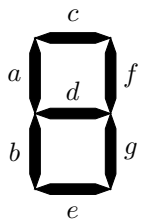


Fig. 9. Display segments

$$
\begin{aligned}
a &= (\neg x_0 \wedge \neg x_3 \wedge \neg c) \vee (\neg x_1 \wedge c)\\
b &= \neg x_0 \wedge ((\neg x_1 \wedge \neg x_2) \vee (x_1 \wedge \neg x_3))\\
c &= (x_0 \wedge x_2 \wedge \neg x_3) \vee (\neg x_2 \wedge ((\neg x_1 \wedge x_3) \vee e))\\
d &= (x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (a \wedge (x_2 \vee x_3))\\
e &= (x_0 \wedge \neg x_3 \wedge d) \vee b\\
f &= (\neg x_2 \wedge c) \vee (\neg x_3 \wedge \neg e)\\
g &= (x_0 \wedge \neg x_3) \vee a
\end{aligned}
$$

Fig. 10. Cyclic circuit for a 7-segment display, after Backes and Riedel [22]

digit	x_3	x_2	x_1	x_0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	0	1	1	1
1	0	0	0	1	0	0	0	0	0	1	1
2	0	0	1	0	0	1	1	1	1	1	0
3	0	0	1	1	0	0	1	1	1	1	1
4	0	1	0	0	1	0	0	1	0	1	1
5	0	1	0	1	1	0	1	1	1	0	1
6	0	1	1	0	1	1	0	1	1	0	1
7	0	1	1	1	0	0	1	0	0	1	1
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	0	1	1	0	1	1
	1	0	1	0	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
	1	0	1	1	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
	1	1	0	0	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
	1	1	0	1	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
	1	1	1	0	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
	1	1	1	1	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$

TABLE I
SPECIFICATION OF THE SEVEN-SEGMENT DECODER

digit	x_3	x_2	x_1	x_0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	0	1	1	1
1	0	0	0	1	0	0	0	0	0	1	1
2	0	0	1	0	0	1	1	1	1	1	0
3	0	0	1	1	0	0	1	1	1	1	1
4	0	1	0	0	1	0	0	1	0	1	1
5	0	1	0	1	1	0	1	1	1	0	1
6	0	1	1	0	1	1	0	1	1	0	1
7	0	1	1	1	0	0	1	0	0	1	1
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	0	1	1	0	1	1
	1	0	1	0	0	0	0	0	0	0	0
	1	0	1	1	0	0	0	0	0	0	0
	1	1	0	0	0	0	0	0	0	0	0
	1	1	0	1	0	0	0	0	0	0	0
	1	1	1	0	0	0	0	0	0	0	0
	1	1	1	1	0	0	0	0	0	0	0

TABLE II
SOLUTION TO FIG. 10'S EQUATIONS

a function of four propositional variables (bits), but only ten input combinations (corresponding to the ten decimal digits) are of relevance. The remaining six combinations fall outside the care set. This is reflected in the table: these six input combinations yield $\top$ for all outputs. They are models for both the positive and the negative content of the specification. If these combinations are guaranteed never to occur, for example, an implementer can utilise the slack in the specification, as we shall see.

Fig. 10 gives recursive definitions for functions a–g. These equations satisfy Table I's specification. This set of definitions was synthesised by Riedel and colleagues [20], [32], [22]. Riedel [20] adapted the Berkeley SIS circuit generator, utilising a strategy for the application of substitution of functions that was more liberal than that of the SIS, allowing for the construction of cyclic circuits. A more recent approach [22] uses an interpolant-generating SAT solver for this kind of synthesis. The original method [20] proceeds in two steps: Synthesis discovers cyclic "candidate" circuits, and subsequent analysis tests each for "well-behavedness", rejecting the ill-behaved candidates. The analysis part uses three-valued logic and is well understood; in Section V we recast it for the four-valued setting.

The equations of Fig. 10 have a two-valued solution shown in Table II. Note that zeros are produced wherever the specification has $\top$. This is a natural set of functions to use when the method of specification does not support a notion of "don't care". A less precise specification, however, allows more flexibility in what circuits we generate. For example, we can see from Table I that in all the cases we care about, b is a function of x_0–x_2, so x_3 is not needed in the definition of b. We can replace Fig. 10's definition by the shorter

$$b = \neg x_0 \wedge (x_1 \vee \neg x_2)$$

saving two and-gates and two inverters. Similarly, we can safely replace the definition of g by the shorter

$$g = x_0 \vee a$$

saving one and-gate and one inverter. Such simplifications do not preserve all the behaviours of the circuit, but they do preserve all the behaviours that we care about, as expressed in the specification. Other such simplifications are also possible. Constraining all outputs to be zero (for example) in cases we don't care about unnecessarily limits our choice of circuits. A more flexible (many-valued) specification may facilitate a better solution.

So far, we have only discussed specifications which contain the three values 0, 1 and $\top$, with $\bot$ only appearing in contraduals of specifications and in implementations. However, the availability of all four values also gives us more flexibility when comparing and manipulating specifications. For comparing the strength of specifications, the information order can be used. Given two specifications σ_1 and σ_2, we can say σ_1 is stronger than (is a *refinement* of) σ_2 iff $\sigma_1 \sqsubseteq \sigma_2$. With four values, we can say that $\sigma_1 \sqcap \sigma_2$ is the weakest specification

which is a refinement of both σ_1 and σ_2 (without $\bot$, $\sqcap$ is not always defined). A circuit implements both σ_1 and σ_2 iff it implements $\sigma_1 \sqcap \sigma_2$. This greatest lower bound contains $\bot$ exactly when there is no circuit which implements both σ_1 and σ_2, so $\bot$ appearing in a specification essentially means "impossible to implement". The contradual of the specification contains $\top$, which cannot be below or equal to the implementation since the implementation can only contain 0, 1 and $\bot$.

Consider the specifications given below:

x	y	σ_1	σ_2	σ_3	$\sigma_1 \sqcap \sigma_2$	$\sigma_1 \sqcap \sigma_3$
0	0	1	1	1	1	1
0	1	$\top$	$\top$	0	$\top$	0
1	0	$\top$	0	$\top$	0	$\top$
1	1	0	$\top$	1	0	$\bot$

The specifications σ_1 and σ_2 are incomparable in the information ordering, as $\sigma_1(1,1) \sqsubset \sigma_2(1,1)$ but $\sigma_2(1,0) \sqsubset \sigma_1(1,0)$, so neither is a refinement of the other. However, $\sigma_1 \sqcap \sigma_2$ is a refinement of both σ_1 and σ_2, and there are many functions which implement it. Next, consider combining σ_1 with σ_3. Observe that $\sigma_1(1,1) = 0$, but $\sigma_3(1,1) = 1$. When we construct a specification refining both σ_1 and σ_3, we obtain a specification containing $\bot$, indicating that the resulting specification is impossible to implement — σ_1 and σ_3 are incompatible.

V. Ternary Analysis Revisited

We have argued that four-valued logic is an appropriate setting for reasoning about both intended and actual behaviour, and how they relate. The additional fourth value does not complicate the definition of "ternary" analysis; if anything, it introduces a pleasing symmetry. For this reason we shall consider the ternary method as one that operates in $\mathbf{4}$— although traditionally, it makes no use of the fourth value.

The "ternary test" used by Riedel [20] has a long history. In its "symbolic" form (the one that we shall use) it was introduced by Bryant [33] and used by Malik [34], Riedel and others. The use of multiple-valued logic more generally for reasoning about circuits goes back to the 1950s.

The ternary test is a fixed point finding procedure. For each wire in a given circuit, a pair (t, f) of expressions are maintained. (As we saw in Section III-B, this is a useful way of representing a four-valued function.) The first component expresses conditions under which the wire will have value 1. The second expresses conditions under which the wire will have value 0. For an input wire x, the pair is $(x, \neg x)$, and this remains unchanged during analysis. For any other wire, initially the pair is $(0, 0)$, but each component may grow as we propagate information from input and across the circuit, including feedback arcs.

It is convenient to introduce a notion of a circuit *decoration*. If a circuit c has set W of wires then a decoration of c maps each $w \in W$ to a function which is intended to describe the conditions under which w is 0 or 1. More specifically, if c has k input wires, we define the set of four-valued decorations $\Delta_{\mathbf{4}} = W \to \mathbf{2}^k \to \mathbf{4}$ and the set of two-valued decorations $\Delta_{\mathbf{2}} = W \to \mathbf{2}^k \to \mathbf{2}$. So a four-valued decoration associates a four-valued function (which may of course be three- or two-valued) with each wire. The ordering on $\Delta_{\mathbf{4}}$ is the pointwise extension of the information ordering $\sqsubseteq$ from $\mathbf{4}$, and similarly the ordering on $\Delta_{\mathbf{2}}$ is that of $\mathbf{2}$, used point-wise. This makes $\Delta_{\mathbf{4}}$ and $\Delta_{\mathbf{2}}$ (complete) lattices.

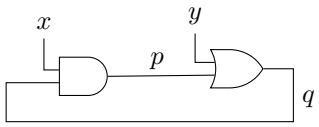


Fig. 11. A small cyclic circuit

For the formal definition of ternary analysis, we need to settle on some circuit representation. We represent a circuit c as a tuple $(W, op, in, in1, in2)$ where W is the set of (input, internal, and output) wires, $op : W \to \{\mathbf{nil}, \mathbf{neg}, \mathbf{and}, \mathbf{or}, \mathbf{xor}, \mathbf{nand}, \mathbf{nor}\}$ is a function which, given a wire w, gives the type of gate that w emanates from, with $op(w) = \mathbf{nil}$ iff w is an input wire. Similarly, in, $in1$, and $in2$ are (partial) functions that, given w, give w's immediate sources. That is, if w emanates from an inverter ($op(w) = \mathbf{neg}$), then $in(w)$ is the inverter's input wire. If w emanates from any other type of gate, $in1(w)$ and $in2(w)$ are the two input wires (the operations considered are all commutative, so the order of inputs is of no concern). As an example, the circuit shown in Fig. 11 is represented as $(\{x, y, p, q\}, op, \{\ \}, \{p \mapsto x, q \mapsto y\}, \{p \mapsto q, q \mapsto p\})$ where $op = \{x \mapsto \mathbf{nil}, y \mapsto \mathbf{nil}, p \mapsto \mathbf{and}, q \mapsto \mathbf{or}\}$.

The following definition captures not only ternary analysis, but also two-valued and four-valued reasoning. It is parameterised over the set of truth values used ($\mathbf{d} \in \{\mathbf{2}, \mathbf{3}, \mathbf{4}\}$). $\Phi_{\mathbf{d}}$ has type $\Delta_{\mathbf{d}} \to \Delta_{\mathbf{d}}$, and is relative to a given circuit c.

$$\Phi_{\mathbf{d}}(\delta)(w) = \begin{cases} init_{\mathbf{d}}(w) & \text{if } op(w) = \mathbf{nil} \\ \neg_{\mathbf{d}}\ \delta(in(w)) & \text{if } op(w) = \mathbf{neg} \\ \delta(in1(w))\ op(w)_{\mathbf{d}}\ \delta(in2(w)) & \text{otherwise} \end{cases}$$

with $init_{\mathbf{2}}(w) = w$, and $init_{\mathbf{3}}(w) = init_{\mathbf{4}}(w) = (w, \neg w)$.

The operations $op(w)_{\mathbf{4}}$ were defined in Section III.[5] It is important to note that they all are isotone with respect to $\sqsubseteq$, and that $\Phi_{\mathbf{4}}$ therefore has fixed points.

$\Phi_{\mathbf{4}}$ pins down exactly how ternary analysis propagates information, based on the semantics of the gates involved.[6] The point when propagation stabilizes corresponds to reaching $\Phi_{\mathbf{4}}$'s least fixed point. At that point, the annotation (u, v) of the individual wire w is checked. If (u, v) satisfies $u \equiv \neg v$ then the wire has passed the test, because it means that there is no truth value "gap": no input combination remained mapped to $\bot$.

[5] Equations 2 dealt with negation, conjunction and disjunction; adding other connectives is trivial.

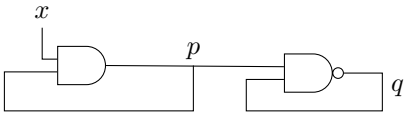
[6] The description by Riedel [20] (Section 4.3.1) differs from our definition of $\Phi_{\mathbf{3}}$, in that Riedel uses (in our notation) $\Phi_{\mathbf{3}}(\delta)(w) = \delta(w) \sqcup e$, where we use $\Phi_{\mathbf{3}}(\delta)(w) = e$. Riedel's formulation forces $\Phi_{\mathbf{3}}$ to be an increasing function. If we were interested only in $\Phi_{\mathbf{3}}$'s least fixed point (as Riedel is), this forcing would be harmless, albeit unnecessary. But given our interest in other fixed points, in particular in the four-valued case, Riedel's definition has to be replaced by the simpler one given here.

Example 5.1: The circuit in Fig. 11 has a "semantic cycle" for $x = 1, y = 0$. In that case, $p = q$, that is, we have two different stable states. The circuit fails the ternary test, since we get the three- or four-valued fixed point $\{p \mapsto (x \wedge y, \neg x), q \mapsto (y, \neg x \wedge \neg y)\}$. There are two additional three-valued fixed points. If we perform Kleene iteration using two-valued logic, we reach a fixed point, namely $\{p \mapsto x \wedge y, q \mapsto y\}$. Another two-valued fixed point is $\{p \mapsto x, q \mapsto x \vee y\}$. ∎

It turns out to be no coincidence that the second fixed point in Example 5.1 consists of the negations of the negative content of the three-valued fixed point. We return to this phenomenon in Section VI.

Example 5.2: Consider the circuit in Fig. 8, again with a semantic cycle for $x = 1, y = 0$. In this case, the equation becomes $h_2 = \neg h_2$, there is no two-valued solution and the circuit fails the ternary test. Here we get a single three- or four-valued fixed point: $\{h_2 \mapsto (\neg x, x \wedge y)\}$. ∎

Example 5.3: Here is a third example of a circuit which fails the ternary test:



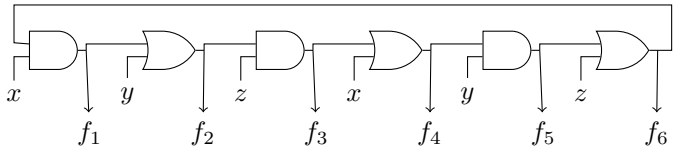
Or, as a set of recursive equations:

$$\begin{aligned} p &= x \wedge p \\ q &= \neg(q \wedge p) \end{aligned}$$

In this case the equations have a unique two-valued solution. If $p = 1$ we have $q = \neg q$ and there is no solution. Hence $p = 0$ and so $q = 1$. To demonstrate that the circuit fails the ternary test, here is the three-valued lower Kleene sequence:

$$\begin{array}{lccc} p: & (0,0) & (0,\neg x) & (0,\neg x) \\ q: & (0,0) & (0,0) & (\neg x,0) \end{array}$$ ∎

Example 5.4: An nice example of a circuit which passes the ternary test (from Rivest [35]) uses just six gates to compute six different functions $f_1 \ldots f_6$. These are functions of three variables x, y, and z (we have drawn six different sources, repeating each of the three variables, so as to avoid cluttering the diagram).



Despite the cycle, Rivest's circuit is safe—each output is uniquely determined by the input. For any input, the circuit stabilizes, implementing these functions:

$$\begin{array}{lll} f_1: & x \wedge (y \vee z) & \qquad f_4: \quad x \vee (y \wedge z) \\ f_2: & y \vee (x \wedge z) & \qquad f_5: \quad y \wedge (x \vee z) \\ f_3: & z \wedge (x \vee y) & \qquad f_6: \quad z \vee (x \wedge y) \end{array}$$ ∎

The circuit given in Fig. 3 fails the test, as expected. The 7-segment circuit from Fig. 10 passes the test, and the result of the analysis is exactly as given in Table II.

Example 5.5: As a final example, Fig. 12 shows a different solution to the 7-segment display problem. It fails the ternary test, which yields the result shown in Table III. It also fails to be an implementation of Table I's specification, according to our definition of the "implements" relation. For example, take the function $g : \mathbf{2}^4 \to \mathbf{3}$ from Table III and let $\sigma : \mathbf{2}^4 \to \mathbf{4}$ be g's specification from Table I. We have $\neg pos(\sigma) = \neg x_0 \wedge x_1 \wedge \neg x_2 \wedge \neg x_3$. There is only one classical model of this, namely 0100, and this truth assignment does not make g true (g is made $\perp$). Hence it violates condition (i-3). ∎

$$\begin{aligned} a &= (\neg x_3 \wedge \neg x_0 \wedge \neg c) \vee (\neg x_1 \wedge c) \\ b &= \neg x_0 \wedge e \\ c &= (\neg x_3 \wedge x_2 \wedge x_0) \vee (\neg x_2 \wedge ((x_3 \wedge \neg x_1) \vee e)) \\ d &= ((x_3 \vee x_2) \wedge a) \vee (x_1 \wedge e) \\ e &= (\neg x_2 \wedge (x_1 \vee \neg x_0) \wedge f) \vee (\neg x_3 \wedge \neg f) \\ f &= ((\neg x_2 \vee (\neg x_1 \wedge \neg x_0)) \wedge g) \vee (\neg x_3 \wedge \neg a) \\ g &= (\neg x_3 \wedge \neg b) \vee a \end{aligned}$$

Fig. 12. Another 7-segment display circuit, from Riedel [20] page 15

digit	x_3	x_2	x_1	x_0	a	b	c	d	e	f	g
0	0	0	0	0	$\perp$	$\perp$	$\perp$	0	$\perp$	$\perp$	$\perp$
1	0	0	0	1	0	0	0	0	0	1	1
2	0	0	1	0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
3	0	0	1	1	0	0	1	1	1	1	1
4	0	1	0	0	1	0	0	1	0	1	1
5	0	1	0	1	1	0	1	1	1	0	1
6	0	1	1	0	1	1	0	1	1	0	1
7	0	1	1	1	0	0	1	0	0	1	1
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	0	1	1	0	1	1
	1	0	1	0	0	0	0	0	0	0	0
	1	0	1	1	0	0	0	0	0	0	0
	1	1	0	0	0	0	0	0	0	0	0
	1	1	0	1	0	0	0	0	0	0	0
	1	1	1	0	0	0	0	0	0	0	0
	1	1	1	1	0	0	0	0	0	0	0

TABLE III
ANALYSIS OF FIG. 12'S EQUATIONS

There are circuits which fail the ternary test yet the function which describes their behaviour is an implementation of their specification—our h_2 of Section IV is an example. The ternary test rejects a circuit if, for any input combination, the corresponding function results in $\perp$ for any output. Here we are suggesting this constraint is unnecessarily strong—ill-behaviour is acceptable for input combinations which are outside the care set. Our definition of "implements" captures this intention.

Of course it is possible to cast this in terms of Boolean functions also. With circuit behaviour represented as the positive and negative Boolean functions (u, v) for each wire, the ternary test fails if $\neg u \wedge \neg v$ is satisfiable. If t is a formula which represents the care set (which may be explicit in the representation of the specification, or may be computed in some way), we could instead check whether $\neg t \wedge \neg u \wedge \neg v$ is satisfiable. In cases where the specification contains only 0, 1 and $\top$, this formulation is equivalent to our definition of "implements".

VI. Fixed Point Results

We now give a number of results that relate 2-, 3- and 4-valued fixed points. The intention is to explain certain phenomena in ternary analysis. Based on an observation that 2-valued Kleene iteration also terminates for the equations given in Fig. 10, we identify a class of non-isotone functions that have fixed points. Then, in a second part, we explain why 4-valued fixed points tend to come in pairs.

A. Non-Isotone Functions that Have Fixed Points

Consider again the definitions given in Fig. 10. From a classical standpoint the collection of functions a–g are not obviously well-defined. The functions a, c, d, and e are mutually recursive, and it is not at all clear that the (higher-order, classical) function $\Phi : (\mathbf{2}^4 \to \mathbf{2})^4 \to (\mathbf{2}^4 \to \mathbf{2})^4$ defined by

$$\Phi \begin{pmatrix} a \\ c \\ d \\ e \end{pmatrix} = \begin{pmatrix} (\neg x_0 \wedge \neg x_3 \wedge \neg c) \vee (\neg x_1 \wedge c) \\ (x_0 \wedge x_2 \wedge \neg x_3) \vee (\neg x_2 \wedge ((\neg x_1 \wedge x_3) \vee e)) \\ (x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (a \wedge (x_2 \vee x_3)) \\ (x_0 \wedge \neg x_3 \wedge d) \vee b \end{pmatrix}$$

should have a fixed point, let alone a unique fixed point, as non-isotone operations are involved in the definition. In fact there is a call chain from c to e to d to a, back to c via negation.

Indeed, if we order the set $(\mathbf{2}^4 \to \mathbf{2})^4$ classically, that is, use entailment in two-valued propositional logic componentwise, Kleene iteration on Φ does not produce a chain. Rather, it produces a sequence $\vec{\varphi_0}$, $\vec{\varphi_1}$, $\vec{\varphi_2}$, $\vec{\varphi_3}$, $\vec{\varphi_4}$, $\vec{\varphi_5}$, $\vec{\varphi_5}$, …, in which $\vec{\varphi_2}$ and $\vec{\varphi_3}$ are incomparable, as are $\vec{\varphi_4}$ and $\vec{\varphi_5}$. More specifically, we get the sequence shown in Table IV, where $\vec{\varphi_5}$ is a fixed point. Table IV uses sets-of-models notation for functions (for example, 0001 represents the truth assignment $\{x_0 \mapsto 0, x_1 \mapsto 0, x_2 \mapsto 0, x_3 \mapsto 1\}$). Note that, regarding f, $\vec{\varphi_2}$ suggests models 0110 and 1010, which ultimately turn out to be counter-models. The same goes for g and 0100: a model is being added, only to be retracted later. In other words, the lower Kleene sequence is not a chain, and indeed, we could not expect it to be.

In general, in the absence of isotony, Kleene sequences will not be chains and we have no guarantee that such sequences will be ultimately stationary, so Kleene iteration may not terminate. Nevertheless, for the equations of Fig. 10, Kleene iteration does terminate and Φ has a unique fixed point in two-valued logic, namely $\vec{\varphi_5}$. Proposition 6.2 below shows that this is no coincidence. The fixed point is as shown in Table II.

Fig. 12 gave a different purported solution to the seven-segment decoder problem. This set of equations is not unlike that of Fig. 10: the (two-valued) lower Kleene sequence is ultimately stationary, and the fixed point is the same as the one from Table II. However, as Example 5.5 showed, once we move to three-valued solutions and the ternary test, the two sets of equations turn out to have different properties.

We now consider the relation between four-valued fixed point finding (noting that a four-valued fixed point always exists) and two-valued Kleene iteration.

$\vec{\varphi_0}$:	$a_0 = b_0 = c_0 = d_0 = e_0 = f_0 = g_0 = \{\ \}$	
$\vec{\varphi_1}$:	a_1	$= \{0000, 0010, 0100, 0110\}$
	b_1	$= \{0000, 0001, 0100, 0110\}$
	c_1	$= \{0001, 1001, 1010, 1110\}$
	d_1	$= \{0100, 1100\}$
	e_1	$= \{\ \}$
	f_1	$= \{0000, 0010, 0100, 0110, 1000, 1010, 1100, 1110\}$
	g_1	$= \{1000, 1010, 1100, 1110\}$
$\vec{\varphi_2}$:	a_2	$= \{0000, 0001, 0010, 0100, 0110, 1001, 1010\}$
	b_2	$= \{0000, 0001, 0100, 0110\}$
	c_2	$= \{0001, 1001, 1010, 1110\}$
	d_2	$= \{0010, 0100, 0110, 1100\}$
	e_2	$= \{0000, 0001, 0100, 0110, 1100\}$
	f_2	$= \{0000, 0001, 0010, 0100, 0110, 1000, 1001, 1010, 1100, 1110\}$
	g_2	$= \{0000, 0010, 0100, 0110, 1000, 1010, 1100, 1110\}$
$\vec{\varphi_3}$:	a_3	$= \{0000, 0001, 0010, 0100, 0110, 1001, 1010\}$
	b_3	$= \{0000, 0001, 0100, 0110\}$
	c_3	$= \{0000, 0001, 0100, 1001, 1010, 1100, 1110\}$
	d_3	$= \{0001, 0010, 0100, 0110, 1001, 1010, 1100\}$
	e_3	$= \{0000, 0001, 0100, 0110, 1100\}$
	f_3	$= \{0001, 0010, 1000, 1001, 1010, 1110\}$
	g_3	$= \{0000, 0001, 0010, 0100, 0110, 1000, 1001, 1010, 1100, 1110\}$
$\vec{\varphi_4}$:	a_4	$= \{0000, 0001, 0010, 0110, 1001, 1010\}$
	b_4	$= \{0000, 0001, 0100, 0110\}$
	c_4	$= \{0000, 0001, 0100, 1001, 1010, 1100, 1110\}$
	d_4	$= \{0001, 0010, 0100, 0110, 1001, 1010, 1100\}$
	e_4	$= \{0000, 0001, 0100, 0110, 1010, 1100\}$
	f_4	$= \{0000, 0001, 0010, 0100, 1000, 1001, 1010, 1100, 1110\}$
	g_4	$= \{0000, 0001, 0010, 0100, 0110, 1000, 1001, 1010, 1100, 1110\}$
$\vec{\varphi_5}$:	a_5	$= \{0000, 0001, 0010, 0110, 1001, 1010\}$
	b_5	$= \{0000, 0001, 0100, 0110\}$
	c_5	$= \{0000, 0001, 0100, 1001, 1010, 1100, 1110\}$
	d_5	$= \{0001, 0010, 0100, 0110, 1001, 1010, 1100\}$
	e_5	$= \{0000, 0001, 0100, 0110, 1010, 1100\}$
	f_5	$= \{0000, 0001, 0010, 0100, 1000, 1001, 1100, 1110\}$
	g_5	$= \{0000, 0001, 0010, 0110, 1000, 1001, 1010, 1100, 1110\}$

TABLE IV
TWO-VALUED LOWER KLEENE SEQUENCE

When ternary analysis identifies a truth value gap ($\bot$) on some wire, there may not be a two-valued fixed point, in which case two-valued Kleene iteration cannot terminate. Alternatively, there may be several two-valued fixed points, in which case two-valued Kleene iteration may or may not terminate. However, if ternary analysis finds no gaps then two-valued Kleene iteration must terminate (Proposition 6.2 below) and there is only one fixed point (Proposition 6.6). The next lemma gives information about cases where two-valued Kleene iteration does *not* terminate. Note that, since the lattice $\mathbf{\Delta_2}$ is finite, non-termination means that Kleene iteration must eventually traverse some *cycle* $\delta_0 \ldots \delta_{m-1}, \delta_0$.

Proposition 6.1: Let $D = \{\delta_0, \delta_1, \ldots, \delta_{m-1}\}$, with $m > 1$, be a set of distinct decorations of some circuit, such that $\Phi_{\mathbf{2}}(\delta_i) = \delta_{(i+1) \bmod m}$ for all $0 \leq i < m$. Let $\delta = \bigsqcap D$. Then

1) $\Phi_{\mathbf{4}}(\delta) \sqsubseteq \delta$;
2) $\mathit{lfp}(\Phi_{\mathbf{4}}) \sqsubseteq \delta$; and
3) $\mathit{lfp}(\Phi_{\mathbf{4}})$ is not two-valued.

Proof: (1) Consider an arbitrary $\delta_i \in D$. We have $\delta \sqsubseteq \delta_i$, so by isotony of $\Phi_{\mathbf{4}}$, $\Phi_{\mathbf{4}}(\delta) \sqsubseteq \Phi_{\mathbf{4}}(\delta_i)$. Since δ_i is two-valued, $\Phi_{\mathbf{4}}(\delta_i) = \Phi_{\mathbf{2}}(\delta_i) = \delta_{(i+1) \bmod m}$. As δ_i was arbitrary, it follows that $\Phi_{\mathbf{4}}(\delta)$ is a lower bound for D. But by definition,

δ is the *greatest* lower bound for D, so $\Phi_{\mathbf{4}}(\delta) \sqsubseteq \delta$.

(2) Now consider the sequence $[\Phi_{\mathbf{4}}^j(\delta) \mid j \geq 0]$. This must be a decreasing chain which is ultimately stationary. Namely, owing to the isotony of $\Phi_{\mathbf{4}}$ and the just established fact that $\Phi_{\mathbf{4}}(\delta) \sqsubseteq \delta$, we have $\Phi_{\mathbf{4}}^{j+1}(\delta) \sqsubseteq \Phi_{\mathbf{4}}^j(\delta)$ for all $j \geq 0$. And, by transitivity of $\sqsubseteq$, $\Phi_{\mathbf{4}}^j(\delta) \sqsubseteq \delta$ for all $j \geq 0$. As the lattice $\Delta_{\mathbf{4}}$ is finite, the decreasing $[\Phi_{\mathbf{4}}^j(\delta) \mid j \geq 0]$ is ultimately stationary, that is, for some j, $\Phi_{\mathbf{4}}^j(\delta)$ is a fixed point of $\Phi_{\mathbf{4}}$. It follows that the *least* fixed point $\mathit{lfp}(\Phi_{\mathbf{4}}) \sqsubseteq \Phi_{\mathbf{4}}^j(\delta)$ and hence $\mathit{lfp}(\Phi_{\mathbf{4}}) \sqsubseteq \delta$.

(3) Since $\delta_0, \ldots, \delta_{m-1}$ are different two-valued decorations, they are incomparable in $\mathbf{4}$'s information ordering $\sqsubseteq$. Hence $\delta \sqsubset \delta_i$ for $0 \leq i < m$, and so $\delta(w)$ is consistent, but not complete, for each wire w. Since $\mathit{lfp}(\Phi_{\mathbf{4}}) \sqsubseteq \delta$, the same holds for $\mathit{lfp}(\Phi_{\mathbf{4}})(w)$, and hence $\mathit{lfp}(\Phi_{\mathbf{4}})$ is not two-valued. ■

Example 6.1: Consider the recursive set of equations

$$\begin{array}{lcl} p & = & q \\ q & = & p \\ r & = & p \vee q \vee s \\ s & = & p \vee q \vee \neg r \\ t & = & t \end{array}$$

There is one two-valued loop which has $\langle p, q, r, s, t\rangle$ map to $\langle 0, 1, 1, 1, 1\rangle$ and $\langle 1, 0, 1, 1, 1\rangle$ alternately. The (information order) meet of these is $\delta = \langle \bot, \bot, 1, 1, 1\rangle$, and $\Phi_{\mathbf{4}}(\delta) = \langle \bot, \bot, 1, \bot, 1\rangle$. There are two fixed points below this, namely $\langle \bot, \bot, \bot, \bot, 1\rangle$ and $\mathit{lfp}(\Phi_{\mathbf{4}}) = \langle \bot, \bot, \bot, \bot, \bot\rangle$. ■

The next proposition says that, if a set of equations determine a *combinational* circuit, then two-valued Kleene iteration terminates for any starting point, even if the Kleene sequence is not a chain.

Proposition 6.2: If $\mathit{lfp}(\Phi_{\mathbf{4}})$ is two-valued then $\Phi_{\mathbf{2}}$ is homing, that is, for all x, the sequence $\mathit{iter}_{\Phi_{\mathbf{2}}}(x)$ is ultimately stationary.

Proof: Suppose Kleene iteration loops, that is, the sequence $[\Phi_{\mathbf{2}}^j(x) \mid j \geq 0]$ is not ultimately stationary. Since the lattice $\Delta_{\mathbf{2}}$ is finite, there must be an integer $m > 1$ and a sequence $\delta_0, \delta_1, \ldots, \delta_{m-1}$ such that $\Phi_{\mathbf{2}}(\delta_i) = \delta_{(i+1) \bmod m}$ and the δ_i are all distinct. Thus by Lemma 6.1(3), $\mathit{lfp}(\Phi_{\mathbf{4}})$ is not two-valued. By contraposition, if $\mathit{lfp}(\Phi_{\mathbf{4}})$ is two-valued, $\mathit{iter}_{\Phi_{\mathbf{2}}}(x)$ is ultimately stationary. ■

Proposition 6.2 explains why the Kleene sequence in Table IV stabilized, seemingly against the odds. The proposition moreover tells us that this would happen irrespective of where we started the iteration.

Recall that if the circuit does *not* pass the ternary test, two-valued Kleene iteration may or may not terminate.

B. Contra-Duality and Pairs of Fixed Points

Using the positive/negative representation for four-valued functions, we now discuss how fixed points for $\Phi_{\mathbf{4}}$ come in pairs. This explains why, as happened in Section V, Example 5.1, an incomplete four-valued fixed point gives rise to two different two-valued fixed points. First, a proposition which is little more than an observation.

Proposition 6.3: A four-valued function (u, v) is self-contradual iff (u, v) is two-valued.

Proof: (u, v) is its own contradual iff $u \equiv \neg v$ iff (u, v) is two-valued. ■

We can lift $\mathbb{C}$, $\mathbb{D}$, $\bigcirc$, and $\between$ to decorations in $\Delta_{\mathbf{4}}$ in the obvious (pointwise) way. For example, $\delta_1 \between \delta_2$ iff $\delta_1(w) \between \delta_2(w)$ for all $w \in W$. This way, Proposition 6.3 applies equally well to four-valued decorations.

The next proposition shows that fixed points for $\Phi_{\mathbf{4}}$ come in pairs of contraduals.

Proposition 6.4: If φ is a fixed point of $\Phi_{\mathbf{4}}$ then so is $\bigcirc(\varphi)$.

Proof: The four-valued connectives all preserve the relation $\between$. For negation, this is easily seen: if $(u, u') \between (x, x')$ holds then so does $(u', u) \between (x', x)$. The cases for the other connectives follow similar patterns, so we present just the case for conjunction. Assume $(u, u') \between (x, x')$ and $(v, v') \between (y, y')$. From this it follows that $u \wedge v = \neg x' \wedge \neg y'$ and that $u' \vee v' = \neg x \vee \neg y$. Hence

$$(u \wedge v, u' \vee v') = (\neg(x' \vee y'), \neg(x \wedge y))$$

that is, $(u, u') \wedge_{\mathbf{4}} (v, v') \between (x, x') \wedge_{\mathbf{4}} (y, y')$. By structural induction, and lifting the result to decorations, we have

$$\delta_1 \between \delta_2 \Rightarrow \Phi_{\mathbf{4}}(\delta_1) \between \Phi_{\mathbf{4}}(\delta_2)$$

Hence if δ_1 is a fixed point of $\Phi_{\mathbf{4}}$, then both of δ_2 and $\Phi_{\mathbf{4}}(\delta_2)$ are its contraduals. But a contradual is unique, so $\Phi_{\mathbf{4}}(\delta_2) = \delta_2$, that is δ_1's contradual is also a fixed point. ■

Proposition 6.5: $\mathit{lfp}(\Phi_{\mathbf{4}}) \between \mathit{gfp}(\Phi_{\mathbf{4}})$.

Proof: Let $\delta = \bigcirc(\mathit{lfp}(\Phi_4))$. By Proposition 6.4, δ is a fixed point of Φ_4, and so $\delta \sqsubseteq \mathit{gfp}(\Phi_4)$. By antitony of $\bigcirc$,

$$\bigcirc(\mathit{gfp}(\Phi_4)) \sqsubseteq \bigcirc(\delta) = \bigcirc(\bigcirc(\mathit{lfp}(\Phi_4))) = \mathit{lfp}(\Phi_4)$$

Since, by Proposition 6.4 $\bigcirc(\mathit{gfp}(\Phi_4))$ is a fixed point of Φ_4, while $\mathit{lfp}(\Phi_4)$ is its *least* fixed point, it follows that $\bigcirc(\mathit{gfp}(\Phi_4)) = \mathit{lfp}(\Phi_4)$, that is, $\mathit{lfp}(\Phi_{\mathbf{4}}) \between \mathit{gfp}(\Phi_{\mathbf{4}})$. ■

Contraduals simply swap $\top$ with $\bot$, and the set of fixed points of $\Phi_{\mathbf{4}}$ is symmetric in the information order, owing to the symmetry of the underlying domain and the connectives we use. The two-valued functions are in the middle of the information ordering. Functions below produce $\bot$ in some cases and functions above produce $\top$ in some cases (additionally, there are incomparable functions which produce both $\bot$ and $\top$). For combinational circuits, $\mathit{lfp}(\Phi_{\mathbf{4}})$ is two-valued, so there are no fixed points strictly below or strictly above the horizontal line of symmetry—we are left with a single fixed point.

Proposition 6.6: $\mathit{lfp}(\Phi_{\mathbf{4}})$ is two-valued iff $\Phi_{\mathbf{4}}$ has a single fixed point.

Proof: Suppose the positive content of $\mathit{lfp}(\Phi_{\mathbf{4}})$ is δ. Since $\mathit{lfp}(\Phi_{\mathbf{4}})$ is two-valued, it must be $(\delta, \neg\delta)$. By Proposition 6.4, $\mathit{gfp}(\Phi_{\mathbf{4}})$ is the contradual, which is also $(\delta, \neg\delta)$. Thus $\mathit{lfp}(\Phi_{\mathbf{4}}) = \mathit{gfp}(\Phi_{\mathbf{4}})$ is the only fixed point. Conversely, if $\mathit{lfp}(\Phi_{\mathbf{4}}) = \mathit{gfp}(\Phi_{\mathbf{4}})$ then, by Propositions 6.3 and 6.4, $\mathit{lfp}(\Phi_{\mathbf{4}})$ is two-valued. ■

Thus for combinational circuits, Kleene iteration from any element will reach the least (and only) fixed point.

Proposition 6.7: If $\Phi_{\mathbf{4}}$ has a single fixed point δ then the sequence $\mathit{iter}_{\Phi_{\mathbf{2}}}(x)$ has final element δ, irrespective of starting point x.

Proof: This follows from Propositions 6.2 and 6.6. ■

VII. Conclusion

The description of circuits by means of Boolean equations is standard. In the presence of recursively defined truth value functions, however, it is natural to ask about the interpretation and well-definedness of the definitions. It is also natural to ask about the well-behavedness of the circuits thus defined, and the relation between properties of a set of recursive equations and behavioural properties of the corresponding circuit.

An appealing aspect of the multiple-valued interpretation is that it is sound with respect to very reasonable assumptions about gate delays [8], namely the so-called up-bounded inertial delay model [6], under which a delay may vary in time, even for the same gate, but within some upper bound. As Example 5.3 shows, this is not the case for a criterion such as the existence of a unique two-valued solution.

Four-valued logic is clearly a useful tool for reasoning not only about circuit (mis-)behaviour, but also for specification purposes, in particular for the incorporation of "don't care" conditions. In this context, we have shown that the question of what it means for a partial Boolean function to implement a specification that involves "don't care" cases is most naturally answered in four-valued logic using a notion of contra-duality.

In summary, a four-valued approach delivers expressive specification, expressive circuit semantics, and useful notions of circuit correctness, all within a single logical framework. An interesting line of research would be to explore how the added expressiveness can be used to simplify cyclic circuits along the lines demonstrated towards the end of Section IV. In Section II we mentioned the interpolation-based approach to circuit synthesis proposed by Lee *et al.* [21], and its extension to the cyclic case by Backes and Riedel [22]. That approach, as presented, takes a two-valued specification as its starting point. An interesting direction for future work would be to revisit the SAT encoding used by Backes and Riedel, so as to make it capture reasoning about our "implements" relation (a relation between four-valued specifications and three-valued functions), rather than classical entailment. This potentially allows for the generation of smaller correct circuits.

The introduction of a fourth value has little impact on the formulation of classical symbolic ternary analysis. We have shown that viewing the analysis of a circuit as propagation of *four* truth values helps explain certain contra-duality phenomena that occur when we compare two-valued fixed points with multiple-valued fixed points. We have also shown that, for combinational circuits, two-valued Kleene iteration must lead to a unique two-valued fixed point, irrespective of starting point, and even when the iterating function is non-monotone and Kleene sequences fail to be chains.

References

[1] C. E. Shannon, "A symbolic analysis of relay and switching circuits," *Trans. AIEE*, vol. 57, pp. 713–723, 1938.

[2] H. H. Aiken and others, *Synthesis of Electronic Computing and Control Circuits*. Harvard University Press, 1951.

[3] E. B. Eichelberger, "Hazard detection in combinational and sequential switching circuits," *IBM Journal of Research and Development*, vol. 9, no. 2, pp. 90–99, 1965.

[4] W. H. Kautz, "The necessity of closed circuit loops in minimal combinational circuits," *IEEE Trans. Comput.*, vol. C-19, pp. 162–164, 1970.

[5] J. A. Brzozowski and C.-J. H. Seger, "Advances in asynchronous circuit theory, part I: Gate and unbounded inertial delay models," *Bull. EATCS*, vol. 42, pp. 198–249, 1990.

[6] ——, "Advances in asynchronous circuit theory, part II: Bounded inertial delay models, MOS circuits, design techniques," *Bull. EATCS*, vol. 43, pp. 199–263, 1991.

[7] G. Berry, "The constructive semantics of pure Esterel: Draft version 3," Technical Report, INRIA Sophia-Antipolis, 1999, *Theory and Practice of Logic Programming*, to appear; http://arxiv.org/abs/1305.0141.

[8] T. R. Shiple, "Formal analysis of synchronous circuits," Ph.D. dissertation, UC Berkeley, 1996.

[9] M. Mendler, T. R. Shiple, and G. Berry, "Constructive Boolean circuits and the exactness of timed ternary simulation," *Formal Methods in System Design*, vol. 40, pp. 283–329, 2012.

[10] M. Ginsberg, "Multivalued logics: A uniform approach to reasoning in artificial intelligence," *Computational Intelligence*, vol. 4, no. 3, pp. 265–316, 1988.

[11] M. Chechik, B. Devereux, S. Easterbrook, and A. Gurfinkel, "Multi-valued symbolic model-checking," *ACM Transactions on Software Engineering and Methodology*, vol. 12, no. 4, pp. 371–408, 2003.

[12] M. Fitting, "Kleene's logic, generalized," *Journal of Logic and Computation*, vol. 1, no. 6, pp. 797–810, 1991.

[13] L. Naish and H. Søndergaard, "Truth versus information in logic programming," *Theory and Practice of Logic Programming*, to appear; http://arxiv.org/abs/1305.0141.

[14] S. Malik, "Analysis of cyclic combinational circuits," in *Proc. IEEE Conf. Comput.-Aided Des.*, 1993, pp. 618–625.

[15] K. S. Namjoshi and R. P. Kurshan, "Efficient analysis of cyclic definitions," in *Computer-Aided Verification*, ser. Lecture Notes in Computer Science, N. Halbwachs and D. Peled, Eds., vol. 1633. Springer, 1999, pp. 394–405.

[16] T. R. Shiple, G. Berry, and H. Touati, "Constructive analysis of cyclic circuits," in *Proceedings of the 1996 European Design and Test Conference*, 1996, pp. 328–333.

[17] N. Halbwachs and F. Maraninchi, "On the symbolic analysis of combinational loops in circuits and synchronous programs," in *Proc. Euromicro '95*, 1995.

[18] C. Wrathall, "Complete sets and the polynomial-time hierarchy," *Theor. Comput. Sci.*, vol. 3, no. 1, pp. 23–33, 1976.

[19] J.-H. R. Jiang, A. Mishchenko, and R. K. Brayton, "On breakable cyclic definitions," in *Proc. IEEE Conf. Comput.-Aided Des.*, 2004, pp. 411–418.

[20] M. D. Riedel, "Cyclic combinational circuits," Ph.D. dissertation, California Institute of Technology, 2004.

[21] C.-C. Lee, J.-H. R. Jiang, C.-Y. Huang, and A. Mishchenko, "Scalable exploration of functional dependency by interpolation and incremental SAT solving," in *Proc. IEEE Conf. Comput.-Aided Des.*, 2007, pp. 227–233.

[22] J. Backes and M. D. Riedel, "The synthesis of cyclic dependencies with Boolean satisfiability," *ACM Trans. Design Automation of Electronic Systems*, vol. 17, no. 4, pp. 44:1–44:24, 2012.

[23] G. Birkhoff, *Lattice theory*, 3rd ed., ser. American Mathematical Society Colloquium Publication. Providence: American Mathematical Society, 1967, no. XXV.

[24] A. Tarski, "A lattice-theoretical theorem and its applications," *Pacific Journal of Mathematics*, vol. 5, pp. 285–309, 1955.

[25] S. C. Kleene, "On notation for ordinal numbers," *The Journal of Symbolic Logic*, vol. 3, pp. 150–155, 1938.

[26] N. D. Belnap, "A useful four-valued logic," in *Modern Uses of Multiple-Valued Logic*, J. M. Dunn and G. Epstein, Eds. D. Reidel, 1977, pp. 8–37.

[27] P. R. Halmos, *Lectures on Boolean Algebras*. Springer, 1963.

[28] M. Fujita, P. McGeer, and J.-Y. Yang, "Multi-terminal binary decision diagrams: An efficient data structure for matrix representation," *Formal Methods in System Design*, vol. 10, no. 2-3, pp. 149–169, 1997.

[29] M. Damiani and G. De Micheli, "Don't care set specifications in combinational and synchronous logic circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 3, pp. 365–388, 1993.

[30] S. Tripakis, B. Lickly, T. A. Henzinger, and E. A. Lee, "A theory of synchronous relational interfaces," *ACM Transactions on Programming Languages and Systems*, vol. 33, no. 4, 2011, article 14.

[31] L. De Alfaro and T. A. Henzinger, "Interface theories for component-based design," in *Embedded Software: Proceedings of the First International Workshop*, ser. Lecture Notes in Computer Science, T. A. Henzinger and C. M. Kirsch, Eds., vol. 2211. Springer, 2001, pp. 148–165.

[32] M. Riedel and J. Bruck, "The synthesis of cyclic combinational circuits," in *Proc. Design Automation Conf.* ACM Press, 2003, pp. 163–168.
[33] R. E. Bryant, "Boolean analysis of MOS circuits," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 6, no. 4, pp. 634–649, 1987.
[34] S. Malik, "Analysis of cyclic combinational circuits," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 13, no. 7, pp. 950–956, 1994.
[35] R. Rivest, "The necessity of feedback in minimal monotone combinational circuits," *IEEE Trans. Comput.*, vol. C-26, pp. 606–607, 1977.