Modeling Hierarchical Combinational Circuits

Jerry R. Burch David Dill Elizabeth Wolf Giovanni De Micheli
Stanford University

Abstract

Hierarchical descriptions of combinational circuits often contain *apparent loops* [1, 3]. Since it may be difficult to distinguish apparent loops from actual loops, it is useful to construct models of combinational circuits that can handle cyclic dependencies. We show that Boolean relations are inadequate for this purpose, and define a ternary model that solves the problem. We use the model to characterize exact solutions to a broad class of substitution and rectification problems. The theory cleanly handles network transformations that might introduce cyclic dependencies.

1 Introduction

The first and most important step in the development of analysis or synthesis methods for any kind of system is the construction of a precisely-defined model. A model should accurately reflect reality (possibly under constraints on the domain of applicability), while providing a "mathematically clean" theory.

In this paper we develop a model for the behavior of combinational circuits. Combinational circuits have been studied so thoroughly for such a long time that it is surprising that anything new can be said. However, there continue to be innovations in the area, particularly related to don't cares [2, 4, 5, 6].

This paper extends previous work on *Boolean relations* for modeling combinational circuits. Boolean relations have the advantage as a circuit model of allowing many implementation choices; while any particular implementation maps each input combination to a unique output combination, the relation can specify several different output combinations. The use of Boolean relations in synthesis allows the expression of various kinds of *don't cares* [5]. In formal verification, the use of Boolean relations for specifications makes it possible to write specifications which capture the *minimum* requirements of a circuit, instead of requiring that we overspecify by including irrelevant details.

Boolean relations were intended to model circuits described using cascade composition. Such networks are guaranteed to form an acyclic directed graph; that is, there are no loops or cycles in the circuit. At the same time, the process of formal verification is facilitated by, and may even require, the representation of both implementation circuitry and specifications by models whose structure reflects the intended *functional* modularity of the design. Unfortunately, such hierarchical representations of circuits often do not allow the circuit's structure to be naturally described using cascade composition.

This work was supported by the NSF under grant MIP-8858807. The third author was also supported by a Xerox Fellowship.

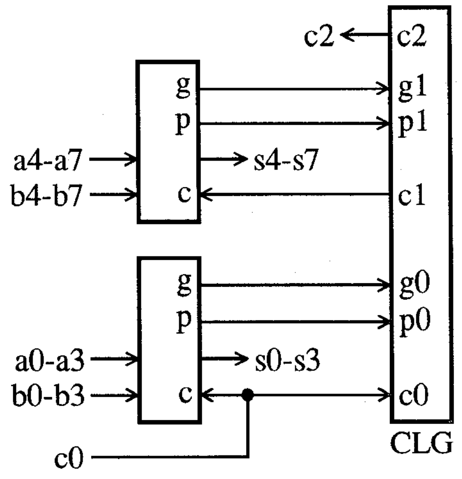

Figure 1: Eight-bit carry look-ahead adder

The carry look-ahead adder in figure 1 is an example of this phenomenon. The bidirectional communication between the adders and the carry look-ahead generator (CLG) is an example of an "apparent loop" [3] (also called a "pseudo-cycle" [1]). There are no loops or cycles in the circuit when it is described at the gate level, but this information is lost in the block diagram of the circuit. In this case, an acyclic block diagram could be formed by splitting the CLG into two smaller blocks [3]. In general, however, requiring logic blocks to be split in this way negates many of the advantages of hierarchical descriptions. Thus, as hierarchical descriptions become more common in design tools, and as formal verification becomes of greater relevance to the design process, we believe that it is increasingly important that modeling techniques allow for apparent loops in combinational circuits.

Providing semantics for circuits with apparent loops is no more difficult than providing semantics for circuits without loops. Unfortunately, there appears to be no way to distinguish between apparent loops and actual loops, short of examining the internal structure of logic blocks (which would defeat the purpose of a hierarchical description). Because we intend to use our models for formal verification, we represent specifications and implementation circuitry using the same kind of models. However, the standard syntactic methods for keeping track of actual dependencies between the inputs and outputs of a circuit (which allow the detection of actual loops) do not work for non-deterministic specifications.

This is illustrated by the following example. Consider the Boolean relation with one input and two outputs, whose outputs take the same Boolean value (both are 0 or both are 1) if the input has value 0, and whose outputs take dis-

1063-6757/93 $03.00 © 1993 IEEE

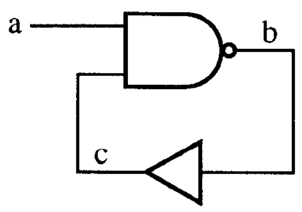

Figure 2: Gated ring oscillator.

tinct values if the input has value 1. This relation is a nondeterministic specification; every Boolean function consistent with it is a possible implementation. Each such implementation has exactly one constant output, while the other ouput's value depends on the input value. However, *which* output is dependent on the input differs with the choice of implementation. Assume we would like to wire this implementation circuit together with an inverting buffer whose input is wired to one of the outputs of the implementation circuit and whose output provides that circuit's input value. Depending on the particular implementation and the choice of output wire, the composition of the two circuits will form either an actual loop or an apparent loop: in the absence of further information concerning this implementation we cannot tell whether a combinational cycle (an actual loop) will be formed or not. Therefore, in order to preclude all *potential* actual combinational cycles we must maximize the dependency set associated with the Boolean relation, thus disallowing composition with even the innocuous inverter.

In this case, use of the set-of-support method in conjunction with a nondeterministic specification clearly leads to undue conservatism, erroneously "detecting" an actual loop where there is only an apparent one. All other methods for constructing the dependency set of a non-deterministic specification seem to fail also. Thus, we want to provide semantics for actual loops as well as apparent loops.

The circuit in figure 2 is a simple example of why Boolean relations are not an adequate model for circuits with actual loops. Using standard methods [5] to construct the Boolean relation $R(a, b, c)$ for this circuit gives

$$(a = 0) \wedge (b = 1) \wedge (c = 1).$$

Notice that it is not possible (according to this relation) for a to take the value 1. This is clearly incorrect since a is an input; its value cannot be controlled by the circuit. The problem is that if a does become 1, then nodes b and c oscillate. Boolean relations cannot represent this oscillation, so the relation does not even represent the possibility that a could be 1.

We address this limitation of Boolean relations by using a ternary model where $\mathcal{T} = \{0, 1, \perp\}$ is the set of possible values for a node of a network. Intuitively, the 0 and 1 values mean that the node eventually settles to the corresponding voltage. The value $\perp$ indicates that the node oscillates or has an intermediate voltage rather than settling to a Boolean value. This is slightly different from the normal interpretations of the value X in ternary models. In particular, $\perp$ does not mean that the value of the node is "unknown"; we use relations that represent unknown Boolean values by explicitly allowing both 0 and 1 as the value of a node.

Recalling the circuit in figure 2, if the value of node a is 1 then nodes b and c oscillate, so b and c are given the value $\perp$. Thus, the oscillating behavior of the circuit can be accurately represented in the ternary model, unlike the Boolean relation model. In section 2, we show how this results from the definition of composition and the models of the individual gates in the circuit.

In section 3, we describe how the ternary model can be used to characterize all of the ways a subnetwork can be modified without changing the input/output behavior of the full network. In addition to being applicable to circuits with loops, the method unifies and extends techniques based on satisfiability don't cares and observability don't cares. It also extends design rectification techniques, both those based on Boolean relations [10] and the method of Boolean unification [9].

2 Ternary I/O-relations

Our models for combinational circuits differ from Boolean relations in two important respects. The first difference is that we use a ternary domain $\mathcal{T} = \{0, 1, \perp\}$ as the set of possible values for a node of a network. Intuitively, the 0 and 1 values mean that the node eventually settles to the corresponding voltage. The value $\perp$ indicates that the node oscillates rather than settling to a Boolean value. The second difference is that we explicitly associate with each relation the set of inputs and outputs of the circuit or subcircuit being represented by that relation. Having the inputs and outputs be implicit, as is normally done with Boolean relations [5], can become confusing as the hierarchical structure of circuits becomes more complicated.

Using the ternary domain described above, we model combinational circuits with *ternary I/O-relations.* A ternary I/O relation is an ordered triple $T = (I, O, R)$. The sets I and O give the names of the inputs and outputs of the circuit. The *alphabet* of T is $A = I \cup O$. The set R gives the possible input/output combinations of the circuit: R is a subset of $\mathcal{T}^{I \cup O}$, which is the set of all functions from $(I \cup O)$ to $\mathcal{T}$. (This is analogous to the set of Boolean functions over this alphabet, except that each function assigns values from the *ternary* domain $\mathcal{T}$). In other words, if $s \in R$ and $a \in (I \cup O)$, then s represents a possible input/output combination of the circuit, and $s(a)$ is the value of the node a in that particular input/output combination. Given an ordering on the elements of $I \cup O$, there is a clear isomorphism between functions in $\mathcal{T}^{I \cup O}$ and ternary vectors in $\mathcal{T}^{|I \cup O|}$ or pairs of vectors in $\mathcal{T}^{|I|} \times \mathcal{T}^{|O|}$. Rather than use vectors, however, we prefer to explicitly associate the name of a node with its value.

2.1 Operations on I/O-relations

The most important operation on I/O-relations is composition, which represents the result of connecting two circuit components together. Let $T = (I, O, R)$ and $T' = (I', O', R')$ be I/O-relations. Assume that $O \cap O' = \emptyset$, which means that T and T' have disjoint sets of outputs. Let $A = I \cup O$ and $A' = I' \cup O'$. The composition $T'' = T \parallel T'$ is defined as follows. A node is an output of T'' if and only if it is an output of either T or T'; so the set of outputs of

T'' is $O'' = O \cup O'$. A node is an input of T'' if and only if it is not an output of T'' and is an input of T or T'; so the set of inputs of T'' is $I'' = (I \cup I') - O''$. Normally, composing two Boolean relations simply requires taking their intersection. With I/O-relations, the basic idea is the same, but we must deal with how our representation of an input/output combination is affected by the set of inputs and outputs. Thus, the third component of T'' is

$$R'' = \{s'' \in \mathcal{T}^{I'' \cup O''} : \exists s \in R[\forall a \in A[s''(a) = s(a)]] \wedge \exists s' \in R'[\forall a' \in A'[s''(a') = s'(a')]]\}.$$

If we describe a circuit by simply composing together the I/O-relations of the circuit's components, there is no way to distinguish the primary outputs of the circuit from its internal nodes; both types of nodes are included in the output set of the resulting I/O-relation. We often want to ignore the internal signals of the circuit and leave only the inputs and primary outputs. The *projection* operator is used for this purpose. If $T = (I, O, R)$ represents a circuit and $B \subseteq I \cup O$ is the set of inputs and intended primary outputs of the circuit (which implies that $I \subseteq B$), then

$$proj(B)(T) = (I, O \cap B, \{s' \in \mathcal{T}^B : \exists s \in R[\forall b \in B[s'(b) = s(b)]]\}).$$

The I/O-relation $proj(B)(T)$ is analogous to the Boolean relation formed by using the *smoothing* operator to remove nodes in $(I \cup O) - B$.

There is also a renaming operation on I/O-relations for changing the names of inputs and outputs. We will not give the formal definition here, however, since renaming is not needed in any of the examples in this paper.

2.2 Implementations and specifications

Let $T = (I, O, R)$ be an I/O-relation constructed by composing the I/O-relations of the components of some circuit and by applying the projection operator, as appropriate. Thus, T describes the possible input/output combinations of the design. Let T' be an I/O-relation representing a specification for the circuit. That is, T' represents the set of correct or acceptable input/output combinations for the circuit, which may be quite different from the set of actual combinations. For T' to be a specification for T, it must have the same inputs and outputs as T. Therefore, T' is of the form $T' = (I, O, R')$. We say that T *satisfies* T', written $T \subseteq T'$, if and only if $R \subseteq R'$.

2.3 Examples

We write $s = \langle a \rightarrow 0,\ b \rightarrow 1 \rangle$ to denote a function s that maps a to 0 and maps b to 1. Using this notation, the ternary I/O-relation T for an inverter with input set $I = \{a\}$ and output set $O = \{b\}$ is written

$$T = (\{a\},\ \{b\},\ \{\langle a \rightarrow 0, b \rightarrow 1\rangle,\ \langle a \rightarrow 1, b \rightarrow 0\rangle,\ \langle a \rightarrow \perp, b \rightarrow \perp\rangle\}).$$

We often describe a relation with a propositional formula, rather than explicitly listing the input/output combinations

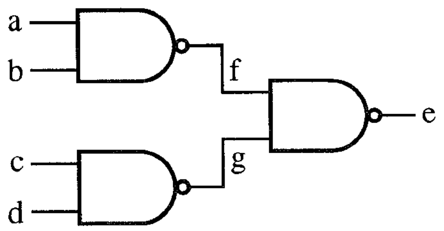

Figure 3: Possible implementation of an example non-deterministic specification.

that are its elements. For example, we might write the ternary I/O-relation for an and-gate as

$$T = (\{a, b\}, \{c\}, c = ab),$$

where conjunction is extended in the normal way for ternary values. Even though the interpretation of $\perp$ is different than that of X in ternary simulation, Boolean functions are extended to ternary functions in the same way. For example, the and-gate above contains $\langle a \rightarrow 0,\ b \rightarrow \perp,\ c \rightarrow 0\rangle$ and $\langle a \rightarrow 1,\ b \rightarrow \perp,\ c \rightarrow \perp\rangle$. We assume all Boolean operators are extended to the ternary domain in analogous fashion. We use "=" to denote equality, and "$\wedge$" and "$\vee$" as logical connectives in propositional formulas.

In addition to the I/O-combinations in the standard ternary extension of an and-gate, one might include I/O-combinations such as $\langle a \rightarrow \perp,\ b \rightarrow \perp,\ c \rightarrow 0\rangle$. This would represent (for example) the case where a and b are non-overlapping clocks that oscillate indefinitely while c remains stable at 0. We do not explicitly include this case in the ternary I/O-relation for an and-gate. By convention, if an input/output combination s is included in an I/O-relation, then any combination s' formed from s by changing a $\perp$ output to 0 or 1 is understood to be implicitly included. Thus, because $\langle a \rightarrow \perp,\ b \rightarrow \perp,\ c \rightarrow \perp\rangle$ is already included in the I/O-relation of an and-gate, $\langle a \rightarrow \perp,\ b \rightarrow \perp,\ c \rightarrow 0\rangle$ is not explicitly included.

For a slightly more complicated example, consider the specification

$$T = (\{a, b, c, d\},\ \{e\},\ (e = (ab + cd)) \vee (1 = \overline{a}\overline{b}\overline{c}\overline{d}e)).$$

It allows the output wire e to stabilize to either 0 or 1 if the input wires all stabilize to 0. However, an implementation satisfying the specification T is allowed to be deterministic, and settle to a specific predetermined value in this case. An implementation $T' = (I, O, R')$ which satisfies T may be identical to T or it may have R' given by

- $e = (ab + cd)$, or
- $e = (ab + cd + \overline{a}\overline{b}\overline{c}\overline{d})$.

For example, if T'' is the I/O-relation representing the circuit illustrated in figure 3 (which has inputs $\{a, b, c, d\}$ and outputs $\{e, f, g\}$) then $proj(\{a, b, c, d, e\})(T'') \subseteq T$.

Recall the example in figure 2 used to illustrate the limitations of Boolean relations. Let $T = (T_1 \parallel T_2)$, where $T_1 = (\{a, c\}, \{b\},\ b = \overline{ac})$ represents a nand-gate, and $T_2 = (\{b\}, \{c\},\ b = c)$ represents a non-inverting buffer. The rules for composition yield

$$T = (\{a\}, \{b, c\},\ (a = 0 \wedge b = 1 \wedge c = 1) \vee (a \neq 0 \wedge b = \perp \wedge c = \perp)).$$

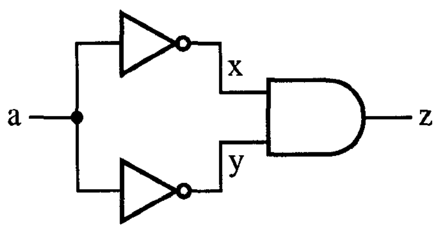

Figure 4: Substitution example.

Thus, the ternary model accurately shows that nodes b and c will oscillate when $a = 1$.

3 Correct substitutions

Most optimization techniques for combinational circuits involve repeatedly replacing a small subnetwork of the circuit with another that is less expensive by some measure such as area or delay. It is essential that this transformation not change the external behavior of the circuit. However, the replacement subnetwork need not be equivalent to the original; properties of the rest of the circuit may allow for changing the behavior of the subnetwork without changing the external behavior of the full circuit. Satisfiability don't cares (SDC-sets) and observability don't cares (ODC-sets) are often used to characterize legal substitutions [2, 4, 6]. Boolean relations have been shown to be more expressive [5].

In this section we describe another method for characterizing legal optimizations. Unlike previous work, this method is applicable for non-cascade composition and for the ternary I/O-relation model described above. As a semantic characterization of correct substitutions, it subsumes satisfiability don't cares, observability don't cares and Boolean relations. It is more general than Boolean unification [9] because of its applicability to circuits with loops.

The method depends on the concept of *mirroring*, which was originally defined for trace structures [7, 8]. Mirroring an I/O-relation involves swapping its inputs and outputs, and complementing its relation. That is, if $T = (I, O, R)$ is a ternary I/O-relation, then the mirror of T is

$$mir(T) = (O, I, T^{I \cup O} - R).$$

We say the I/O-relation T is *empty* if $R = \emptyset$. The following theorem is easy to prove.

Theorem 1 *Let T and T' be I/O-relations with the same inputs and outputs. Then $T' \subseteq T$ if and only if $T' \parallel mir(T)$ is empty.*

Intuitively, $mir(T)$ is the maximal environment of T such that $T \parallel mir(T)$ is empty.

3.1 Substitution examples

Consider the circuit in figure 4. Let T_1 and T_2 be the ternary I/O-relations representing the inverters driving x and y, respectively, and let T_3 be the I/O-relation representing the and-gate.

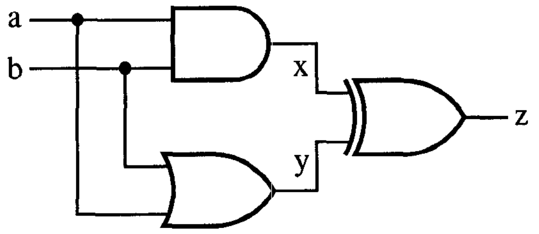

Figure 5: Substitution and rectification example.

We wish to determine the possible replacements for the and-gate in figure 4, which corresponds to finding the I/O-relations T_3' such that

$$(T_1 \parallel T_2 \parallel T_3') \subseteq (T_1 \parallel T_2 \parallel T_3).$$

This example is simple enough that one can tell by inspection that T_3' is a legal substitution if and only if

$$T_3' \subseteq (\{x, y\},\ \{z\},\ (x \neq y) \vee (z = x)).$$

This allows many different functions to replace the and-gate: $z = x$, $z = y$, $z = x + y$, etc.

The same characterization of the the legal substitutions can be derived formally as follows. Let $T_{12} = T_1 \parallel T_2$ and $T = T_{12} \parallel T_3$. Also, let $A_3' = \{x, y, z\}$ be the alphabet of T_3' (recall that the *alphabet* of an I/O-relation is the union of its input set and its output set). We can use theorem 1 to characterize the set of all legal T_3' as follows:

$$\begin{aligned}
&(T_{12} \parallel T_3') \subseteq T \\
&\iff T_{12} \parallel T_3' \parallel mir(T) \text{ is empty} \\
&\quad T' \text{ is empty iff } proj(B)(T') \text{ is empty, for all } T',\ B \\
&\iff proj(A_3')\big(T_{12} \parallel T_3' \parallel mir(T)\big) \text{ is empty} \\
&\quad \text{property of projection and composition} \\
&\iff T_3' \parallel proj(A_3')\big(T_{12} \parallel mir(T)\big) \text{ is empty} \\
&\quad \text{theorem 1} \\
&\iff T_3' \subseteq mir\big(proj(A_3')\big(T_{12} \parallel mir(T)\big)\big) \\
&\quad \text{substitution and simplification} \\
&\iff T_3' \subseteq (\{x, y\},\ \{z\},\ (x \neq y) \vee (z = x)).
\end{aligned}$$

In computing the final result, one may note that when ternary I/O-relations are described with logical formulas, the mirror operation is analogous to logical negation, composition is analogous to conjunction, and projection is analogous to existential quantification.

This relation could have been derived using only satisfiability don't cares. Unlike satisfiability don't cares, however, our method is not restricted to networks described using cascade composition. In addition, our method can handle potential optimizations that would normally require the use of observability don't cares. Consider the circuit in figure 5. Let T_1, T_2 and T_3 be the I/O-relations representing the and-gate, the or-gate and the exclusive-or gate, respectively. Let $T_{12} = T_1 \parallel T_2$ and $T = T_{12} \parallel T_3$.

For our first analysis of this circuit, consider the possible candidates for simultaneously replacing the and-gate and the or-gate. That is, characterize the I/O-relations T_{12}' such that

$$(T_{12}' \parallel T_3) \subseteq T.$$

Let $A'_{12} = \{a, b, x, y\}$ be the alphabet of T'_{12}. Using reasoning similar to that used for the previous circuit, we find that

$$\begin{aligned}&(T'_{12} \parallel T_3) \subseteq T\\ &\iff T'_{12} \subseteq mir\big(proj(A'_{12})(T_3 \parallel mir(T))\big)\\ &\iff T'_{12} \subseteq (\{a,b\}, \{x,y\},\ (x = ab) \wedge (y = a+b)).\end{aligned}$$

This result offers essentially no opportunity for optimizing the and-gate and the or-gate, even though intuitively there should be many possible optimizations. The problem is in our original formulation of the requirement for legal values of T'_{12}:

$$(T'_{12} \parallel T_3) \subseteq T.$$

This formulation requires that the x and y outputs of T'_{12} be the same as the x and y outputs of the original circuit T. However, such a requirement conflicts with our intuition that x and y can be ignored except for their effect on z. This intuition is made explicit in the following reformulation of the requirements on T'_{12}:

$$proj(\{a,b,z\})(T'_{12} \parallel T_3) \subseteq proj(\{a,b,z\})(T).$$

By projecting out x and y, the above inequality makes it explicit that only the relationship between a, b and z is significant. It can be shown that

$$\begin{aligned}&proj(\{a,b,z\})(T'_{12} \parallel T_3) \subseteq proj(\{a,b,z\})(T)\\ &\iff T'_{12} \subseteq mir\big(proj(A'_{12})\big(T_3 \parallel mir(proj(\{a,b,z\})(T))\big)\big)\\ &\iff T'_{12} \subseteq (\{a,b\}, \{x,y\},\ (x \oplus y) = (a \oplus b)).\end{aligned}$$

The new characterization of legal values for T'_{12} allows all of the intuitively correct optimizations for the nodes x and y, including, for example, $x = a$ and $y = b$.

The above two analyses of the circuit in figure 5 show the important role that the projection operation plays in making explicit the primary inputs and outputs of a circuit. Using projection to change the set of circuit nodes that are considered primary outputs can change the set of legal substitutions of a subnetwork.

3.2 General theorem

All of the analyses of circuits in this section are specific applications of a general theorem concerning legal substitutions. Its proof will appear in the full version of this paper.

Theorem 2 *Let T'_1, T_2 and T be I/O-relations with alphabets A'_1, A_2 and A, respectively. Assume that T'_1 and T_2 do not share any outputs (so they can be composed), and that $A \subseteq A'_1 \cup A_2$. Also assume that $A'_1 \subseteq A \cup A_2$. Then*

$$proj(A)(T'_1 \parallel T_2) \subseteq T$$

if and only if

$$T'_1 \subseteq mir(proj(A'_1)(T_2 \parallel mir(T))).$$

In the above theorem, T is a specification of the intended input/output behavior of the full circuit; A contains only the primary inputs and primary outputs. A replacement subnetwork of the circuit is represented by T'_1. The remainder of the circuit is represented by T_2. The assumption that $A \subseteq A'_1 \cup A_2$ means that the composition of T'_1 and T_2 drives every output, and uses every input, of T. The assumption that $A'_1 \subseteq A \cup A_2$ means that any wires that are local to the new subnetwork (that is, not used by T_2 or T) have already been projected out of the I/O-relation T'_1. Even with these assumptions, there is potentially a wide range of choices for A'_1. We will see an example of this below.

3.3 Rectification examples

In the examples so far, we considered how a subnetwork might be changed without affecting the behavior of the full network. It is also possible to change the specification of the circuit, and then determine whether a subnetwork can be changed so that the new specification is satisfied by the full circuit. This is a very general form of the rectification problem [10].

For example, let T_1, T_2, T_3 and T_{12} be as in our earlier analysis of the circuit in figure 5. Rather than have this circuit compute $z = a \oplus b$, suppose we change the specification T to require that $z = ab$. Let $A = \{a, b, z\}$ be the alphabet of T. Consider the problem of finding those I/O-relations T'_3 such that

$$proj(A)(T_{12} \parallel T'_3) \subseteq T.$$

Let $A'_3 = \{x, y, z\}$ be the alphabet of T'_3. Using theorem 2, it can be shown that

$$T'_3 \subseteq (\{x,y\}, \{z\},\ z = x \vee (x = 1 \wedge y \neq 1) \vee (x \neq 0 \wedge y = 0)).$$

This allows the xor-gate in figure 5 to be replaced by $z = x$ or $z = xy$.

Suppose we change the specification T to require $z = a\bar{b}$, and we keep $A'_3 = \{x, y, z\}$ as the alphabet of T'_3. In this case, we must have

$$\begin{aligned}T'_3 \subseteq (\{x,y\}, \{z\},\ &(x = y \wedge z = x) \vee (x = 1 \wedge y \neq 1)\\ &\vee (x \neq 0 \wedge y = 0)).\end{aligned}$$

Notice that T'_3 cannot contain any input/output combination with $x = 0$ and $y = 1$. Since x and y are both inputs, this means there is no implementation of T'_3 that satisfies the above constraints: there is no *total* ternary function consistent with this relation. This is consistent with the fact that there is no way to change the xor-gate in figure 5 to a subcircuit with only x and y as inputs so that $z = a\bar{b}$. Our method for finding possible legal substitutions for a subnetwork also shows us when there are none.

If we expand A'_3 (the alphabet of T'_3) to include a or b (or both), then theorem 2 can be used to show that in this case there do exist legal implementations of T'_3. This is an example of how changing the inputs of a proposed replacement subnetwork can change the set of legal substitutions, even possibly from an empty set to a non-empty one.

These methods can also be applied to the mixed mode rectification problem considered by Watanabe and Brayton [10]. Let W, X, Y and Z be sets of wires, as in fig-

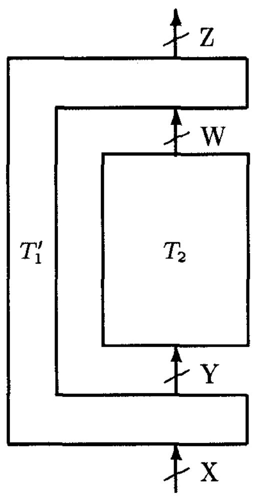

Figure 6: Mixed Mode Rectification

ure 6. Let $T = (X, Z, R)$ be a specification of desired input/output behavior for the circuit, and let $T_2 = (Y, W, R_2)$ be an existing implementation. We wish to find $T_1' = ((X \cup W), (Y \cup Z), R_1')$ such that

$$proj(X \cup Z)(T_1' \parallel T_2) \subseteq T.$$

By theorem 2, this is true if and only if

$$\begin{aligned} T_1' &\subseteq mir(proj(W \cup X \cup Y \cup Z)(T_2 \parallel mir(T))) \\ &= mir(T_2 \parallel mir(T)). \end{aligned}$$

This characterization of the possible implementations of T_1' allows more flexibility than the characterization given by Watanabe and Brayton [10]. Since their model could not completely represent the ramifications of loops introduced by having Y depend on W in T_1', they had to give a more conservative characterization to guarantee correctness. In fact, it can be shown that for any mixed-mode rectification problem, and for any rectifying circuit T_1' that is allowed by Watanabe and Brayton, there exists a circuit that is no larger and no slower than T_1' that directly implements the specification T without making any use of the original circuit T_2. Because our model can handle loops, we can give an exact characterization of the legal implementations of T_1', allowing the full potential efficiency of rectification.

4 Conclusions

This work generalizes models of combinational circuits to support non-cascade composition of modules, which is crucial for hierarchical reasoning about circuits. We hope that this foundation can be applied to a variety of problems. In synthesis, it provides a theory of combinational don't cares that works for non-cascade composition. Also, it gives a basis for defining and perhaps proving the soundness of synthesis and optimization techniques: starting with a given specification, the result of synthesis or optimization should conform to the specification in the precise sense we have defined.

This model should also be useful for verification. The use of hierarchical and compositional methods is crucial for containing the computational complexity of verification methods, and allowing non-cascade composition greatly expands the ways that hierarchical methods can be used.

This research is intended to be a major step towards finding a hierarchical, relational model of the sequential behavior of synchronous machines, such as Mealy machines. One of the central problems in modeling non-cascade compositions of Mealy machines is dealing with apparent combinational loops, which can occur because the output of a Mealy machine may respond to an input through a combinational path. The results described here solve that core problem, so the extension to the sequential case should be straightforward.

Another direction for further investigation would be to explore some concepts of trace theory [7] that are missing from the theory here: *failures* (inputs that the environment of the circuit is not allowed to generate), *conformation equivalence* and *canonical forms*. It remains to be seen whether the benefits of such a theory outweigh the additional complexity it would entail.

References

[1] A. Appel. Simulating digital circuits with one bit per wire. *IEEE Trans. CAD*, 7(9):987–993, Sept. 1988.

[2] K. A. Bartlett, R. K. Brayton, G. D. Hachtel, R. M. Jacoby, C. R. Morrison, R. L. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang. Multilevel logic minimization using implicit don't cares. *IEEE Trans. CAD*, 7(6):723–740, June 1988.

[3] J. F. Beetem. Hierarchical topological sorting of apparent loops via partitioning. *IEEE Trans. CAD*, 11(5):607–619, May 1992.

[4] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli. Multilevel logic synthesis. *Proc. IEEE*, 78(2):264–300, Feb. 1990.

[5] R. K. Brayton and F. Somenzi. Boolean relations and the incomplete specification of logic networks. In *Proc. Int. Conf. VLSI*, pages 231–240, Aug. 1989.

[6] M. Damiani and G. D. Micheli. Observability don't care sets and Boolean relations. In *Proc. Int. Conf. CAD*, pages 502–505, 1990.

[7] D. L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits.* ACM Distinguished Dissertations. MIT Press, 1989.

[8] J. C. Ebergen. A technique to design delay-insensitive VLSI circuits. Report CS-R8622, CWI, The Netherlands, June 1986.

[9] M. Fujita, Y. Tamiya, Y. Kukimoto, and K. Chen. Application of boolean unification to combinational logic synthesis. In *Proc. Int. Conf. CAD*, pages 510–513, 1991.

[10] Y. Watanabe and R. K. Brayton. Incremental synthesis for engineering changes. In *Proc. Int. Conf. Computer Design*, pages 40–43, 1991.