A Canonical Form for Circuit Diagrams

Kathi Fisler*
Department of Computer Science
Lindley Hall 215
Indiana University
Bloomington, IN 47405
kfisler@cs.indiana.edu

May 1, 1995

Abstract

A canonical form for circuit diagrams with a designated start state is presented. The form is based upon finite automata minimization and Shannon's canonical form for boolean expressions.

1 Introduction

This paper presents a canonical form for circuit diagrams with a designated start state. It builds upon our previous work in the logic of hardware diagrams [Fis94]. This canonical form will be used to prove a set of inference rules on circuit diagrams logically complete in a companion paper. We present a general discussion of our canonical form in section 2, a background of circuit diagram logic in section 3 and the canonical form algorithm in section 4.

2 Canonical Forms and Circuit Diagrams

In section 3, we define an equivalence relation on circuit diagrams such that all diagrams in the same equivalence class are behaviorally equivalent. Intuitively, two circuits are behaviorally equivalent if and only if they have the same observable external behavior when operated on the same sets of inputs. We would like to be able to determine whether two circuits are behaviorally equivalent without the need for exhaustive simulation. One way to do this would be to select a unique representative from each equivalence class and to define a function that maps any circuit diagram to the representative of its class. Two circuit diagrams could then be compared for behavioral equivalence by applying the function to each diagram and seeing if the same representative circuit was returned. Assuming we can define both the function and the representatives, we refer to the function as a *canonicalizer* and say that those circuit diagrams in the range of the function are in *canonical form*.

A circuit diagram canonicalizer could take many forms. For example, given the correlation between circuit diagrams and state machines, we could design a canonicalizer that converted a circuit diagram into a Mealy/Moore machine, minimized the state machine, then used a fixed implementation technique to convert the state machine back into a circuit diagram. The only drawback to

*Research supported by AT&T Bell Laboratories through the PhD Fellowship Program.

this approach is that it would not help us achieve the eventual goal of proving the completeness of our set of circuit diagram transformations; proving completeness of the transformations requires that our algorithm remain at the level of the circuit diagram at all times.

The key to remaining at the circuit diagram level is what we call the *one-hot* representation of a circuit diagram. There are many techniques for implementing state machines in physical hardware; two examples are the state-encoding method and the one-hot method [PW87]. In state-encoding, the states of the device are encoded in binary labels with length logarithmic base two in the number of states in the machine. State encoded designs are compact in the number of delay elements they require and are thus quite common. However, in such designs the names of the states become hardwired into the physical realization of the machine, with the result that two devices isomorphic with respect to state transitions can appear quite different structurally. This is a significant problem for our project; we need to be able to easily visualize isomorphism from the diagrammatic structure of a circuit.

The one-hot method, on the other hand, uses one delay element per state in the machine being realized. While this results in exponentially many more delay elements than required by the state-encoding method, one-hot implementations do not hardwire state names into the circuitry. Essentially, one-hot implementations are as close in structural appearance to state machines as we can get; this property makes them very useful for checking circuits for structural isomorphism.

Assuming we can use automata-theoretic minimization theory to canonicalize the transition structure of a circuit diagram, all that remains is that we be able to canonicalize blocks of combinational logic in a circuit. Combinational logic blocks correspond naturally to boolean expressions and there are many known normal forms for boolean expressions, such as conjunctive and disjunctive normal forms (CNF and DNF). However, it is possible to have two logically equivalent CNF or DNF boolean expressions that are not structurally identical due to associative and commutative operations. Given that our circuit diagrams employ strictly binary gates, these operations are problematic when checking for structural isomorphism. As a result, we need to impose some organization within the individual clauses in order to achieve isomorphism. Our combinational logic canonicalizer is essentially the same as Shannon's canonicalizer for boolean expressions [Sha38], which imposes organization by using an ordering on the variables within the expression.

Several papers have addressed the problem of determining observational equivalence between sequential machines. Myhill-Nerode based minimization algorithms [HU79] yield canonical forms for deterministic finite automata; similar results exist for Moore machines and general sequential machines which encorporate Mealy outputs as well as Moore outputs [Gin62] [Moo56]. The spirit of the minimizations for each type of machine are analogous, though this work most closely follows that of [Gin62]. Each of these forms concentrates on the transition structure of a system only; they do not address the canonicalization of the boolean expressions that represent the state transition functions. Canonical forms for boolean expressions such as Shannon's canonical form [Sha38] and binary decision diagrams [Bry86] are well known, but do not provide any means for handling sequential components.

Our notion of behavioral equivalence, defined with respect to single, fixed start states, is more restrictive than some other classifications of behavioral equivalence such as that proposed in [SP94]. We have chosen to use the simpler, more restrictive definition in order to simplify our current project. Our present interest is in studying the formal use of diagrams in hardware reasoning; the use of a simpler definition allows us to focus more on the logical aspects of the research.

3 The Logic of Circuit Diagrams

Much of the material in this section is provided for reference from [Fis94]. The original logic has been modified slightly in order to handle start states. In addition, we provide a few new definitions required by the proofs in this paper.

In this logic, we consider circuit diagrams composed of icons representing binary and and or gates, unit delay elements, inverters, and wires. We use a set-theoretic model to capture the syntactic information contained in a given circuit diagram. In this model, we represent a wire line as an ordered pair of the points it connects; a binary gate icon is modelled as an ordered triple $\langle x, y, z \rangle$, indicating that the icon connects x and y on the left, in that order, from top to bottom, with z on the right. Unary icons are similarly represented with a two-tuple.

Specifically, a *circuit sketch* s is a tuple $\langle P, I, O, W, N, D, A, R \rangle$ where P is a set of objects called the *connection points* of s, I and O are disjoint subsets of P called the input points and output points of s, W, N, and D are disjoint subsets of $P \times P$ called the *wire lines*, *negation icons*, and *delay icons*, and A and R are disjoint subsets of $P \times P \times P$ called the and *gate icons* and or *gate icons*. A wire $w = \langle p_a, p_b \rangle$ is *branch-free* iff p_b is contained in only one element of $N \cup D \cup A \cup R$; otherwise w is a *branching* wire. We assume the existence of two special connection points which we call *power* and *ground*; these are denoted by T and F, and correspond to the logical values true and false, respectively.

We use the term circuit *sketch* as opposed to circuit *diagram* so that we may reserve the latter term for only those diagrams which are well-formed. A formal definition of this term will be presented shortly.

The above model is syntactic; we still need to give these diagrams a semantics. The common factor to all the diagrams covered in our overall study (circuit diagrams, state machines, and timing diagrams) is that they all describe physical hardware. It is therefore quite reasonable that we should use a model of physical hardware as a basis for assigning semantics to diagrams. In the particular case of circuit diagrams, they have a natural homomorphism with physical devices as defined below. This will make the semantic definitions of this work seem rather trivial, but the semantics are more complicated when capturing other representations [Fis94].

3.1 Physical Devices

Our model of physical hardware is defined in two stages. First, we capture the structural aspects of a device along with its interface with the external environment. Later, we augment this model so that it can express how a device behaves over time while interacting with the environment. We assume that wires in devices can carry values in $\{0, 1\}$. Ports are primitive objects providing connection points for wires. We choose to associate voltage values with ports; the term *assignment* will refer to a total function from a set of ports to the set $\{0, 1\}$. Two assignments are *consistent* if they agree on the values of each port in the domain of both assignments. It is often convenient to assign names to ports. If the name n is associated with a port p, we will refer to p as an *n-port*. The term *negative port* will refer to a port that is wired to the input of an inverter; *positive ports* are all ports that are not negative.

We assume that devices are composed only of binary and and or gates, inverters, and unit delay elements; the term *basic component* can refer to any of these items. Specifically, a basic component is a tuple $\langle I, O, F, D \rangle$ where I and O are disjoint sets of ports called the input ports and output ports of c, respectively; F is a function which assigns each $p \in O$ a function F_p from assignments on I into U; and D is a function from O to the non-negative integers called the output delay function. We assume that delay elements have delay one and all other basic components have delay zero.

Capturing the structural aspects of a physical component, an *abstract device* is a 5-tuple $D = \langle I, O, B, W, c\rangle$ where I and O are disjoint sets of ports providing the external input and output interface to D, B is a set of gates and unit delay elements, W is a set of wires and c is a function from W to sets of ports. We assume that all ports in a device are distinct and that the sets $c(w)$ partition the ports of D.

An abstract device is called *combinational* if it does not contain any delay elements; all other abstract devices are called *sequential*. Given an abstract device D, the elements of $I \cup O$ are called the *interface ports* of D. All other ports in D are called *internal ports*. *Internal input ports* are those internal ports serving as an input port to some basic component b in D; *internal output ports*, *input interface ports* and *output interface ports* are defined analogously.

Structural comparisons between devices are based upon their graph-like structure; such comparisons examine the paths between ports within devices. Given an abstract device D, there is a *connecting step* from port p_i to port p_j, denoted $p_i \rightsquigarrow p_j$, iff either p_i is an input port and p_j the output port to some basic component, or p_i is an internal output port or an input interface port, p_j is an internal input port or an output interface port and p_i and p_j are connected by some wire w. A finite transitive chain of connecting steps is called a *connecting path* and denoted by $p_0 \rightsquigarrow^* p_n$; a device contains a connecting cycle if $p \rightsquigarrow^* p$ for some port p.

Although any abstract device corresponds to a piece of physical hardware, we are only interested in considering those that meet certain well-formedness conditions: treating the basic components as nodes and the wiring function as giving rise to edges should yield a connected and directed graph, for each internal port p in D there must exist a connecting path from p to an element of O, and every connecting cycle in D must contain some step $p_i \rightsquigarrow p_j$ such that the connecting element of p_i and p_j is a delay element. Any abstract device meeting these conditions is called *well-connected*.

The canonical form problem requires that we be able to identify when two abstract devices are structurally isomorphic. Formally, we will say that abstract devices $D_1 = \langle I_1, O_1, B_1, W_1, c_1\rangle$ and $D_2 = \langle I_2, O_2, B_2, W_2, c_2\rangle$ are *isomorphic* if there exists a bijection ϕ from the ports of D_1 to the ports of D_2 such that: $\phi(I_1) = I_2$, $\phi(O_1) = O_2$, for each $b_1 \in B_1$, there exists a $b_2 \in B_2$ such that the restriction of ϕ to the input ports of b_1 maps to the input ports of b_2 and the restriction of ϕ to the output port of b_1 maps to the output port of b_2, and for all ports p_a and p_b in D_1, p_a and p_b are wired together iff $\phi(p_a)$ and $\phi(p_b)$ are wired together.

A subdevice $D' = \langle I', O', B', W', c'\rangle$ of a device $D = \langle I, O, B, W, c\rangle$ is an abstract device such that each component in D' is a subset of its corresponding component in D. In this work we consider only well-connected subdevices. D' will be called *conjunctive* iff B' contains only and gates. D' is *maximally conjunctive* iff it is conjunctive and for all $b \in B'$, any and gate wired to b by a branch-free wire is also an element of B'. D' is *maximally combinational* if it is combinational and for all $b \in B'$, any combinational gate wired to b by a branch-free wire is also an element of B'. A gate in a subdevice is in *input position* if none of its input ports are wired to the output of another gate.

We now extend our terminology to address the computational behavior of devices. Some additional information is needed in order to view an abstract device as a physical object in mid-computation; specifically, we need to know the values on the ports of the device. This work also requires that we know the start state of the device, where a start state can be viewed as an assignment to the delay element output ports. A tuple $\langle D, s\rangle$ consisting of an abstract device and a start state will be called an *initial device*. A tuple $\langle I, i\rangle$ consisting of an initial device and an assignment to the ports of the associated abstract device will be called a *concrete device*. Most of the definitions that follow do not rely upon the start state component of an initial device; we will write concrete devices as tuples of the abstract device and the port assignments in such cases. We

require that all assignments included as part of a concrete device are consistent with the structure of the device. That is, if D is an abstract device and i an assignment, for all gates g in D, the value in i on the output port of g is consistent with the values in i for the input ports of g and the function associated with g.

Definitions provided for abstract devices apply to concrete devices that contain them. For example, a concrete device is well-connected if the abstract device it contains is well-connected. Similarly, two concrete devices are isomorphic iff their respective abstract devices are isomorphic.

We are interested in determining when two devices exhibit the same external behavior. This requires that we be able to operate our devices over time. We established in [Fis94] that given a concrete device and an assignment to its input ports, there exists a unique concrete device reflecting the new values on the delay element and interface output ports; this unique device will be said to *follow* from the original device. The term *assignment sequence* refers to a sequence of assignments $i_1, i_2, \ldots, i_k$ to the interface input ports of a device. Given concrete device $C = \langle D, i \rangle$, a *run* of C under $\langle i_1, i_2, \ldots, i_k \rangle$ is a sequence $\langle C_0, C_1, \ldots, C_k \rangle$ of concrete devices such that $C_0 = C$, and for each $k \geq j \geq 1$, C_j is the concrete device that follows from C_{j-1} given i_j. The *output* of a concrete device $\langle D, i \rangle$ is the restriction of i to the interface output ports O of D. The output of a run $\langle C_0, C_1, \ldots, C_k \rangle$ is a sequence $\langle O_0, O_1, \ldots, O_k \rangle$ where each O_i is the output of C_i. The state of devices and runs are similarly defined by restricting assignments to the delay output ports.

Two delay elements are *computationally equivalent* if, for every assignment sequence, the output of the run of the device starting from one delay state is the identical to the output of the run of the device starting from the other delay state. A circuit is said to be *minimized* if it contains no pair of computationally equivalent delay elements. We will say that state s_2 is *computationally reachable* from state s_1 if there exists an assignment on which the device transitions from s_1 to s_2. Concrete devices $C_1 = \langle \langle D_1, s_1 \rangle, i_1 \rangle$ and $C_2 = \langle \langle D_2, s_2 \rangle, i_2 \rangle$ are *behaviorally equivalent* if D_1 and D_2 have the same sets of input and output interface ports, i_1 and i_2 are consistent with s_1 and s_2 respectively, and for every assignment sequence a for D_1, the output of the run of C_1 under a is the same as the output of the run of C_2 under a. This definition of behavioral equivalence gives rise to a natural equivalence relation on devices in which all devices in a single class are behaviorally equivalent to one another.

A related notion to that of behavioral equivalence is that which we call *isomorphic with respect to state transitions*. Concrete devices C_1 and C_2 will have this property iff there exists a bijection ϕ between the states of C_1 and the states of C_2 such that ϕ relates the start states of C_1 and C_2 and C_1 transitions from s to s' on input i iff C_2 transitions from $\phi(s)$ to $\phi(s')$ on input i. C_1 and C_2 may be isomorphic with respect to state transitions without being isomorphic because the combinational logic capturing the transitions could differ in implementation between the two circuits. Similarly, C_1 and C_2 may be isomorphic with respect to state transitions without being behaviorally equivalent because no conditions are placed on the outputs produced by the two circuits. It will follow, however, that minimized behaviorally equivalent circuits are isomorphic with respect to state transitions, as we will prove later.

Lemma 1 *Isomorphic devices are behaviorally equivalent.*

We now have enough information to be able to assign semantics to circuit sketches.

Definition 1 Let s be a circuit sketch and D be an abstract device.

1. A *depiction map* from s to D is an injective function ϕ from the connection points of s into the ports of D such that for all $p \in s_P$;

(a) $p \in s_I \rightarrow \phi(p) \in D_I$.

(b) $p \in s_O \rightarrow \phi(p) \in D_O$.

(c) If $l = \langle p_a, p_b \rangle$ is a wire line of s then $\phi(p_a)$ and $\phi(p_b)$ are wired together by some wire $w \in D_W$; $\phi(p_a)$ must be an input interface port or internal output port and $\phi(p_b)$ must be an output interface port or internal input port.

(d) If $n = \langle p_a, p_b \rangle$ is a negation icon of s then $\phi(p_a)$ is connected to the input port and $\phi(p_b)$ connected to the output port of some inverter in D_G.

(e) If $d = \langle p_a, p_b \rangle$ is a delay icon of s then $\phi(p_a)$ is connected to the input port and $\phi(p_b)$ connected to the output port of some delay element in D_R.

(f) $g = \langle p_a, p_b, p_c \rangle$ is an and-gate icon of s then $\phi(p_a)$ and $\phi(p_b)$ are connected to the input ports and $\phi(p_b)$ connected to the output port of some and-gate in D_G.

(g) If $g = \langle p_a, p_b, p_c \rangle$ is an or-gate icon of s then $\phi(p_a)$ and $\phi(p_b)$ are connected to the input ports and $\phi(p_b)$ connected to the output port of some or-gate in D_G.

2. D is a *structural implementation of* s if there is a surjective depiction map from s to D and the converse of requirements 1c through 1g in the definition of a depiction map hold. This is written $D \models_s s$, where the subscript stands for "structural".

3. D is a *behavioral implementation of* s if D is behaviorally equivalent to some device D' which is a structural implementation of s. This is written $D \models_b s$.

Lemma 2 *For any device D there exists a circuit sketch s that is unique up to isomorphism on circuit sketches such that $D \models_s s$. For any circuit sketch s there exists a device D that is unique up to isomorphism on devices such that $D \models_s s$.*

One final note on terminology: due to the natural homomorphism between devices and circuit sketches reflected by this lemma, many of the definitions given earlier on devices could apply to circuit sketches as well. Rather than recast the earlier definitions in terms of the structure of circuit sketches, we rely on the following convention to provide a bridge between the two.

> Let s be a circuit sketch, let D be a device such that $D \models_s s$, and let ϕ be the depiction map from s to D. For any definition F on devices, F applies to s by replacing each reference to a port $p \in D$ with a reference to the connection point $t \in s$ such that $\phi(t) = p$.

This observation allows us to talk about behavioral equivalence classes on circuit diagrams. We can also use it to make a deferred definition: a *circuit diagram* is any circuit sketch s for which the device D such that $D \models_s s$ is well-connected. For the remainder of this work, we assume we are dealing only with circuit diagrams, as opposed to sketches.

4 The Canonicalization Algorithm

We first give a high-level view of our algorithm, expanding the details of each step in the sections that follow. The proof that the algorithm is a canonicalizer appears at the end of the section.

All canonical form circuits are one-hot circuits in which each delay element corresponds to a reachable state. In addition, the circuits are minimized in the automata-theoretic sense — no two delay elements play the same computational role in the circuit — and all blocks of combinational logic are in a combinational canonical form. The steps of the algorithm are as follows:

Algorithm 1 Let C be a circuit diagram and let $(v_0, \ldots, v_k)$ be an ordering on the input interface and delay element output ports.

1. Convert C to one-hot-form.
2. Push all branching points as close to input interface and delay output ports as possible.
3. Canonicalize all maximally combinational subcircuits leading into interface output and delay input ports with respect to $(v_0, \ldots, v_k)$.
4. Remove unreachable states.
5. Minimize with respect to state transitions.
6. Construct a reachability ordering on the delay element output ports.
7. Re-canonicalize all maximally combinational subcircuits leading into interface output and delay input ports with respect to the concatenation of the ordering in step 6 and an ordering on the input ports.

We push branching points to insure that no two circuits fail to be structurally isomorphic on the basis of optimizations in the use of gates in one circuit. Each of the two combinational logic canonicalization steps is needed: the first to aid us in identifying unreachable states and the second to clean up any residual redundancies in circuitry left after the minimization stage.

The need for the reachability ordering computed in step 6 is more subtle. When comparing two behaviorally equivalent circuits, the algorithm is only guaranteed to return isomorphic canonical forms if the circuits are canonicalized on the same variable ordering. While any ordering on the input ports would apply to each of the circuits being compared, determining how to order the delay output ports in the same way across two circuits is more difficult. This analysis provides of means of ordering the delay output variables consistently across the two circuits so that the outcome of the canonicalizer is not affected.

4.1 Converting to One-Hot Form

As stated in the introduction, a one-hot circuit diagram is one in which exactly one delay element output port can have value 1 at any given time. The following algorithm converts any circuit diagram into a behaviorally equivalent diagram in one-hot form. Given the bijective correspondence between delay elements and states in this form, we will often use delay elements and states interchangeably on circuits that we know to be one-hot. When doing so, we assume that the delay output port assignment that traditionally represents a state maps the asserted delay element to true and all other delay elements to false.

Algorithm 2

1. Let C be a circuit diagram with n delay elements. Choose some ordering $r_1, \ldots, r_n$ for these delay elements. Add 2^n new delay elements to C and associate with each a unique boolean combination of $r_1, \ldots, r_n$.
2. For each of these new delay elements d, create an n-ary conjunction with one input for each r_i. If r_i is positive in the boolean combination for d, attach the input of r_i to the input wire for r_i in the conjunction. Otherwise, attach the input of r_i to the input of a new inverter and attach the output of that inverter to the input wire for r_i in the conjunction.

3. For each r_i, create a disjunction j_t of the outputs of all of the new delay elements in which the boolean value of r_i is true. Create a disjunction j_f of the outputs of all of the new delay elements in which the boolean value of r_i is false. Connect any wire attached to the output port of r_i to the disjunction of j_t and j_f.

4. Remove $r_1, \ldots, r_n$ from C. Call this new circuit diagram C_1. The start state of C_1 will be the delay element whose corresponding boolean combination captures the start state of C.

Lemma 3 *C_1 is a well-formed circuit diagram.*

Proof This is true by construction. Given that C was well-formed by assumption, the input and output ports of the delay elements in C were each attached to some wire in C. The wires connected to the delay ports in C were all reattached to the new circuitry, and all new delay input and output ports are also attached in the construction. Therefore C_1 is a single connected component. The rest of the well-formedness conditions follow trivially. □

Lemma 4 *Delay element d_1 is asserted in C_1 iff the delay elements in C_1 have the values associated with the boolean combination for d_1.*

Proof This is true by construction. □

Lemma 5 *C_1 is a one-hot circuit diagram.*

Proof By construction, there is a one-to-one mapping between the possible states of C and the delay elements in C_1. Given that C is well-formed and hence deterministic, it follows that C has exactly one state at any point in time. It follows using lemma 4 that exactly one delay element in C_1 can be asserted at any time, so C_1 is a one-hot circuit.

□

Lemma 6 *Given states d and d_1 of C and C_1 respectively, if d_1 corresponds to the boolean combination representing d, then for all input assignments a and for all outputs o, o is asserted in d under a iff o is asserted in d_1 under a.*

Proof The only change in the circuitry computing o from C to C_1 is that the wires coming from delay elements in C have been attached in C_1 by disjunctions as described in algorithm 2. Let w be some wire involved in the computation of o in C that was attached to the output port of some delay element d'. By construction, w carries a high value in C_1 iff the state of C_1 corresponds to a boolean combination in which the value of d' is true. If d' is true in state d, it follows that d' must have value true in the boolean combination for d_1. Therefore, w is asserted in C iff w is asserted in C_1. It follows that o is asserted in d under a iff o is asserted in $Dd1$ under a.

□

Lemma 7 *C_1 is behaviorally equivalent to C.*

Proof Let A be an assignment sequence for C; since the one-hot conversion process does not affect the input ports, A is also an assignment sequence for C_1. Let R and R_1 be the runs of C and C_1 from their start states under A, respectively. We need to prove that the output of each stage of R is identical to the output of the same stage in R_1. By lemma 6, the outputs are identical at the start states. It follows inductively from lemmas 4 and 6 that C_1 is behaviorally equivalent to C.

□

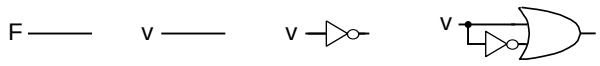


Figure 1: Valid canonical forms on a single variable v.

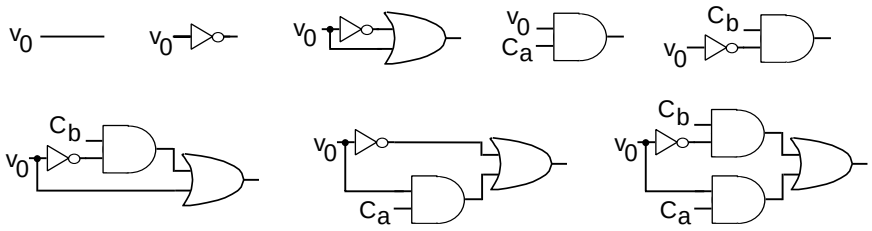


Figure 2: Valid canonical forms on multiple variables. v_0 is the first port name in the ordering and C_a and C_b are canonical with respect to the rest of the ordering.

Lemma 8 *Let r and r' be distinct orderings on the delay elements of C and let C_1 and C_1' be the one-hot circuits derived from C under r and r', respectively. C_1 and C_1' are isomorphic.*

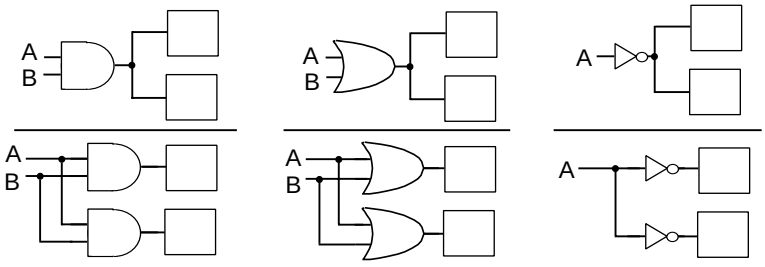
Proof By construction, each of C_1 and C_1' contains a delay element for each combination of values on delay elements of C. Let ϕ be a bijection from the delay elements of C_1 to those of C_1' such that d and $\phi(d)$ agree on the combination of values assigned to the delay elements of C.

Given that d and $\phi(d)$ correspond to the same delay values, it follows by construction that the inputs to d and $\phi(d)$ are isomorphic. Therefore, the two circuits are isomorphic after step 2 of the one-hot conversion algorithm. By similar reasoning, the two circuits are still isomorphic after step 3 of the algorithm. Step 4 simply removes the original delay elements that were contained in both circuits, so the resulting circuits C_1 and C_1' must be isomorphic.

□

4.2 Pushing Branching Points

In order to eventually compare circuits for isomorphism, we need some fixed placement to use for branching points. We choose to push all branching points as close to input interface and delay element output ports as possible by iterating the transformations pictured below until these rules no longer apply. Let C_2 refer to the circuit that results from this iteration on C_1. Given that these rules introduce no changes in functionality, C_1 and C_2 are behaviorally equivalent.



4.3 Canonicalizing Boolean Logic

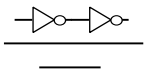
Our combinational logic canonicalization procedure is a diagrammatic version of Shannon's canonical form [Sha38]. Figure 1 shows the valid canonical forms for expressions on a single variable, while

figure 2 shows the valid canonical forms for expressions on more than one variable. The remainder of this section presents our implementation of Shannon's form and proves it canonical with respect to the diagrams. The algorithm converts any well-formed combinational circuit diagram C_O into canonical form relative to a finite length ordering $(v_0, v_1, \ldots, v_k)$ of the input interface ports. The variable ordering must be of length at least one since a wire is the simplest circuit diagram in our system.

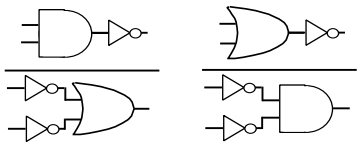
Algorithm 3 In these rules, v_0 can represent either v_0 or $\neg v_0$, T and F represent power and ground, and all other letters are subcircuits other than wires directly to v or $\neg v$. Possible applications of the rules should be checked in the order that the rules are listed within each section. The commutative rules on and gate and or gate inputs may be applied any time as necessary for applications of the rules.

1. Convert C_0 into a diagrammatic analog of disjunctive normal form, applying the rules (a) and (b) until no more applications are possible, followed by applications of (c) until no more applications are possible.

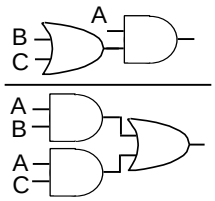
   (a) Remove any pairs of consecutive inverters.

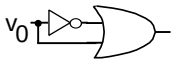
   (b) If an inverter is in non input position, use DeMorgan's Law to move it closer to input position.

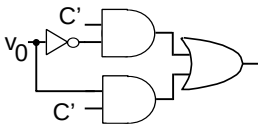
   (c) If there exists a connecting path from an or gate to an and gate, pull the or gate forward with the distributive law.

2. Convert any subcircuits consisting of an inverter taking its input from ground to a wire to power and vice-versa. Replace any wires connected to power with a new circuit $v_0 \vee \neg v_0$.[1]

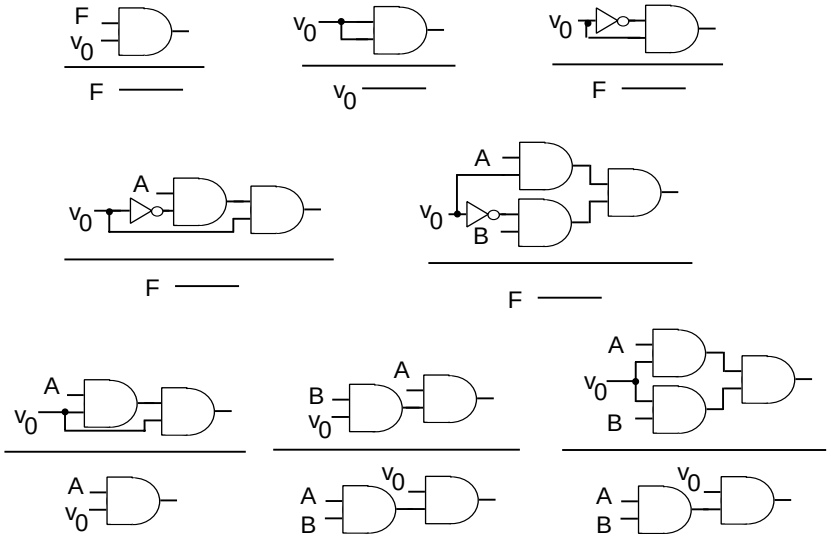
   

3. For each maximal conjunctive subcircuit C' of C_O, if C' does not contain a v_0-port, transform C' into

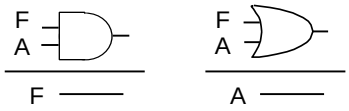
   

[1] We use a new circuit each time in order to insure that branching points remain at the input interface and delay output ports.

Otherwise, move v_0-ports towards the output ports of the diagram using the following rules, until C' has the form $v_0 \wedge B$, where B contains no v_0-ports or has been reduced to a wire to ground.

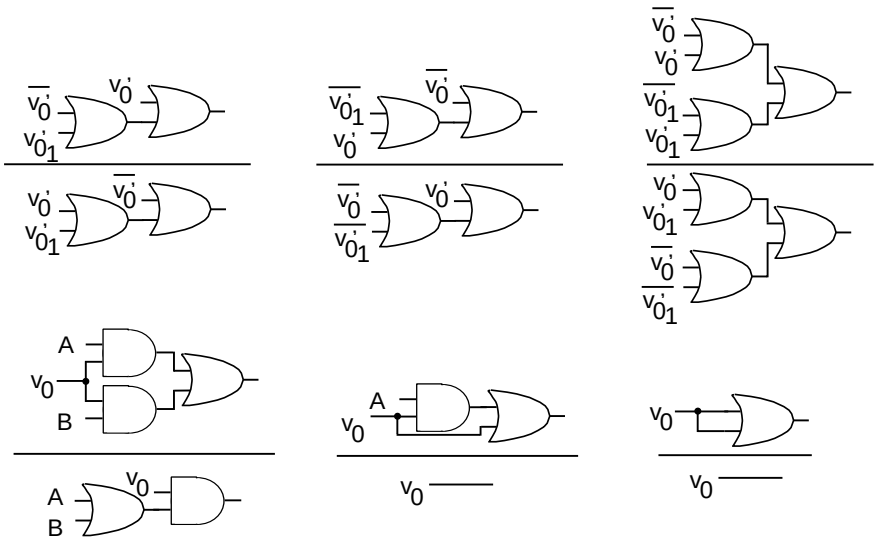


4. Iterate the following rules until no applications are possible:

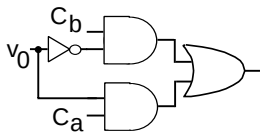


If the circuit has been reduced to a wire from ground, return from the algorithm.

5. Factor the occurrences of v_0 and $\neg v_0$ as follows, where $v_0{}'$, $v_0{}'_1$ are conjunctive clauses with a v_0-port and $\overline{v_0}'$, $\overline{v_0}'_1$ are conjunctive clauses with a $\neg v_0$-port (we prove later that each clause remaining at this point must be one of these):

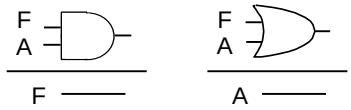


6. The circuit is now either a wire to ground, one of the valid canonical forms on a single variable, or in the following form, in which no v_0-ports appear in C_a or C_b:



If the rest of the variable ordering is not empty, canonicalize C_a and C_b relative to the ordering $(v_1, \ldots, v_k)$.

7. Iterate the following rules until no more applications are possible.



Lemma 9 *Following step 4, the circuit is either a wire from ground, a wire from a v_0-port, or a collection of disjunctions of conjunctions with all inverters at input positions, no wires attached to ground, and all v_0-ports at most one connecting step away from the input to a disjunction.*

Proof The rules in step 1 first move all inverters to the input ports of the circuit. Once the inverters have been moved, the relative positions of the or gates and and gates can be adjusted until one of the above structural forms is reached. Clearly, no inverters can be introduced into the later stages of the circuit while adjusting the positions of the or gates and and gates. At this point, the circuit is of one of the forms wire to ground, v_0-port or a disjunctions of conjunctions with inverters on input port wires.

Now assume we have a maximally conjunctive clause C' containing v_0-ports. If none of the rules listed in step 3 apply, then C' must be either a wire to a v_0-port or of the form $v_0 \wedge rest$ where $rest$ contains no v_0-ports, otherwise one of the rules would have to be applicable. Inverters are still in input position and the v_0-port is at most one connecting step away from the output port of C'; the output port of C' is wired to the input port of an or gate by definition. For any maximally conjunctive clauses not containing v_0-ports, the algorithm produces a circuit that is a disjunction of conjunctions in which each v_0-port is one connecting step from an or gate and inverters are still in input position. The only subcircuit that would not be maximally conjunctive would be a wire to ground, which is one of our acceptable circuits.

It follows that the only desired characteristic not satisfied by our circuit is that there may still be ground wires feeding into gates. Step 4 removes all such instances, but leaves inverters and v_0-ports in the desired position unless the entire circuit is reduced to a wire from ground.

□

Lemma 10 *The rules given in step 5 are sufficient to convert a circuit canonicalized up to step 4 into one of the forms pictured in figure 2.*

Proof It follows from lemma 9 that at the start of step 5, every or-gate input in the circuit is either another or-gate output, a wire to an v_0-port, or an and-gate with one input an v_0-port.

Assuming we have an or gate with each input a conjunctive clause on the same port, we need to show that we can collapse those clauses into a single clause. The clauses must be in one of the forms appearing in the second set of rules in step 5 by lemma 9, and in each case, the clause is collapsed to one containing a single port. Notice that each clause is still either an v_0-port or a conjunction with one input being an v_0-port.

It follows that any v_0-clauses not yet collapsed must be inputs to distinct or-gates. Since every clause is either an v_0-clause or a $\neg v_0$-clause by lemma 9, then one of the patterns listed in the first set of rules in step 5 must appear in the circuit. Applying the rule will allow for another collapse via the second set of rules. Iterating this process therefore must yield a circuit in which all v_0-clauses and all $\neg v_0$-clauses have been collapsed to subcircuits with at most one instance of each.

□

Corollary 1 *Following step 5, the circuit is in one of the forms shown in figure 2.*

Lemma 11 *Given a circuit on a single variable v_0, algorithm 3 returns one of the circuits shown in figure 1.*

Proof By lemma 10, following step 5 the circuit must be in a form such that there is at most one v_0-port and at most one $\neg v_0$-port. Given a circuit on only variable v_0, the only other possible gate input aside from v_0 and $\neg v_0$ is a wire from ground, seeing as all wires to power were eliminated in step 2 of the algorithm and no transformation rule introduces a wire from power into the circuit. Step 7 assures that all wires from ground are eliminated from the circuit unless the circuit reduces to a wire from ground, which is one of our desired circuits. The only other circuits on one input that do not involve wires to ground are v_0, $\neg v_0$, $v_0 \vee \neg v_0$, so the algorithm must return one of the desired forms given a circuit on a single input variable.

□

Lemma 12 *Given a variable ordering containing more than one variable, algorithm 3 returns a circuit that is either a wire to ground or one of the forms shown in figure 2.*

Proof The proof is by induction on the number of variables in the variable ordering. If the ordering contains only two variables, it follows from corollary 1 and lemma 11 that the circuit is in the desired form unless one of the canonicalizations on the second variable produced a wire to ground. In that case, the iterations in step 7 take care of either removing the wires to ground or reducing the circuit to a wire to ground, either of yields an acceptable circuit. Now assume the lemma holds on k variables. The proof that this works on $k+1$ variables is similar to the proof of the base case. Following the canonicalization on k variables, from corollary 1 and the inductive hypothesis we know that the circuit is in the desired form unless one of the canonicalizations on k variables produced a wire to ground. In that case, the iterations in step 7 take care of either removing the wires to ground or reducing the circuit to a wire to ground, either of yields an acceptable circuit. □

Lemma 13 *Any circuit that is behaviorally equivalent to false is transformed into a wire connected to ground by algorithm 3.*

Proof A circuit is only behaviorally equivalent to false if there exists no assignment to the input ports that can assert the output of the circuit. Assume C is behaviorally equivalent to false and C' is the circuit as it stands after step 2 is initially run on C. Given the form of C', it follows that every or-gate input in C' must evaluate to false; particularly, each maximally conjunctive clause in C' must evaluate to false. A conjunctive clause can only evaluate to false if both the positive and negative of a single input i are conjunctions to that clause, otherwise the values on the ports in the clause would give rise to an asserting assignment for the clause; note that the duplicated input need not be the same for each conjunctive clause in C'.

Let v_0 be the head of the variable ordering. In step 3 we move all v_0-ports towards the output ports of each conjunctive clause. If both the positive and negative v_0-ports are contained in some clause L, then at some point during step 3, L must contain either subcircuit $v_0 \vee \neg v_0$ or subcircuit $v_0 \wedge (\neg v_0 \wedge A)$, either of which reduces to a wire to ground. All of clause L will be reduced to a wire to ground in step 4.

If clause L does not contain both the positive and negative v_0-ports, then at the end of step 3, L is either in form $v_0 \wedge C_a$ or form $(v_0 \wedge C_a) \vee (\neg v_0 \wedge C_a)$, where C_a contains both the positive and negative ports of some variable other than v_0. During step 5, the C_a from each L will be partitioned into two disjunctive clauses, each of which will be sent to the canonicalization algorithm in turn on the rest of the variable ordering. Since every variable in the circuit is in the ordering by assumption, it follows inductively that eventually each clause must reduce to a wire to ground. The iterations in step 7 insure that these ground wires will be propagated up through the circuit until the entire circuit is reduced to a wire to ground.

□

Lemma 14 *Combinational circuits C_a and C_b are behaviorally equivalent iff algorithm 3 transforms them to isomorphic circuits under the same variable ordering.*

Proof The backward direction follows since all of the transformations used are known to be behavior-preserving on boolean expressions. We will prove the forward direction by induction on the length of the variable ordering.
Base: Variable ordering has length 1. Then the expression can only contain variable v_0. Given that the four possible canonical circuits for a single variable are behaviorally distinct, the result follows from lemma 11.
Assume: Works for variable ordering of length k.
Show: Works for variable ordering of length $k+1$. After running the algorithm, we know we have circuits that are either wires to ground or of the forms given in lemma 12. If both circuits are wires to ground, they are isomorphic and we are done. If one circuit is a wire to ground and the other is in the more complex form, then the circuits must not have been behaviorally equivalent by lemma 13, which contradicts our initial assumption. Otherwise, we claim that S_1 is behaviorally equivalent to S_3 and S_2 is behaviorally equivalent to S_4. Assume S_1 and S_3 were not behaviorally equivalent. Then by definition there is some assignment to the input variables that can distinguish S_1 and S_3. Then the same assignment with v_0 given value false must be able to distinguish C_a and C_b, contradicting that C_a and C_b are behaviorally equivalent, so S_1 and S_3 must be behaviorally equivalent. The argument that S_2 and S_4 are behaviorally equivalent is analogous.

Since S_1 (S_2) and S_3 (S_4) are behaviorally equivalent, it follows by the inductive hypothesis that they reduce to the same canonical form. It follows that C_a and C_b are isomorphic after being canonicalized.

□

Lemma 15 *Let P be any path from the input to the output of a canonicalized combinational circuit. P must contain a wire to the port for each variable in the ordering used to canonicalize the circuit.*

Proof Let v be a variable from the ordering used in constructing canonicalized combinational circuit C_O and assume that there exists a path P from the input to the output of C_O that does not contain a v-port. When v is head of the variable ordering, by corollary 1 we see that v must be a port in P unless the portion of the circuit canonicalized after v reduced to a wire to ground. However, in this case, the propagation of the ground value in step 7 of algorithm 3 will continue

until the entire path is eliminated. Therefore, there can exist no path P in the final canonicalized circuit that does not contain a v-port. □

Lemma 16 *Let C_3 be the circuit obtained from C_2 by canonicalizing all maximal blocks of combinational logic. C_3 is behaviorally equivalent to C_2.*

Proof Since the only difference between C_3 and C_2 is in the combinational logic, it follows from lemma 14 that they are behaviorally equivalent.

□

Corollary 2 *C_3 is a one-hot machine.*

4.4 Removing Unreachable States

Given a circuit and its start state, a general algorithm for computing the set of reachable states can be stated as follows:

Algorithm 4

1. Initialize the set of reachable states to null and the set of states to be processed to the start state.
2. While the set to be processed is non-empty, remove an element from the set and add it to the set of reachable states. For each delay element d in the circuit, if d is reachable from the delay element for the state and d is not in the set of reachable states, add d to the set of states to be processed.

In order to apply this general algorithm, we need a method for determining reachability on circuit diagrams. One possibility would be to provide a semantic definition, such as the definition for computationally reachable given in section 3. We would prefer, however, to have a syntactic characterization of reachability so that we could reason about reachability without having to explicitly simulate the circuit diagram on sets of input values. The following lemma proposes one such possible characterization:

Lemma 17 *Let d_1 and d_2 be delay elements in some circuit diagram C that has had its boolean logic canonicalized. Delay element d_1 is computationally reachable from d_2 iff the output of d_2 appears as the only positive delay output port along some path to the input to d_1.*

Proof Assume d_2 appears as the only positive delay output port along some path P to the input to d_1. We need to show that d_1 is computationally reachable from d_2; this requires that we find an assignment a to the input ports of C such that when C is in d_2 and a is applied to the input ports, the state of C becomes d_1. Assignment a can be constructed from P where for all input ports p, $a(p) = 1$ if a positive port for p appears along P and $a(p) = 0$ otherwise. It follows that all ports along P will be true, thus making the input to d_1 true. Thus d_1 is computationally reachable from d_2 under a.

Assume d_1 is computationally reachable from d_2. Then there exists an input assignment a for C_2 such that if C_2 is in state d_2 and run on a, the new state of C_2 will be d_1. We must show that d_2 appears as the only positive delay output port along some path to the input to d_1. Since C_2 has been proven to be a one-hot machine, it follows that no path along which two delay elements are positive ports can ever be satisfied. Similarly, no path along which no delay elements are positive ports can ever be satisfied. It follows that only paths containing one positive delay output port can

ever be satisfied. We know that some path is satisfied by d_2 and a, so d_2 must be the only positive delay port along that path to the input of d_1.

□

This lemma motivates why we do not remove subcircuits that will always evaluate to true during our combinational logic canonicalization as we do subcircuits that always evaluate to false. Assume we had a circuit that transitioned to state d whenever input i was true, regardless of the current state of the circuit. The process of removing the always true circuitry could reduce the input to d to a wire from i. This would invalidate our syntactic reachability method because d would be reachable from any state, but not receiving state inputs diagrammatically. Our combinational logic canonicalization process avoids this problem by not removing such circuitry.

Let C_4 be the circuit diagram obtained from C_3 by removing all delay elements corresponding to states that are not in the set of reachable states and wiring the wires that were connected to the outputs of the removed delay elements to false.

Lemma 18 *C_4 is behaviorally equivalent to C_3.*

Proof We have only removed states that were computationally unreachable, affecting no other part of the circuit. It follows that C_4 must be behaviorally equivalent to C_3.

□

4.5 Machine Minimization

In order to minimize the state machine captured by our algorithm, we need to be able to identify computationally equivalent states. As in reachability analysis, we would like to have a syntactic characterization of computational equivalence, but we start by proposing a general algorithm for determining computational equivalence. This algorithm relies on the following definition:

Definition 2 Delay elements d_1 and d_2 are said to have the *same effect* on an output o in a one-hot circuit iff for all assignments i, o is asserted in d_1 under i iff o is asserted in d_2 under i.

Recall that since we are in a one-hot circuit, d_1 and d_2 can never be asserted simultaneously. We now propose the general algorithm:

Algorithm 5 Let C be a circuit diagram. Initialize a set S' to be empty; S' will keep track of which states are not computationally equivalent.

1. For each pair of delay elements d_1 and d_2, check to see if they have the same effect on every output in C. If there exists an output on which they do not have the same effect, put the pair (d_1, d_2) into set S'.

2. For each pair of delay elements d_1 and d_2 not in S', check all possible input assignments to see if a transition is ever made to a pair of states in S'. If so, add (d_1, d_2) to S'.

3. If S' changed, return to step 2 and repeat. If S' did not change, any pairs of states not in S' are behaviorally equivalent.

The following lemma defines our syntactic characterization of the same-effect property, which can be used in the above algorithm to syntactically check for computational equivalence.

Lemma 19 *Let C be a one-hot circuit, d_1 and d_2 be delay elements of C and let o be an output of C. Let C_a be the maximally combinational subcircuit of C that computes o. Let C_b be the circuit obtained from C_a by swapping the labels on d_1 and d_2 and recanonicalizing the circuit under the same variable ordering that was used to canonicalize C_a. d_1 and d_2 have the same effect on o iff C_a is isomorphic to C_b.*

Proof We have already established that the canonicalization algorithm reduces any two behaviorally equivalent circuits to isomorphic circuits when canonicalized under the same variable ordering. It follows that we need to prove that d_1 and d_2 have the same effect on o iff C_a and C_b are behaviorally equivalent.

If d_1 and d_2 have the same effect on C_a, then they must have the same effect on C_b. Any assignment i making o true under d_1 in C_a must therefore make o true under d_2 in C_b, which in turn must make o true under d_1 in C_b. Given that the switch of names is the only difference between C_a and C_b, it follows that C_a and C_b must be behaviorally equivalent. Similarly, if C_a and C_b are behaviorally equivalent, then d_1 and d_2 must have the same effect on C_a.

□

Let C_5 be the circuit diagram obtained from C_4 by unifying all sets of computationally equivalent states. To unify a set $d_1, \ldots, d_k$ of states, replace the output of each of $d_2, \ldots, d_k$ with the output of d_1 and replace the input to d_1 with a disjunction formed from the inputs of $d_1, \ldots, d_k$. Remove $d_2, \ldots, d_k$. C_5 is now a minimized machine by construction.

Lemma 20 *C_5 is behaviorally equivalent to C_4.*

Proof The only change from C_4 to C_5 is that any computationally equivalent states were unified into a single state. It follows by the definition of computationally equivalent that there cannot exist an assignment sequence on which different outputs are detected from running on C_4 versus C_5. □

Lemma 21 *Any two minimized behaviorally equivalent circuits are isomorphic with respect to state transitions.*

Proof Let C_a and C_b be minimized behaviorally equivalent circuits. They must contain the same number of states since they are minimized and behaviorally equivalent. Define a mapping ϕ between the states of C_a and the states of C_b inductively as follows: $\phi(s_1) = s_2$ where s_1 and s_2 are the start states of C_a and C_b, respectively. If $\phi(a_1) = a_2$ and there is an assignment i such that a_1 transitions to b_1 on i and a_2 transitions to b_2 on i, then $\phi(b_1) = b_2$. We need to show that ϕ is one-to-one and onto.

Unreachable states were removed earlier in the algorithm. It follows that all states in C_a and C_b are reachable. Since each state s in C_b is reachable, there exists an assignment sequence that takes C_b from the start state to s. That sequence can be used to help define ϕ, so ϕ must be onto.

Assume ϕ were not one-to-one. Then some state a_1 in C_a must map to two distinct states a_2 and b_2 in C_b. Since C_b is minimized, it follows that there exists an assignment sequence i that distinguishes a_2 and b_2. By the definition of ϕ, the output of a_1 under i must equal the output of each of a_2 and b_2 under i. Our machines are deterministic, so this is not possible unless C_b is not minimized, which is a contradiction. Therefore, ϕ is a bijection and C_a and C_b are isomorphic with respect to state transitions. □

Lemma 22 *C_5 is the unique minimization of C_4 with respect to state transitions.*

Proof This follows as a corollary to lemma 21. □

4.6 Constructing a Reachability Ordering

C_5 is now minimized with respect to state transitions and has no unreachable states. In preparation for our final combinational logic canonicalization step, we need to choose an ordering on interface input and delay element output ports. Given that our definition of behavioral equivalence requires that circuits being compared have the same input interface ports, the only real difficulty encountered in creating such an ordering that can be used to compare two circuits is in ordering the delay element output ports; we need to guarantee that delay elements isomorphic across the two circuits appear in the same positions in the ordering if the canonicalizer is going to behave properly.

Fortunately, since our circuits are minimized, we can use a reachability analysis to infer an ordering on the delay elements that will be consistent across two behaviorally equivalent circuit diagrams using the following algorithm:

Algorithm 6 Let d_0 be the start state of C_5. Initialize the ordering to contain only d_0. Initialize a workqueue to only contain d_0. Let $i_0, \ldots, i_{2^n-1}$ be the canonical ordering of the boolean combinations of the n input variables in C_5. While the workqueue is empty, iterate the following steps:

1. Let d be the first delay element in the workqueue. Append d to the end of the ordering
2. Let $D_0, \ldots, D_{2^n-1}$ be the delay elements to which d transitions under $i_0, \ldots, i_{2^n-1}$, respectively.
3. Remove from $D_0, \ldots, D_{2^n-1}$ any delay element that is already in the ordering or already in the workqueue.
4. Append what remains of $D_0, \ldots, D_{2^n-1}$ to the end of the workqueue.

Given a delay element and an assignment to the input variables, we can syntactically determine the delay element to which the circuit will transition by looking for a satisfiable input path along which the original delay element is the only positive delay output literal and the input variables have the values associated with the assignment; the delay element into which this path feeds is the delay element to which the circuit transitions under the assignment.

Lemma 23 *Given a current state and an input assignment, the delay element to which the circuit will transition is unique.*

Proof This follows from the fact that our circuits are well-formed, and hence deterministic. □

Lemma 24 *The delay output ordering produced by the above method is unique.*

Proof Given a circuit, the start state is fixed by assumption, as is the order in which we consider assignments to input interface ports. By lemma 24 then, the sequence of delay elements transitioned to from a given delay element d must be unique. It follows that the order in which delay elements are considered during the construction is unique, as is the final ordering resulting from the algorithm. □

4.7 Final Canonicalization

Let C_N be the circuit diagram obtained from C_5 by canonicalizing all maximal blocks of combinational logic with respect to the concatenation of the ordering produced in section 4.6 and an ordering on the input ports. We claim that C_N is a canonical form.

5 Proving the Form Canonical

Theorem 1 *Circuit diagrams C_a and C_b are behaviorally equivalent iff the algorithm canonicalizes them to isomorphic circuits.*

Proof Assume C_a and C_b are behaviorally equivalent circuit diagrams. We need to prove that they reduce to isomorphic circuits under algorithm 1 when the algorithm is run under the same variable ordering. After the minimization step, by lemma 21 we know that the circuits are isomorphic with respect to state transitions. It follows that we must prove that the circuitry feeding into each output and each mapped delay input is isomorphic. Since the state transition structures are isomorphic, the circuitry feeding into the respective delay element inputs must be behaviorally equivalent; by lemma 14 it follows that these subcircuits must be isomorphic. Similarly, since the machines are behaviorally equivalent, it follows that the circuitry producing the outputs must be behaviorally equivalent, and hence isomorphic after canonicalization. It follows that behaviorally equivalent circuits must canonicalize to isomorphic circuits.

Now assume C_a and C_b canonicalize to isomorphic circuits. We must prove that they are behaviorally equivalent. We have proven in lemmas 7, 16, 18, and 20 that each stage of the algorithm is behavior-preserving. Since behavioral equivalence is transitive, it follows that C_a and C_b must have been behaviorally equivalent.

□

6 Acknowledgements

The author thanks Jon Barwise, Gerry Allwein, Steve Johnson, and Larry Moss for reviewing the canonical form construction.

References

[Bry86] Randal E. Bryant. Graph based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C35(8):677–691, 1986.

[Fis94] Kathi Fisler. A logical formalization of hardware design diagrams. Technical Report TR416, Indiana University, September 1994.

[Gin62] Seymour Ginsburg. *An Introduction to Mathematical Machine Theory.* Addison-Wesley, 1962.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[Moo56] Edward F. Moore. Gedanken-experiments on sequential machines. In Claude Shannon and John McCarthy, editors, *Automata Studies*, pages 129–153. Princeton University Press, 1956.

[PW87] Franklin P. Prosser and David E. Winkel. *The Art of Digital Design.* Prentice-Hall, 2nd edition, 1987.

[Sha38] C. E. Shannon. A symbolic analysis of relay and switching circuits. *Transactions AIEE*, 57:713–723, 1938.

[SP94] V. Singhal and C. Pixley. The verification problem for safe replaceability. In D. Dill, editor, *Proc. 6th International Conference on Computer Aided Verification, Springer-Verlag Lecture Notes in Computer Science vol. 818*, pages 311–323, June 1994.