# Analysis of Digital Circuits Through Symbolic Reduction

R. P. Kurshan
AT&T Bell Laboratories
Murray Hill, NJ 07974

K. L. McMillan
Carnegie Mellon University
Pittsburgh, PA 15213

February 18, 1997

**Abstract**

A well-known difficulty associated with formal verification of digital circuits is to define a formal model which both accurately reflects the behavior of the circuit, and is analytically tractable. Analog models, such as those used in circuit-level simulation, can quite accurately predict the electrical behavior of transistor circuits for each given input waveform and given values of circuit parameters. However, formal analysis of more general circuit behavior over continuous ranges of input waveforms and circuit parameters has not been thought to be possible due to the infinite number of dynamical behaviors such an analysis would need to handle. On the other hand, switch level models, which may be analytically tractable, are not sufficiently general to model all aspects of switching circuit behavior, and have no formal correspondence with measurable quantities in the circuit.

This paper describes a semi-algorithmic method to extract finite-state models from an analog, circuit-level model by means of homomorphic (behavior-preserving) transformations. Properties to be verified are defined by $\omega$-automata. Efficient algorithms for testing language containment of automata then can be applied to verify properties of the finite-state models. Proof of the property in the finite-state model guarantees the property in the analog circuit-level model over a continuous range of input waveforms and circuit parameters. While in practice this method applies directly only to smaller circuit components, it can be used to analyze larger circuits as well by deriving a hierarchy of increasingly abstract models, through repeated applications of homomorphic transformations. Examples of extraction, homomorphism and verification are described in the paper.

# 1 Introduction

The high cost of integrated circuit fabrication requires the designer to have a high degree of confidence in the correctness of a design before committing it to hardware. The customary method for validating a circuit design is to test a model of the circuit through computer simulation, running the model under a finite set of operating conditions which is hoped to be sufficiently general to expose any flaws in the design. It is now widely accepted that current simulation techniques are not by themselves adequate to ensure the correctness of complex designs. For this reason, alternatives to simulation, collectively known as "formal verification", have been the focus of considerable interest in the last dozen years. A formal verification method provides a formalism for modeling a system, for expressing specifications of the system, and for establishing whether the model meets the specifications.

Traditionally, there have been two schools of digital circuit analysis, both based upon simulation. Those who are concerned with detailed behavior have used analog models founded in classical network analysis, while those who consider the large-scale behavior of digital circuits have used models based upon switching theory. Analog models [Nag75] generally are used where very precise information about timing is required, or where analog behavior is crucial to the circuit's operation, as in the sense amplifiers of a dynamic memory circuit. Switching models include *transistor switch-level models* [Bry80] used for MOS circuits and *gate-level models* [Rue81]. These models establish levels of abstraction sufficient to render tractable the behavioral simulation of a circuit at a more global level. While switching models generally are viewed as abstractions of the underlying analog model, this is not strictly the case, and indeed, the relationship between a switching model and an analog model is rarely expressed precisely. In fact, as we will see, switching models break down entirely under certain conditions, including those where "hazards" or "metastability" can occur.

A problem common to simulation using either kind of model is that simulation cannot cover the behavior of the model over a continuous range of conditions and input waveforms. One must hope that enough behaviors can be simulated to expose the "bugs" in the design. Even then, without a formal specification, there is no precise definition of what constitutes a "bug". Often this decision is left to the intuition of the designer. Variations in circuit parameters resulting from manufacturing tolerances, temperature changes, or simply inaccuracies in the model may result in behaviors which are not anticipated and thus not covered in simulation. Simulation methods not only fail to test a circuit model completely, but often lack even well-defined criteria to determine if a given simulation run exposes incorrect behavior. Formal verification attempts to solve both of these problems by providing a precise definition of correct behavior, and a decision procedure for determining if all behaviors of the circuit model are correct.

While much recent work in this direction is based upon new models and methodologies, the roots of formal analysis of digital circuits can be found in classical automata theory, namely Shannon's seminal 1938 AIEE paper "A Symbolic Analysis of Relay and Switching Circuits", the 1943 work of McCulloch and Pitts on nerve nets, von Neumann's work on digital computers, Huffman's "The Synthesis of Sequential Switching Circuits" (1954) and Mealy's 1955 BSTJ paper "A Method For Synthesizing Sequential Circuits". More recently one finds formal verification techniques for digital circuits using context-free grammars[Fos81], Floyd-Hoare logic[Sho83] and temporal logic[MO81], [Boc82], [FST84], [DC85], [BCDM85]. The analytical methods behind such formal verification techniques typically are based upon either deductive theorem-proving or state-space exploration. Deductive theorem-proving has been extensively studied, but remains somewhat controversial with regard to its general utility. Methods based upon state-space exploration are limited by the size of the model's state-space, which grows exponentially with the number of circuit components. This fact has limited the applicability of state-space search to small circuits[DC85].

In all of these formal methods, there is the vulnerability to over-simplification and misrepresentation already mentioned; the underlying models of circuit behavior are only heuristic approximations and are subject to error. In some cases these models are gross misrepresentations of the real circuit's behavior. In any case, without a formal relationship between the model chosen for formal verification and an analog circuit model, one cannot assert that a proof about the former relates to a claim about the latter.

In our view, three factors are basic to the success of any formal verification method for circuits. The first is the accuracy with which the circuit model represents the behavior of the actual circuit. The second is the tractability of the analysis procedure on realistically large circuits. Last is the generality of the analysis: a very accurate model may lack sufficient generality to model variations in input waveforms and variations in circuit parameters resulting from manufacturing tolerances, temperature variations, and changes in process and layout. It is of considerable interest to be able to verify a circuit "generically", for a range of inputs and circuit parameters, and furthermore, to state this range in terms that allow layout and process engineers to ensure that a circuit implementation stays within it.

The purpose of this paper is to show how the most accurate available models of circuit behavior, namely analog circuit-level models, can be used as the basis of formal verification. We describe a semi-algorithmic method to extract finite-state models from an analog, circuit-level model by means of homomorphic (behavior-preserving) transformations. Properties to be verified are defined by $\omega$-automata, which constitute the formal specification of correct circuit behavior. Efficient algorithms for testing language containment of automata then can be applied to verify properties of the finite-state models. For each property to be verified, a homomorphism is chosen relative to that property. Proof of the

property in the finite-state model guarantees the property in the analog circuit-level model over a continuous range of input waveforms and circuit parameters. While in practice this method applies directly only to smaller circuit components, it can be used to analyze larger circuits as well by deriving a hierarchy of increasingly abstract models, through repeated applications of homomorphic transformations. Examples of extraction, homomorphism and verification are described in the paper.

This papers covers the following topics. Section 2 deals with the basics of homomorphic reduction. Homomorphic reduction provides the mathematical basis for the reduction of analog models to finite-state models, and for the hierarchical analysis of large systems. Section 3 discusses reductions from analog models to discrete models. Section 4 presents in detail the analysis of a particular circuit example. Section 5 describes reduction hierarchies and gives a further reduction of the example finite state model from section 4 which can be used in a hierarchical analysis of a larger circuit containing the example circuit as a component.

# 2 Homomorphic Reduction

We describe now our mathematical basis for formal verification, namely $\omega$-automata language containment. The models we present in this paper are built from concurrent components defined by respective $\omega$-automata. Section 2.2 describes this construction; formal verification in this context is described in section 2.3. Although this approach provides a powerful syntax to describe complex systems very concisely, it carries with it an inherent problem of computational complexity known as the "state explosion problem", deriving from the geometric growth of the system state space as a function of the number concurrent components in the system model. In the remaining two sections, we explain our strategy for managing this problem.

## 2.1 $\omega$-automata

Generally, an $\omega$-automaton $M$ is a structure used to define a set of (infinite) sequences over a set $\Sigma$. The set $\Sigma$ is called the *alphabet* of $M$; the set of sequences defined by $M$ is called the *language* of $M$ and is denoted $\mathcal{L}(M)$. Thus, $\mathcal{L}(M) \subset \Sigma^\omega$.

More specifically, $M$ is defined in terms of a set of *states* $V(M)$, a set of *initial* states $I(M) \subset V(M)$, a *transition matrix* $M_t : V(M) \times V(M) \rightarrow 2^\Sigma$ and an *acceptance condition* defined variously (see below) in terms of a set of subsets of $V(M)$. For brevity, we write $M$ in place of $M_t$. For each pair of states $(v, w)$, $M(v, w)$ is a *transition predicate* which defines the subset of $\Sigma$ which "enables" the transition $v \rightarrow w$. Let $\sigma = (\sigma_0, \sigma_1, \ldots) \in \Sigma^\omega$. A *run* of $\sigma$ in $M$ is a sequence $\mathbf{v} = (v_0, v_1, \ldots)$ of states of $M$ such that $v_0 \in I(M)$ and $\sigma_i \in M(v_i, v_{i+1})$ for all

$i \geq 0$. If some run $\mathbf{v}$ of $\sigma$ satisfies the given acceptance condition, then $\sigma$ is said to be *accepted* by $M$; $\mathcal{L}(M)$ is the subset of $\Sigma^\omega$ accepted by $M$.

Such automata are treated extensively in the literature [Cho74] and will not be developed here. However, we rely upon certain properties of a particular subclass of such automata, which we explore more fully. This subclass, called the *L-process* automata, is defined and treated in [Kur87, Kur90].

What distinguishes $L$-processes from other automata is the nature of their acceptance conditions. Conventionally, acceptance conditions are "inclusive" in the sense that if a run $\mathbf{v}$ of $\sigma$ satisfies the acceptance condition, then $\sigma$ is in the language of the automaton. In the case of $L$-processes, the acceptance condition is "exclusive" in the sense that if $\mathbf{v}$ satisfies the stated condition then the corresponding $\sigma$ is *out* of the language of the $L$-process. For this reason, with $L$-processes it is natural to refer instead to their corresponding *exception* conditions. Defining an automaton in terms of exception conditions instead of the customary inclusive acceptance conditions is convenient when the language of the automaton is defined primarily by its transition matrix, excepting certain runs which violate "fairness" or "liveness" conditions. This is the case with the models of interest to us here, whose behavior are closely linked to an underlying state-machine architecture. (Formally, this corresponds to automata $M$ with the property that $\mathcal{L}(M) = \mathcal{L}(N) \cap \mathcal{L}(F)$ where $\mathcal{L}(N)$ is prefix-closed and $F$ is small relative to $M$ [Kur90]; $N$ defines the state machine and $F$ defines the "fairness" constraints on $N$.)

An $L$-process $M$ is defined in terms of two types of exception conditions: a set of *recur edges* $R(M) \subset V(M) \times V(M)$ and *cycle sets* $Z(M) = \{C_1, \ldots, C_n\}$ where each $C_i \subset V(M)$. A run $\mathbf{v}$ of $M$ is *excepted* if it crosses a recur edge infinitely often or eventually stays inside some cycle set $C_j$. Intuitively, recur edges correspond to recurrent eventualities and cycle sets correspond to ultimate eventualities. A sequence $\sigma \in \Sigma^\omega$ is in $\mathcal{L}(M)$ provided not every run $\mathbf{v}$ of $\sigma$ is excepted. Formally, $\sigma \in \mathcal{L}(M)$ provided that for some run $\mathbf{v}$ of $\sigma$ in $M$,

1. there exists an integer $N$ such that for all $i > N$, $(v_i, v_{i+1}) \notin R(M)$ and
2. there does not exist an integer $N$ such that for some $C_j$, for all $i > N$, $v_i \in C_j$.

This seemingly minor variation in definition (exception *vs.* acceptance) provides us with a very powerful asset. While $L$-processes are no less general than other classes of $\omega$-automata in terms of the class of languages they can define (both define all $\omega$-regular languages), unlike other $\omega$-automata, $L$-processes are closed under a very simple product construction, which has the property that the language of the product is the intersection of the component languages. As we shall see, this language intersection property provides us with a tractable mechanism to analyze large models.

Following convention, if for some state $v \in V$ and some $\sigma \in \Sigma$, no transition out of $v$ is defined for $\sigma$, that is, $\sigma \notin \cup_{w \in V} M(v, w)$, then we infer a transition

from $v$ under $\sigma$ to a "death" state $D$ having no exit, with $\{D\} \in Z(M)$. Thus, a continuation of a run past $v$ with the input $\sigma$ is excepted.

## 2.2 Product of $L$-processes

We model a system in terms of component sub-systems, each sub-system modelled by an $L$-process. The *product* of these component $L$-processes models the global system. The product of $L$-processes is very intuitive: the state space of the product and the initial states of the product are each the Cartesian product of the respective component sets. The transition predicate for a product transition is the logical conjunction of the transition predicates of the respective component transitions. (In terms of alphabet symbols, conjunction corresponds to intersection.) Cycle sets and recur edges in the product are the respective unions of liftings from the components.

Formally, given $L$-processes $M_1, \ldots, M_k$ over a common alphabet $\Sigma$, define their *product* $M = \bigotimes_{i=1}^{k} M_i$ to be the $L$-process over $\Sigma$ satisfying:

$$\begin{aligned}
V(M) &= \times_{i=1}^{k} V(M_i) \text{ (Cartesian product)}, \\
I(M) &= \times_{i=1}^{k} I(M_i), \\
M(\mathbf{v}, \mathbf{w}) &= \cap_{i=1}^{k} M_i(v_i, w_i), \\
R(M) &= \{(\mathbf{v}, \mathbf{w}) \in V(M) \times V(M) | (v_i, w_i) \in R(M_i) \text{ for some } i\}, \\
Z(M) &= \{\{\mathbf{v} \in V(M) | v_i \in C_j\} | C_j \in Z(M_i) \text{ for some } i\}.
\end{aligned}$$

There is an interpretation of "$L$-process" which is useful when the alphabet $\Sigma$ is defined by the values of system variables. For example, if system behavior is understood in terms of the values of system variables $x_1, \ldots, x_k$, then the alphabet associated with this system could be the set of all $k$-dimensional vectors $(a_1, \ldots, a_k)$ where for each $i$, $a_i$ is a possible value of $x_i$. The notational burden of manipulating such an alphabet can be eased by using, for example, the "predicate" $(x_1 = a_1) * (x_2 = a_2)$ (where $*$ is understood as *and*) to denote the subset of the alphabet satisfying this constraint. Under this convention, the transition predicates of $L$-processes are elements in the Boolean algebra of all such predicates and thus we may use *or* and *negation* as well, in the construction of transition predicates. In fact, the $L$ refers to this Boolean algebra, and the transition matrix of a product of $L$-processes is the tensor product of the component transition matrices, over this Boolean algebra; $L$ determines the alphabet $\Sigma$: it is the set of atoms of $L$ [Kur87].

It is easy to check that for any $\sigma \in \Sigma^\omega$, if $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \ldots)$ is a run of $\sigma$ in $\otimes M_i$, then $\mathbf{v}_i = (v_{0i}, v_{1i}, \ldots)$ is a run of $\sigma$ in $M_i$, for $i = 1, \ldots, k$, and conversely. Furthermore, $\mathbf{v}$ is excepted by $\otimes M_i$ if and only if $\mathbf{v}_i$ is excepted by $M_i$ for some $i$. Thus,

$$\mathcal{L}(\otimes M_i) = \bigcap_{i=1}^{k} \mathcal{L}(M_i). \tag{1}$$

## 2.3 Tasks and language containment

Given a system modeled by an $L$-process $M$ (perhaps $M = \otimes M_i$ for components $M_i$, or not), we specify properties we wish to prove about $M$ in terms of $L$-processes $T_1, \ldots, T_m$ (over the same alphabet $\Sigma$). We say that each $T_j$ is a *task* for $M$, and that $M$ *performs* the task $T_j$ provided $\mathcal{L}(M) \subset \mathcal{L}(T_j)$. Task-performance provides the context for the formal verification we perform here. Thus, for us, verification is exclusively *relative* to stated tasks, and has no intrinsic meaning by itself.

The language intersection property 1 has the following interpretation for a system $M$ modelled by a family of $L$-processes: $M = \otimes M_i$. The set of possible behaviors of a system component $M_1$ is decreased through its coordination with other components. Thus,

$$\mathcal{L}(M_1) \supset \mathcal{L}(M_1 \otimes M_2) \supset \mathcal{L}(M_1 \otimes M_2 \otimes \cdots). \tag{2}$$

On account of this, if we succeed to show that for some $m$, $M_1 \otimes \cdots \otimes M_m$ performs a given task, then the same holds for $M$.

## 2.4 Reduction and the state explosion problem

Customary algorithms for testing the performance of a task $T$ by $M$ entail a search of the state space of $M \otimes T$ [Kur87]. However, if $M$ is defined in terms of components: $M = \otimes M_i$, then the size of its state space is geometric in the size of the component state spaces, and may be too large to search. (This is called the "state explosion problem".)

One way around this problem is to exploit 2. This may be helpful if a task is "localized" in the sense that for some $m$,

$$\mathcal{L}(M_1 \otimes \cdots \otimes M_m) \subset \mathcal{L}(T). \tag{3}$$

By 2 this implies that $\mathcal{L}(M) \subset \mathcal{L}(T)$; if $m$ is sufficiently small, 3 may be verified in the customary fashion through a search of $M_1 \otimes \cdots \otimes M_m \otimes T$. Unfortunately, it is often the case that a task is "global" in the sense that 3 fails for all small $m$ (although presumably it is performed for large $m$). In this case, more powerful methods must be applied to test task performance.

In order to test

$$\mathcal{L}(M) \subset \mathcal{L}(T)$$

it may be possible to find a *reduction* $M'$ which is "simpler" than $M$ (in a sense to be specified) and a $T'$ associated with $T$ such that

$$\mathcal{L}(M') \subset \mathcal{L}(T') \Rightarrow \mathcal{L}(M) \subset \mathcal{L}(T). \tag{4}$$

One obtains such a reduction by means of a behavior-preserving transformation called a homomorphism. The homomorphism maps states of $M$ into states of

$M'$ (in a many-one fashion), transition predicates in $M$ to (abstracted) transition predicates in $M'$, and likewise maps $T$ to $T'$, in such a way that any behavior of $M$ (sequence of alphabet symbols in $\mathcal{L}(M)$) excepted by $T$, maps to a corresponding behavior of $M'$ which is excepted by $T'$. Given a homomorphism (which may be defined implicitly through homomorphisms on component automata, as explained below), the reduction itself can be carried out algorithmically.

A particularly trivial form of homomorphic reduction is state-space minimization: an automaton sometimes may be replaced by another automaton with fewer states, but the same language. More significant, however, are state-space reductions accompanying homomorphic abstractions of inputs; these typically lead to much greater reductions in state-space size than state-space minimization alone. For example, suppose that the alphabet $\Sigma$ were defined as the set of values of a pair of variables, each with range $1, ..., N$. If the correctness of a particular task is independent of the respective variable values, so long as the two values have have the same parity, then, the $N^2$ possible values of the pair may be abstracted to 2 values: "same" and "opposite", forming a new alphabet with only two elements. If each state is associated with a variable pair, then $N^2$ states are homomorphically reduced to 2 states. If the original alphabet distinguishes states, as in the case of a Moore machine or transducer model, state-space minimization alone would provide no reduction.

An $L$-process homomorphism is built from a standard language homomorphism. Let $\Sigma$ and $\Sigma'$ be arbitrary sets and let

$$f : \Sigma \rightarrow \Sigma'$$

be an arbitrary map. Then $f$ induces a map

$$\Phi : \Sigma^\omega \rightarrow (\Sigma')^\omega$$

by

$$\Phi(\sigma)_i = f(\sigma_i).$$

The map $\Phi$ is called a *language homomorphism* and $f$ is said to be its *support*.

Now, let $M$ be an $L$-process over the alphabet $\Sigma$, and let $M'$ be an $L'$-process over the alphabet $\Sigma'$. Let $\Phi_s$ be a (many-to-one) mapping from the states of $M$ to the states of $M'$, and let $\Phi$ be a language homomorphism as above. We say that $(\Phi_s, \Phi)$ is a (*process*) *homomorphism* from $M$ to $M'$ if, for all $v, w$,

$$\sigma \in M(v, w) \Rightarrow f(\sigma) \in M'(\Phi_s v, \Phi_s w),$$

that is, if whenever $M$ can make a transition from $v$ to $w$, then $M'$ can make a corresponding transition from $\Phi_s(v)$ to $\Phi_s(w)$.

A basic theorem for homomorphic reduction[Kur87] states that given process homomorphisms

$$(\Phi_s, \Phi) : M \rightarrow M'$$

and

$$(\Psi_s, \Phi) : T \rightarrow T'$$

then 4 holds.

## 2.5 Component-wise reduction

Thus, by abstracting the alphabet as well as the state space, one can achieve a far greater reduction in complexity than reducing the state space alone. However, if to do this one must construct the product $M = \otimes M_i$, then the complexity of this intermediate step (constructing the product) would defeat the purpose of the reduction. Fortunately, it is not necessary to construct the product in order to define a homomorphism on the product. Instead, it is possible to define the product homomorphism implicitly through homomorphisms on components. This will satisfy the condition of the reduction theorem just cited, implying 4. Explicitly, the language intersection property 1 guarantees that the product of respective images of component homomorphisms has a language equal to the image of the language of the original system.

Specifically, let $M_1, ..., M_k, T$ be $L$-processes over the alphabet $\Sigma$ with $M = \bigotimes M_i$, let $M'_1, ..., M'_k, T'$ be $L'$-processes over the alphabet $\Sigma'$ with $M' = \bigotimes M'_i$, let $\Phi$ be a language homomorphism as above, and suppose for each $i$,

$$(\Phi_i, \Phi) : M_i \rightarrow M'_i,$$

as well as

$$(\Psi_s, \Phi) : T \rightarrow T'$$

are process homomorphisms. Then for each $i$,

$$\Phi : \mathcal{L}(M_i) \rightarrow \mathcal{L}(M'_i)$$

and thus by 1,

$$\Phi : \mathcal{L}(M) = \bigcap \mathcal{L}(M_i) \rightarrow \bigcap \mathcal{L}(M'_i) = \mathcal{L}(M')$$

Letting $\Phi_s = \times \Phi_i$, it follows that

$$(\Phi_s, \Phi) : M \rightarrow M'$$

is a process homomorphism, and 4 follows by the result cited in section 2.4. Thus, it is sufficient to verify only that each $(\Phi_i, \Phi)$ and $(\Psi_s, \Phi)$ are process homomorphisms.

# 3 Reduction of Continuous Models

Our analysis begins with an analog transistor circuit model, similar to models used in circuit simulation, and a "task", or $\omega$-regular language property to be verified. The circuit model is standard for circuit analysis, defined by a system of ordinary differential equations governing the circuit behavior. By starting with this kind of model, we insure that our results can be related to well-established, physically accurate device modeling techniques. The task is defined by an $\omega$-automaton, representing some desired circuit behavior at an abstract, switching-theoretic level. The interpretation of physically observable quantities in the circuit model in terms of abstract symbolic quantities in the task is defined precisely through a support map, which translates each continuous function of time of the analog model into a sequence of symbols in the finite state model. Thus, if the task specifies a set of valid sequences of binary values, we have a well-defined notion of what this means in terms of signal waveforms in the continuous model. (Although one may expect time-sampling to be a natural candidate for the support map, we will see this is insufficient, as slope must also be taken into account.) Proving that a task is performed is accomplished by abstracting the continuous model into a non-deterministic finite automaton. This abstraction is a homomorphic reduction of the continuous model relative to the task to be verified, and may be more or less detailed, according to the demands of verifying the given task. The construction of the finite model by homomorphic reduction insures that the interpretation of every solution of the system of ODE's is a sequence in the language of the automaton. As a result, we can prove that the task is performed in the circuit model by testing language containment between the task and the extracted finite model. This test can be performed automatically by the software tool *COSPAN* [HK88]. Moreover, we can construct the extracted finite state model such that the homomorphism holds over a range of circuit parameters due to temperature and process variations.

## 3.1 Circuit-level models

The circuit-level model is our lowest-level model of circuit behavior. Both time and state in this model are continuous. As in circuit level simulation, the model is a network of voltage-variable current sources, independent voltage sources, and capacitances[1]. The state of the circuit-level model is a vector quantity $\mathbf{x} = (x_1, \ldots, x_n)$, which assigns a voltage to each node in the network. We don't include nodes which are controlled by independent voltage sources in the state vector $\mathbf{x}$; such nodes represent either power supply nodes, or inputs to the circuit. The states of the input nodes are grouped into a vector $\mathbf{y} = (y_1, \ldots, y_m)$. In what follows, we describe briefly the construction of the node current equations which result in an ODE model of the dynamic behavior of the

[1] Inductances, in the type of circuit we analyze, are negligible.

circuit.

We begin with the capacitors. The current flowing from the '+' to the '−' terminal of each capacitor has the form

$$I_c = C(\dot{v}_+ - \dot{v}_-),$$

where $v_+$ is the voltage at the '+' terminal, $v_-$ is the voltage at the '−' terminal, and $C$ is the capacitance. The current at each node due to the capacitors is the sum of the currents of those capacitors whose '+' terminals are connected to the node, minus the currents of those capacitors whose '−' terminals are connected to the node. In general, these currents are a linear combination of the elements of $\dot{\mathbf{x}}$, and we can write them as a vector $\mathbf{I}_c = \mathbf{C}\dot{\mathbf{x}}$ for some $n \times n$ matrix $\mathbf{C}$.

The currents in the voltage-dependent current sources are given by a vector of functions

$$\mathbf{i}(\mathbf{x}, \mathbf{y}) = \begin{bmatrix} i_1(\mathbf{x}, \mathbf{y}) \\ \vdots \\ i_k(\mathbf{x}, \mathbf{y}) \end{bmatrix}$$

where $k$ is the number of transistors in the circuit. For reasons that will become apparent in the next section, we restrict ourselves to functions $i_1 \ldots i_k$ which are either monotonic non-decreasing or monotonic non-increasing in each element of $\mathbf{x}$ and $\mathbf{y}$. This is not a serious limitation, since the $I/V$ curves of the commonly used circuit elements, such as MOS and bipolar transistors, are monotonic.[2] The net contribution of these current sources to the current at each node can be computed by multiplying $\mathbf{i}(\mathbf{x}, \mathbf{y})$ by an $n \times k$ matrix $\mathbf{K}$ which represents the connectivity of the current sources. This matrix has a 1 in position $ij$ if node $i$ is connected to the '+' terminal of source $j$, and a $-1$ if it is connected to the '−' terminal. By Kirchoff's current law, the total current flowing out of any circuit node must be zero. Thus

$$0 = \mathbf{C}\dot{\mathbf{x}} + \mathbf{K}\mathbf{i}(\mathbf{x}, \mathbf{y}) \tag{5}$$

which is a system of first order non-linear ODE's. It is not difficult to show that under fairly general conditions, the matrix $\mathbf{C}$ is non-singular. The matrix $\mathbf{C}$ is non-singular if the capacitance from each node to the ground (datum) node is non-zero. This is because $\mathbf{C}$ is a sum of singular matrices of the form

$$\begin{bmatrix} \ddots & & & & \\ & C & \cdots & -C & \\ & \vdots & & \vdots & \\ & -C & \cdots & C & \\ & & & & \ddots \end{bmatrix}$$

[2] The only notable exception to this rule is the n-type tunnel diode, which is not commonly used in digital circuits.

corresponding to capacitors between state nodes, plus a diagonal matrix $D$ whose elements correspond to the capacitors to ground (or other DC nodes). If all of these capacitances are positive, $D$ is non-singular, hence the sum is non-singular. Since capacitances to ground tend to dominate in MOS integrated circuits, inverting the $\mathbf{C}$ matrix presents no problem in practice. The system can be rewritten as

$$\dot{\mathbf{x}} = -\mathbf{C}^{-1}\mathbf{K}\mathbf{i}(\mathbf{x}, \mathbf{y}) \tag{6}$$

This system of equations will be referred to as the "circuit level" model, $M$. Given an initial state $\mathbf{x}(0)$ and an input function $\mathbf{y}(t)$, which is continuous almost everywhere, this system has a unique solution $\mathbf{x}(t)$. We call each such pair $\langle \mathbf{y}(t), \mathbf{x}(t) \rangle$ a *trajectory* of $M$. The set of all trajectories of $M$ will be denoted $\mathcal{T}(M)$. Since time is viewed as semi-infinite, each element of this set is a function $\mathcal{R}^+ \rightarrow \mathcal{R}^m \times \mathcal{R}^n$.

## 3.2 Reduction from continuous to discrete models

Our goal in this section is to form a precise correspondence between a system of differential equations $M$ and a finite $\omega$-automaton $M'$. Although the system of equations is deterministic (in the sense that it has a unique solution for every input function), the automaton model may be highly non-deterministic, due to the abstraction of the state space. Our only further requirement on $M'$, which we discuss later, is that the extent of this non-determinism should not conflict with our goal to verify the performance of a given task; for each given task, the nature of tolerable non-determinism may vary, and thus so may our choice for $M'$.

The formal language of $M'$ is the set of infinite sequences of symbols accepted by $M'$ and is denoted $\mathcal{L}(M')$. The correspondence between the two models is defined by a map $\Phi$ from $\mathcal{T}(M)$ to $\mathcal{L}(M')$. This map has the form $(\mathcal{R}^+ \rightarrow \mathcal{R}^m \times \mathcal{R}^n) \rightarrow \Sigma^\omega$, where $\Sigma$ is the alphabet of $M'$. For any $\Delta t > 0$, we can think of a trajectory in $\mathcal{T}(M)$ as the concatenation of a sequence of partial trajectories on the interval $[0, \Delta t]$. Under this interpretation, any map from the partial trajectories into $\Sigma$ defines a support for a classical language homomorphism $\mathcal{T}(M) \rightarrow \mathcal{L}(M')$. Thus, dividing continuous time into uniform intervals of length $\Delta t$, each $\mathbf{x} \in \mathcal{T}(M)$ corresponds to the sequence in $\mathcal{L}(M')$ defined by mapping the resulting partial trajectories to the symbols in $\Sigma$ defined by the support. As an example of a useful support map, consider figure 1. The continuous curve depicts a trajectory for one of the circuit nodes, $x_j$ with the corresponding sequence of symbols in $\Sigma$ given below. The alphabet $\Sigma$ in this case is the set $\{0, 1, X\}$, where 0 corresponds to an unambiguous logic 0, 1 corresponds to an unambiguous logic 1, and $X$ corresponds to an ambiguous signal, or a "glitch". An ambiguous signal is defined in terms of bounds on the slope of $x_j$ as a function of $x_j$. A typical such bounding region is shown in figure 2. If the hatched region is entered at any time during the interval, the corresponding symbol is X. This

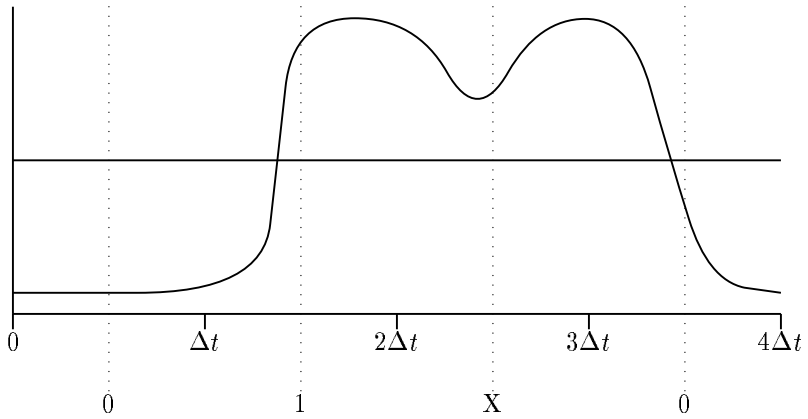

Figure 1: Mapping a trajectory to a sequence of symbols.

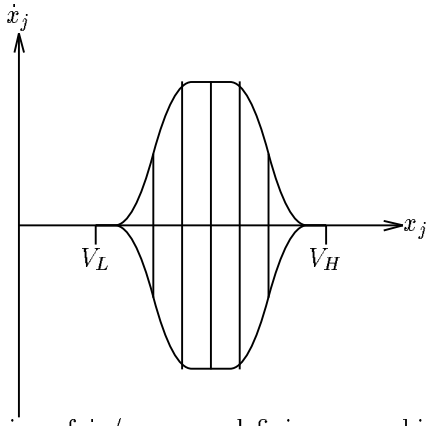

Figure 2: Region of $\dot{x}_j/x_j$ space defining an ambiguous signal.

situation is exemplified by the "glitch" depicted in figure 1. For unambiguous signals, the symbol is determined by the value of $x_j$ at the midpoint of the interval (indicated by the dashed lines in figure 1). If $x_j$ is above the logic threshold at the midpoint, the symbol is 1, otherwise it is 0. The symbols in $\Sigma$ generally are tuples $\langle \sigma_1, \sigma_2, \ldots, \sigma_l \rangle$, where each element is chosen from $\{0, 1, X\}$ and corresponds in the above manner to a particular element of $\mathbf{y}$ or $\mathbf{x}$.

This particular choice of mapping from trajectories to sequences is designed to preserve only certain important aspects of the waveform from an abstract point of view. Clearly, since not all waveforms are valid digital signals, the map must preserve the distinction between valid waveforms and waveforms with "glitches". This is the purpose of the X value. The symbol X in this case does not have the usual interpretation of "don't care", but instead indicates a

failure of some component of the system to produce a valid waveform. This information is important, since we do not usually require that a circuit produce valid output when such a signal is applied to its input. Another somewhat unusual aspect of this mapping is the assignment of a value of 0 or 1 based on sampling at the midpoint of the interval. It would seem at first glance that one would want to preserve information about more than one point in the interval. Using this mapping, for example, if one observes a 0 followed by a 1, there is no information indicating the exact time between these two samples at which the signal crosses the logic threshold. However, such a mapping may still preserve enough information about the waveform to determine whether it is correct, since no reasonable specification would require a transition to occur at a certain time with arbitrary precision. Instead, there should exist a $\Delta t$ sufficiently small to base acceptance of a waveform only on a sampling at this interval. Another reason for taking the sampling approach is that we will be dealing not with single waveforms but with sets of waveforms falling between time varying bounds. In this situation, it is not possible to determine the exact time at which a given signal crosses the logic threshold. Since this information is not available from our analysis, it is useless to attempt to preserve it in the mapping from waveforms to sequences.

The discrete model $M'$ is said to be a $\Phi$-*reduction* of the continuous model $M$ if $\Phi$ maps every trajectory in $\mathcal{T}(M)$ onto a sequence in $\mathcal{L}(M')$:

$$\mathcal{T}(M) \xrightarrow{\Phi} \mathcal{L}(M')$$

This condition implies that any property we might prove about the language $\mathcal{L}(M')$ can be pulled back via the map $\Phi$ to a property of the set of continuous trajectories $\mathcal{T}(M)$ [Kur87]. This is true whatever the choice of support map. It is necessary only to choose a map which allows the desired properties of $\mathcal{T}(M)$ to be proved at the discrete level, and which makes the construction of $M'$ tractable.

Given a continuous model $M$ and a support map, we construct the reduced model $M'$ by means of a map from the state space of $M$ to the state space of $M'$. In this map each state of the discrete model corresponds to a region of the state space of the continuous model (*ie.*, some subset of $\mathcal{R}^n$). We take a very simple approach: we divide the state space of $M$ into a rectilinear grid, as depicted in figure 3. Each state of $M'$ corresponds to one ($n$-dimensional) rectangular region of this grid. The function that maps each point in $\mathcal{R}^n$ onto the grid region containing it is called the *state map* and is denoted $\Phi_s : \mathcal{R}^n \rightarrow V(M')$. (The points along the grid partitions can be assigned to any adjacent rectangle without affecting the results which follow.)

Recall that a sequence $\sigma = (\sigma_0, \sigma_1, \ldots)$ is in the language of $M'$ if there exists an (unexcepted) run $\mathbf{v} = (v_0, v_1, \ldots)$ such that for all $i \geq 0$, $\sigma_i \in M'_i(v_i, v_{i+1})$. If $M'$ is a $\Phi$-reduction of $M$, then every trajectory of $M$ maps onto sequence in the language of $M'$ via $\Phi$. The following is a sufficient condition for this:

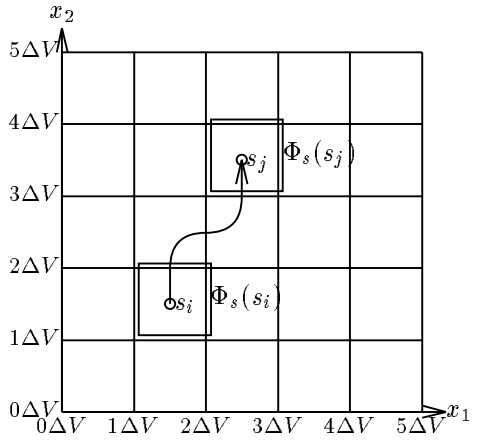

Figure 3: Mapping points to grid squares.

**Condition 1** *For all trajectories* $\langle \mathbf{y}(t), \mathbf{x}(t) \rangle$ *in* $\mathcal{T}(M)$*, the transition matrix element* $M_t(\Phi_s(\mathbf{x}(0)), \Phi_s(\mathbf{x}(\Delta t)))$ *contains the initial symbol of the sequence* $\Phi\langle \mathbf{y}(t), \mathbf{x}(t) \rangle$*.*

In other words, if there is a partial trajectory over the interval $[0, \Delta t]$ which begins in grid square $s_i$ and ends in grid square $s_j$, then there is a transition from $s_i$ to $s_j$ on the symbol associated with that partial trajectory. If this condition holds for the time interval $[0, \Delta t]$, then since the system $M$ is time-independent (and the initial state $\mathbf{x}(0)$ is arbitrary), it must also hold for any interval of duration $\Delta t$. In particular, for all $n \geq 0$, the element $M_t(\Phi_s(\mathbf{x}(n\Delta t)), \Phi_s(\mathbf{x}((n+1)\Delta t)))$ of the transition matrix contains the $n^{th}$ symbol of the sequence $\Phi\langle \mathbf{y}(t), \mathbf{x}(t) \rangle$. Hence the sequence of states $\Phi_s(\mathbf{x}(n\Delta t))$ for $n = 0, 1, \ldots$ is a run of $\Phi\langle \mathbf{y}(t), \mathbf{x}(t) \rangle$ in $M'$, hence $\Phi$ maps all trajectories in $\mathcal{T}(M)$ onto sequences in the language of $M'$, hence $M'$ is a $\Phi$-reduction of $M$.

Constructing a model $M'$ that satisfies condition 1 is not straightforward, however, because it is not possible to compute the trajectory of $M$ for all initial conditions and all input functions $\mathbf{y}(t)$. Instead, we break the problem down into a case analysis. We group the initial conditions and input functions into classes using upper and lower bounds. For the initial conditions, we provide an upper and lower bound on each state component $x_i$, and for the input function, we provide functions of time bounding each component $y_i$ from above and below. For each case, we then use numerical methods to solve for upper and lower bounds on the component functions of $\mathbf{x}(t)$. The symbol $\sigma$ is then added to $M_t(s_i, s_j)$ if (1) $s'_i$ contains any points falling within the bounds at $t = 0$, (2) $s'_f$ contains any points falling within the bounds at $t = \Delta t$ and (3) some trajectory falling within the bounds maps to the symbol $\sigma$. This is illustrated in figure 4.

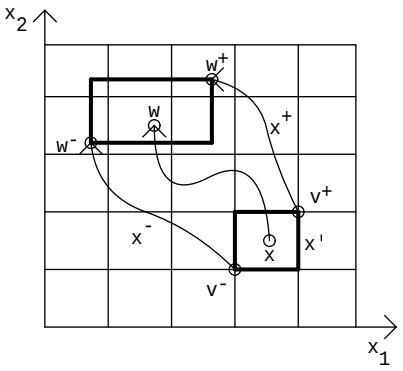


Figure 4: The trajectories of $\mathbf{x}^+$ and $\mathbf{x}^-$ provide bounds for the dynamics of all states $\mathbf{x} \in \mathbf{x}'$, for the time-step $\Delta t$, giving a computational procedure to define the set of all "next" states $\mathbf{w}'$ satisfying $\mathbf{w} \in \mathbf{w}'$, $\mathbf{w}^- \leq \mathbf{w} \leq \mathbf{w}^+$ (component-wise inequality), where $\mathbf{v}^-$ and $\mathbf{v}^+$ are the respective initial values of $\mathbf{x}^-$ and $\mathbf{x}^+$, and $\mathbf{w}^-$ and $\mathbf{w}^+$ are the respective values of $\mathbf{x}^-$ and $\mathbf{x}^+$ after time $\Delta t$.

### 3.3 Bounding trajectories of the continuous model

We now describe how we establish bounds on $\mathbf{x}(t)$, given bounds on the input functions. A discussion of how to establish bounds on the input functions is deferred to section 4, where we present an example analysis.

We assume we have established bounds on the input function given by two vector functions $\mathbf{y}^+(t)$ and $\mathbf{y}^-(t)$, providing the upper and lower bounds respectively on $\mathbf{y}(t)$. We assume that for all $0 \leq t \leq \Delta t$, and for all $1 \leq j \leq n$, $y_i^-(t) \leq y_i(t) \leq y_i^+(t)$. Throughout this section, we will use vector inequality notation, *eg.* $\mathbf{y}^-(t) \leq \mathbf{y}(t) \leq \mathbf{y}^+(t)$, to denote component-wise inequality. The upper and lower bounds on $\mathbf{x}(t)$ which we establish here, are denoted by the vector functions $\mathbf{x}^+(t)$ and $\mathbf{x}^-(t)$, respectively. The values of $\mathbf{x}^+(0)$ and $\mathbf{x}^-(0)$ are provided by assumed bounds on the initial conditions. Our goal is to define a system of differential equations governing $\mathbf{x}^-(t)$ and $\mathbf{x}^+(t)$ such that $\mathbf{x}(t)$ is bounded from above by $\mathbf{x}^+(t)$ and from below by $\mathbf{x}^-(t)$, and then to solve this system numerically. The system is constructed to satisfy the following condition:

**Condition 2** *For all* $\mathbf{x}^-, \mathbf{x}, \mathbf{x}^+$ *such that* $\mathbf{x}^- \leq \mathbf{x} \leq \mathbf{x}^+$, *and for each dimension* $i$ *of the state space,*

- $\dot{x}_i^- \leq \dot{x}_i$ *whenever* $x_i = x_i^-$ *and*
- $\dot{x}_i \leq \dot{x}_i^+$ *whenever* $x_i = x_i^+$.

This condition is sufficient to to insure that $\mathbf{x}(t)$ never escapes from the time varying "box" defined by $\mathbf{x}^-(t)$ and $\mathbf{x}^+(t)$, as stated in the theorem below. In

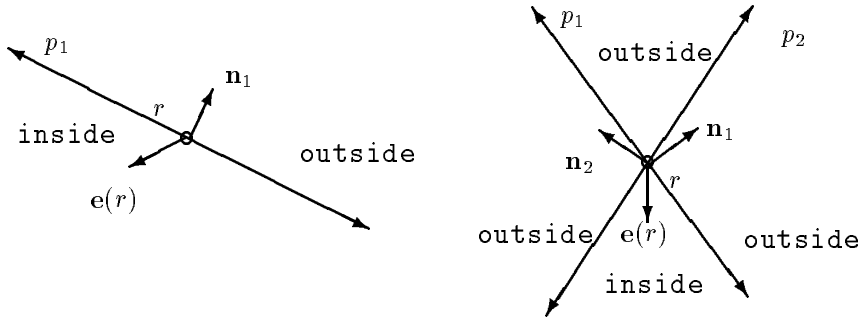


Figure 5: There is no trajectory passing through point $r$ from the inside to the outside.

particular, this implies that if $\mathbf{x}^-(0) \leq \mathbf{x}(0) \leq \mathbf{x}^+(0)$, then $\mathbf{x}^-(\Delta t) \leq \mathbf{x}(\Delta t) \leq \mathbf{x}^+(\Delta t)$. The intuition behind theorem 1 is simple. The system state $\mathbf{x}$ is a point bounded within a box whose dimensions and position are time-varying. If, at all times when the point is on one of the bounding surfaces, the outward rate of movement of the bounding surface is greater than that of the point, the point can never escape the box.

Consider a continuous, time-varying vector field $\mathbf{e} : \mathcal{R}^n \rightarrow \mathcal{R}^n$. The trajectories of this field are all of the solutions of the equation $\dot{\mathbf{x}} = \mathbf{e}(\mathbf{x})$. Define a polyhedral region $P \subset \mathcal{R}^n$ bounded by a set of planes $p_1, \ldots, p_m$, where $\mathbf{n}_1, \ldots, \mathbf{n}_m$ are the normal vectors to $p_1, \ldots, p_m$, defining the direction outward from the polyhedron. Assume that for each point $r$ on the surface of the polyhedron, the vector $\mathbf{e}(r)$ is directed inward. In other words, assume $\mathbf{e}(r) \cdot \mathbf{n_i} \leq 0$ for each plane $p_i$ containing $r$, as depicted in figure 5.

**Lemma 1** *There is no trajectory of* $\mathbf{e}$ *such that* $\mathbf{x}(0) \in P$ *and* $\mathbf{x}(\Delta t) \notin P$, *for* $\Delta t \geq 0$.

Proof. Any trajectory crossing from the inside of the polyhedron to the outside must pass through some point $r$ on the surface. Now consider each plane $p_i$ containing this point r. A trajectory must cross at least one of these planes to exit the polyhedron. However, for each such plane the field $\mathbf{e}$ is directed inward or tangential to the plane at point $r$. Since the field is continuous, it has a unique trajectory passing through point $r$. If the field is inward at $r$, this trajectory remains on the inward side of the plane. If the field is tangential at $r$, then the unique trajectory remains in the plane. In either case, the assumption that a trajectory crosses outside the plane in forward time is contradicted.

**Theorem 1** *If Condition 2 holds and* $\mathbf{x}^-(0) \leq \mathbf{x}(0) \leq \mathbf{x}^+(0)$, *then* $\mathbf{x}^-(t) \leq \mathbf{x}(t) \leq \mathbf{x}^+(t)$, *for all* $t \geq 0$.

Proof. Assume condition 2 holds and consider the simultaneous solution of the equations governing $\mathbf{x}$, $\mathbf{x}^+$ and $\mathbf{x}^-$. Let $\mathbf{s}$ be the vector whose components are $x_1, \ldots, x_n$, $x_1^-, \ldots, x_n^-$ and $x_1^+, \ldots, x_n^+$. Let $\mathbf{e}(\mathbf{s}) = \dot{\mathbf{s}}$. This defines a $3n$ dimensional continuous vector field. We assume that at time $t = 0$, $\mathbf{x}^- \leq \mathbf{x} \leq \mathbf{x}^+$. These inequalities define a polyhedral region in this space bounded by the planes $x_1 = x_1^-, \ldots, x_n = x_n^-$ and $x_1 = x_1^+, \ldots, x_n = x_n^+$. Consider any point $r$ in this region and in the plane $x_i = x_i^-$. The outward surface normal to this plane is the vector $\mathbf{n}_i = \hat{x}_i^- - \hat{x}_i$, where the vector $\hat{x}_i$ denotes the unit vector in the $x_i$ direction. Thus $\mathbf{e}(r) \cdot \mathbf{n_i} = \dot{x}_i^- - \dot{x}_i$. Condition 2 implies $\dot{x}_i^- \leq \dot{x}_i$, so $\mathbf{e}(r) \cdot \mathbf{n_i} \leq 0$. Likewise, consider the planes of the form $x_i = x_i^+$. The outward surface normal in this case is $\mathbf{n}_i = \hat{x}_i - \hat{x}_i^+$, so $\mathbf{e}(r) \cdot \mathbf{n_i} = \dot{x}_i - \dot{x}_i^+$. Again, by condition 2, $\dot{x}_i^+ \geq \dot{x}_i$, so $\mathbf{e}(r) \cdot \mathbf{n_i} \leq 0$. By the lemma, no trajectory beginning in the region $\mathbf{x}^- \leq \mathbf{x} \leq \mathbf{x}^+$ can cross outside it. The conclusion follows.

The application of this principle to our circuit models is fairly straightforward since we have restricted the current drive functions to be monotonic. Consider $x_j^+$, a single component of $\mathbf{x}^+$. Suppose that $\mathbf{x}(0)$ is bounded from above by $\mathbf{x}^+(0)$ and from below by $\mathbf{x}^-(0)$, and further that $x_j = x_j^+$. Condition 2 is satisfied if the first derivative $\dot{x}_j^+$ is an upper bound on $\dot{x}_j$. If we expand equation 6 to obtain the slope of $x_j$, letting $\kappa = \mathbf{C}^{-1}\mathbf{K}$, we get an expression of the form

$$\begin{aligned} \dot{x}_j \quad = \quad & \kappa_{j1} i_1(x_1, \ldots, x_n, y_1, \ldots, y_m) \\ & + \kappa_{j2} i_2(x_1, \ldots, x_n, y_1, \ldots, y_m) \\ & \ldots \\ & + \kappa_{jk} i_k(x_1, \ldots, x_n, y_1, \ldots, y_m) \end{aligned} \tag{7}$$

This is a linear combination of monotonic functions in $\mathbf{x}$ and $\mathbf{y}$. Given upper bounds on these quantities, we can easily construct an upper bound on such a function, by taking a sum of upper bounds on each term. A given term $l$ is either non-increasing or non-decreasing in each argument, depending on the sign of the coefficient $\kappa_{jl}$. If the coefficient is non-negative, the dependency is the same as that of the current function $i_l$, else it is the opposite. Assume that a given term $l$ is non-decreasing in an argument $x_m$ (or $y_m$). In this case, if we replace $x_m$ with any upper bound on $x_m$ the result will be an upper bound on the term. By hypothesis however, $x_m \leq x_m^+$, hence we may substitute $x_m^+$ for $x_m$ (or $y_m^+$ for $y_m$). In the opposite case, where the term is non-increasing in $x_m$ (or $y_m$), an upper bound on the term is obtained by substituting the lower bound $x_m^-$ for $x_m$ (likewise, $y_m^-$ for $y_m$). This procedure would be sufficient to construct an upper bound on $\dot{x_j}$, hence to satisfy condition 2, but a slight variation produces a much more useful bound. Condition 2 assumes that $\mathbf{x}$ lies on a particular surface of

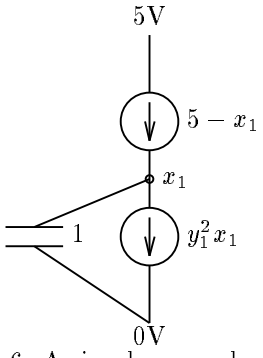

Figure 6: A simple example.

the bounding region where $x_j = x_j^+$. In the special case where $m = j$, we can therefore substitute $x_m^+$ for $x_m$, independent of whether the current function is increasing or decreasing in $x_m$. This change is significant because it often makes the difference between a diverging and a converging system of equations in modeling digital circuits (*cf.* section 3.4). A similar, but oppositely signed construction applies to the lower bound.

## 3.4 A simple example

Here it might be worthwhile to consider a simple example. The network of figure 6 has one input node $y_1$ and one output node $x_1$, two current sources, and a capacitor from $x_1$ to ground. The values in the figure are unitless, to simplify the presentation. The current source functions are arbitrary, and not meant to represent any real devices. The system which results from this network is

$$0 = (1)\dot{x}_1 + \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} 5 - x_1 \\ y_1^2 x_1 \end{bmatrix}$$

or

$$\dot{x}_1 = -\begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} 5 - x_1 \\ y_1^2 x_1 \end{bmatrix}$$

First, we derive the equation governing the upper bound $x_1^+$. We observe that the current source $i_1$ is monotonic non-increasing in $x_1$. In this case, substituting either $x_1^+$ or $x_1^-$ for $x_1$ will provide a suitable upper bound (since we only need an upper bound on the derivative when $x_1 = x_1^+$), so we choose $x_1^+$, to provide the tighter bound. The current source $i_2$ is monotonic increasing in both $x_1$ and $y_1$ (for positive voltages) and has a negative overall coefficient. Hence, we substitute $x_1^+$ for $x_1$ for the same reason as above, but we must substitute $y_1^-$ for $y_1$. The resulting equation for $\dot{x}_1^+$ is

$$\dot{x}_1^+ = -\begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} 5 - x_1^+ \\ (y_1^-)^2 x_1^+ \end{bmatrix}$$

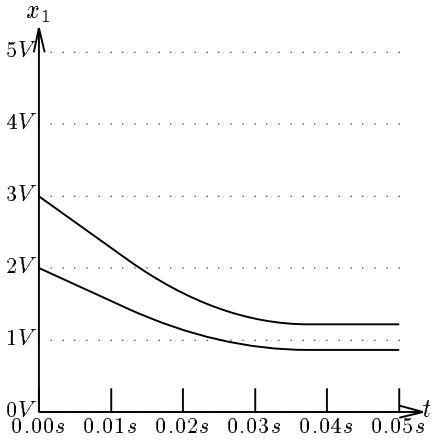

Figure 7: Solution for the bounds functions.

The equation governing the lower bound is

$$\dot{x}_1^- = -\begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} 5 - x_1^- \\ (y_1^+)^2 x_1^- \end{bmatrix}$$

Let the grid divide the voltage range into uniform intervals of $1V$, and let $\Delta t$ be 0.05. Further, let the assumed bound on the initial condition be the interval $[2V, 3V]$. We will assume, for the sake of simplicity, that the input function $y_1(t)$ is bounded in the range $[4V, 5V]$ over the entire interval. Thus $y_1^-(t) = 4V$ and $y_1^+(t) = 5V$. Figure 7 shows the result of solving out for the upper and lower bounds as a function of time (this system is easily solved without resort to numerical methods). We can see that the upper and lower bounds actually converge. This is due to the fact we have substituted $x_1^+$ for $x_1$ in the definition of $\dot{x}_1^+$ and $x_1^-$ for $x_1$ in the definition of $\dot{x}_1^-$. Making the substitutions simply according to the dependence of the current functions would have resulted in a diverging system of equations, even though the original system $M$ is converging (as the reader may verify). Convergence of the bounds in the case of converging $M$ is crucial, since diverging bounds tend to produce reductions which are too under-determined to be useful. In digital circuits, convergence is the norm, since the output of a gate tends toward an equilibrium value given a fixed input. The next step in our analysis is to observe the set of grid regions which overlap the bounds at time $t = 0$ and time $t = \Delta t$. As a result, we insert a transition in the model $M'$ from the region $[2V, 3V]$ to the regions regions $[0V, 1V]$ and $[1V, 2V]$. Note that according to the support map we described, the only possible symbol for $y_1$ is 1, and the only possible symbol for $x_1$ is 0. Next, we go on to analyze the remaining cases for initial conditions and input function. When all the reachable grid regions have been analyzed in this way, the model $M'$ is complete.

### 3.5 Accounting for parameter variation

We note here that any parameters on which the model behavior might depend can be accounted for in the equations governing $\mathbf{x}^-$ and $\mathbf{x}^+$, provided that bounds are known on these parameters and that the dependencies of the current functions on these parameters are monotonic. The current source models for transistors, for example generally depend on a number of parameters related to variations in temperature and process. Some examples of these are described in section 4.1, where the model we use of MOS transistors is described. Since the dependence of the current drive functions on these parameters are monotonic, bounds on these parameters can be used to derive a model $M'$ which is a reduction of all models $M$ whose parameters fall within the specified ranges. Any properties of the language of $M'$ therefore apply independent of parameter variations within the specification. This is of great importance for the analysis of real circuits, since variability in the electrical parameters is inevitable, and it is also provides a means to account for minor inaccuracies in the circuit level model.

Variations in the values of capacitors are a little more difficult to handle. However, since we can obtain upper and lower bounds on $i_1, \ldots, i_k$, and since the matrix $\mathbf{C}$ is a linear combination of the capacitor values, we can treat equation (5) as a linear program and use the simplex algorithm to minimize and maximize $\dot{x}_i$ subject to the given constraints.

The method we have described here for bounding $\dot{x}_i$ finds bounds on a linear combination of monotonic functions using independent upper and lower bounds on these functions. Since the arguments of the monotonic functions are not in fact independent, there may be better bounds on $\dot{x}_i$ than we obtain by this method. We have not explored this possibility. Another possible advantage may be garnered from non-uniform voltage partitions. Our experience is limited to the case reported here, where we partition each node voltage $x_i$ into a set of ranges of equal measure $\Delta V_i$ and let the state space of $M'$ be the Cartesian product of these partitions. Clearly, this is not the only possible choice. It may be advantageous to use partitions which are non-uniform, or regions which have some overlap, or perhaps to construct the set of regions based on some heuristics which are adjusted dynamically, as the reachable state space is being searched. Although we have not explored these possibilities, in the next section we do discuss the effect of varying the voltage partition interval sizes $\Delta V_i$ on the utility and the tractability of the resulting model.

## 4 Analyzing the Seitz Interlock Circuit

As an illustration of the use of homomorphic reduction in the analysis of circuit-level models, we present an analysis of the interlock element of Seitz [MC80] (see the circuit diagram in figure 8). The interlock element is designed to prevent two

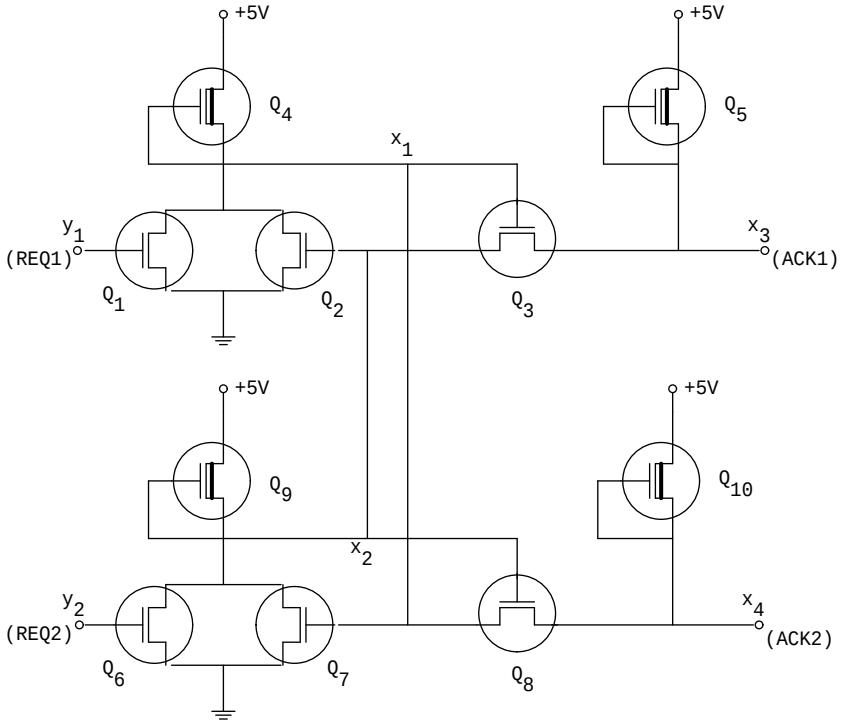


Figure 8: Seitz interlock circuit

asynchronous client circuits from gaining simultaneous access to some resource, provided the clients observe a request/acknowledge protocol. When a client requires access to the resource, it asserts its request signal, by drawing it to a low voltage level, and waits for its acknowledge signal to be asserted by the interlock. The client then may use the resource. When it no longer needs the resource, it negates its request (i.e., raises it to a high voltage level) and waits for its acknowledge signal to be negated by the interlock. What makes the interlock circuit interesting is that requests may occur completely asynchronously. If both request signals are asserted nearly simultaneously, it is possible for the circuit to enter a 'metastable state' [CM73, CW75] – the voltages $x_1$ and $x_2$ balance at an intermediate level between 'logic 0' and 'logic 1' and (in the absence of thermal noise) remain there indefinitely. This type of behavior makes the interlock impossible to analyze using switch level models. The circuit is designed so that, in this metastable state, neither acknowledgement can be asserted, and thus a conflict in the use of the resource cannot occur.

We have formally verified three tasks for the interlock, as follows:

1. *Mutual exclusion.* It is never the case that both acknowledge signals are

simultaneously asserted.

2. *Starvation freeness.* It is never the case that a request signal is asserted and not eventually acknowledged.

3. *Fairness.* It is never the case that one acknowledge signal is asserted twice consecutively while the other request signal is active.

These properties are not intended to completely characterize the behavior of the circuit. Instead, they are examples of properties a designer might rely on in constructing a system using the interlock circuit. The three properties can be represented as the respective languages of three task automata. Each of these automata embodies certain assumptions about the behavior of the environment and the circuit. For example, in all three cases, we assume that the environment obeys a four-phase protocol described below, *vis-a-vis* the request and acknowledge signals. This assumption (for REQ1/ACK1) is represented by the automaton in figure 9, whose language is all sequences in which the four-phase protocol is obeyed by the environment. To *except* from the analysis the sequences in which the four-phase protocol is violated by the environment, we take the product of this automaton with the reduced model of the interlock circuit (recall that this results in the language intersection of the two automata). In the case of the starvation freeness task, we also assume that if an acknowledge is asserted, the corresponding request signal eventually will be released. In the absence of this assumption, it would be possible for clients to "hog" the resource, in which case one could not guarantee eventual acknowledgement to any client. Simply by adding a cycle-set consisting of state 3 of the automaton of figure 9, we except this case from the analysis.

It should be noted that in general, the validity of assumptions about the environment may depend in turn upon properties of the interlock circuit. For example, one may prove that entity $A$ obeys one half of a protocol assuming entity $B$ obeys the other half, and vice versa, but this circular argument is insufficient to conclude that both entities together obey the complete protocol. This circularity can be avoided using the component-wise reduction method described in section 5.3.

Even under assumptions about the environment, it is still not possible to guarantee freedom from starvation, on account of the existence of a metastable state. If both requests are asserted nearly simultaneously, the interlock circuit may enter a metastable state where (in the absence of thermal noise, not modeled here) it will remain forever. It can be shown theoretically [CM73, CW75] that metastable states are unavoidable in bistable systems such as the interlock circuit. The starvation freeness task therefore is performed only under the assumption that the metastable condition does not persist forever. Since the exact location of the metastable state may vary with the inputs and the circuit parameters, we will in fact use a region $R_m$ of the state space to define the metastable condition. Another reason for using a region instead of a point is

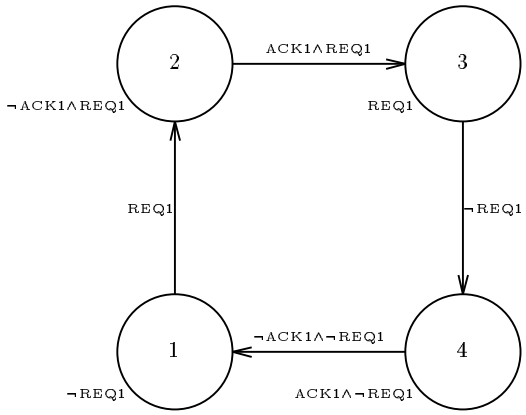


Figure 9: Automaton representing four-phase protocol assumption. The automaton remains in the same state if the condition labeling the state holds. It follows an arrow if the condition labeling the arrow holds. All other behaviors are excepted from the language it accepts.

that the discrete model $M'$ cannot distinguish individual points in the state space. Using the cycle-set mechanism, we can except from the model any sequence of states which remains in $R_m$ infinitely. When we analyze the circuit model, we consider the model with and without this exception.

## 4.1 Modeling the interlock at the circuit level

Figure 10 shows the equivalent circuit we use for a MOS transistor, in our circuit-level model. It has four parameters: the threshold voltage $V_T$, the transconductance parameter $\beta$, the channel modulation parameter $\lambda$, and the gate capacitance $C_{gs}$. Each of these parameters has a specified minimum and maximum value. Three regions of operation are distinguished: cutoff, linear, and saturation. The voltage variable current source $I_{ds}$ is defined for these regions as follows.

$$I_{ds} = \begin{cases} 0 & \text{if } V_{gs} \leq V_T \text{ (cutoff)} \\ \beta \frac{(V_{gs} - V_T - V_{ds}/2)V_{ds}}{(1 - \lambda V_{ds})} & \text{if } V_{gs} > V_T \text{ and } V_{ds} \leq V_{gs} - V_T \text{ (linear)} \\ \frac{(\beta/2)(V_{gs} - V_T)^2}{(1 - \lambda V_{ds})} & \text{if } V_{gs} > V_T \text{ and } V_{ds} > V_{gs} - V_T \text{ (saturation)} \end{cases}$$

The reader may confirm that these functions are continuous at the region boundaries, and that they are monotonically non-decreasing in $v_g$, $v_d$, $\beta$ and $\lambda$, and monotonically non-increasing in $v_s$ and $V_T$. We model $C_{gs}$ as a fixed capacitance, although in reality this capacitance is voltage variable. Also, we ignore the overlap capacitance between gate and drain.

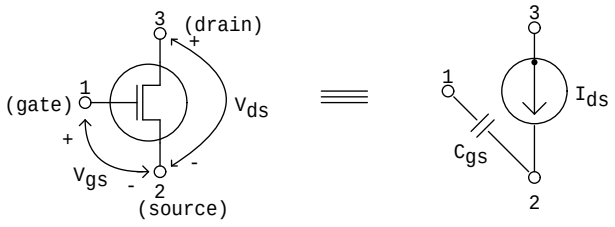


Figure 10: MOS transistor equivalent circuit.

More accurate models for MOS transistors can be found in [Nag75]. Although these more complex models also could be used in the analysis that follows, our simple model suffices for purposes of illustration. In any case, no model gives a perfectly accurate reflection of the behavior of the real circuit. Unless correct behavior of the circuit depends upon "second order" effects not reflected in the simple model, we prefer to use the simple model and allow enough variation in the model parameters (*cf.* section 3.5) to account for the inaccuracies of the model.

The equivalent circuit depicted in figure 10 is used to convert each transistor $Q_j$ in the interlock circuit into a capacitor $C_j$ and a voltage-variable current source $i_j = I_{ds}(v_{g_j}, v_{s_j}, v_{d_j}, V_{T_j}, \lambda_j, \beta_j)$. In addition, we model the load on the output nodes $x_3$ and $x_4$ with capacitances $C_{L1}$ and $C_{L2}$.

The current drive vector is

$$\mathbf{i} = \begin{bmatrix} I_{ds}(y_1, V_{ss}, x_1, V_{T_1}, \lambda_1, \beta_1) \\ I_{ds}(x_2, V_{ss}, x_1, V_{T_2}, \lambda_2, \beta_2) \\ \vdots \\ I_{ds}(x_4, x_4, V_{dd}, V_{T_{10}}, \lambda_{10}, \beta_{10}) \end{bmatrix}$$

the capacitance matrix is

$$\mathbf{C} = \begin{bmatrix} c_7 + c_3 + c_8 & -c_3 - c_8 & 0 & 0 \\ -c_3 - c_8 & c_2 + c_3 + c_8 & 0 & 0 \\ 0 & 0 & c_{L1} & 0 \\ 0 & 0 & 0 & c_{L2} \end{bmatrix}$$

and the current source connectivity matrix is

$$\mathbf{K} = \begin{bmatrix} 1 & 1 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix}$$

where the rows correspond to $x_1, \ldots, x_4$ and the columns correspond to $i_1, \ldots, i_{10}$. For example, current source $i_3$ has its '+' terminal connected to node $x_3$ and its '$-$' terminal connected to node $x_1$.

As an example of the equations governing the upper and lower bound functions $\mathbf{x}^+$ and $\mathbf{x}^-$, consider the component $x_3^+$ in the example circuit. By computing $\kappa = \mathbf{C}^{-1}\mathbf{K}$, the expansion we obtain for $\dot{x}_3$ in the example is:

$$\dot{x}_3 = -\frac{1}{c_{L0}} i_3(x_2, x_1, x_3, V_{T_3}, \lambda_3, \beta_3) + \frac{1}{c_{L0}} i_5(x_3, x_3, v_{dd}, V_{T_5}, \lambda_5, \beta_5)$$

(where $v_{dd}$ is the positive supply voltage). In the first term, the gate voltage argument is $x_2$, and the coefficient $\kappa_{33} = (-\frac{1}{c_{L0}})$ is negative. Since the current function $I_{ds}$ is non-decreasing in the $v_g$, we replace $x_2$ with $x_2^-$. Since $v_{s_3} = x_1$, and $I_{ds}$ is non-increasing in the source voltage, we replace $x_1$ with $x_1^+$. Carrying on this fashion, we arrive at

$$\dot{x}_3 = -\frac{1}{c_{L0}} i_3(x_2^-, x_1^+, x_3^-, V_{T_3}^+, \lambda_3^-, \beta_3^-) + \frac{1}{c_{L0}} i_5(x_3^+, x_3^-, v_{dd}, V_{T_5}^-, \lambda_5^+, \beta_5^+)$$

## 4.2 Reducing the model

We now describe the reduction of the circuit-level interlock model $M$ to a discrete model $M'$. There are a number of important choices to be made in constructing $M'$, which determine the complexity of the model, and its usefulness in proving properties of the system. The grid spacing, which we call $\Delta V$, determines the number of states in the discrete model. This number is $g^n$, where g is the number of grid divisions in each state space dimension (assuming this is the same in each dimension) and $n$ is the number of dimensions. Although most of these states will not be reachable from the initial region, it is clear that $\Delta V$ should be as coarse as possible. On the other hand, $\Delta V$ is the quantization uncertainty in the state of the discrete model. Increasing this parameter will result in less precise bounds on the trajectories of $M$, which will in turn add greater nondeterminism to the model, providing less information about the underlying system of equations. (In the extreme case, with only one grid division, the discrete model provides no information at all.) For this reason, choosing $\Delta V$ involves a tradeoff between computational tractability and the properties that can be proved using the discrete model.

The parameter $\Delta t$ determines the interpretation of electrical signals as sequences of digital values. Therefore, it must be fine enough to resolve any signal of long enough duration to be labeled a legitimate logical 1 or 0. The value of $\Delta t$ does not affect the number of states in $M'$, but it does affect the degree of non-determinism. If $\Delta t$ is too large, in those cases where the bounds on a given state node are diverging, this uncertainty will tend to propagate through the system and result in almost unconstrained behavior of $M'$. On the other hand, if $\Delta t$ is too small, another kind of uncertainty is introduced into $M'$. The non-determinism inherent in a reduction from a continuous to a discrete system introduces uncertainty into the rise and fall times of signals. As $\Delta t$ becomes smaller, the number of voltage regions through which a given node rises or falls

in time $\Delta t$ becomes smaller. This means that the 'quantization' uncertainty becomes larger relative to the overall voltage change, making the uncertainty in the apparent rise and fall times greater. However, if the circuit's function is delay-sensitive, that is, if its correctness depends on relationships between certain rise and fall times, then it is important that we keep the uncertainty in rise and fall times to a minimum. If $\Delta t$ must be decreased for some reason, then it is of course possible to reduce $\Delta V$ proportionately and retain the same level of uncertainty. This however could increase the number of states in the model drastically. Thus, it is important to make $\Delta t$ small enough to be able to specify the required behaviors, but no smaller. In our experience, a good compromise for $\Delta t$ is roughly one third of a typical rise or fall time of nodes in the circuit. Nodes with particularly long rise or fall times relative to other nodes in the circuit may require a finer grid spacing.

A final issue of importance in constructing the discrete model is choosing the case breakdown for the initial conditions and input functions. It is clear that for the initial conditions, this breakdown should be at least as fine as the grid that defines the state space of $M'$. We found no need in practice to use a finer division than this, however. Thus, in constructing $M'$, we perform the numerical solution for the bound functions using the boundaries of the grid square as our initial condition bounds, for each state analyzed. Choosing the input function breakdown is a more complex issue. One factor that simplifies this choice, however, is that the tasks always except the case when the input symbol contains an $X$. For this reason, we only need a breakdown of the partial input functions (over the interval $[0, \Delta t]$) which map to 0 or 1 according to the support map. This provides a constraint on the the slope of these functions as a function of voltage (see figure 2). Thus, one approach to partitioning the input functions would be according to a set of intervals on the value at time $t = 0$ and $t = \Delta t$. The constraint on the slope then provides bounds on the value of the function in the intervening interval. At this point, we have not attempted to find a general solution to the problem of partitioning the input functions. For the sake of the example, we consider here only two cases: the case where $\mathbf{y}(t)$ is bounded from above by a constant $V_L$, and the case where it is bounded from above by a constant $V_H$ (*cf.* figure 2). Clearly, these two cases do not cover the whole space of partial functions which map onto 0 or 1. On the other hand, even these simple cases cover a broad class of inputs, substantially more general than could be checked by simulation.

## 4.3 Analyzing the reduced model

We now describe in detail the analysis through which we prove that the model $M$ of the interlock circuit performs the mutual exclusion, starvation freeness and fairness tasks, subject to the previously stated assumptions. The three tasks are formalized as the languages of three task automata $T'_1 \ldots T'_3$. We use the "prime" notation to indicate that these tasks are specified at the abstract

symbolic level rather than the continuous level. The languages of these tasks may be pulled back to the continuous domain via the mapping $\Phi$, but it seems more natural to think of the tasks as being defined at the symbolic level, with the interpretation of signals at the continuous level being determined by $\Phi$.

For each task $T_i'$ we experimented with several different reductions $M_i'$. The subscript on $M_i'$ is intended to indicate the reduction we use is relative to the particular task to be proved. A reduction useful for one task may not preserve enough information about the system behaviors to prove a different task. The various reductions are obtained by varying $\Delta V$ and $\Delta t$. We do this until either a reduction is found which works (*i.e.*, the language-containment test $\mathcal{L}(M') \subset \mathcal{L}(T')$ is verified) or we conclude that the circuit is not correct. While this may seem a bit *ad hoc* and undirected, in practice, using the *COSPAN* system to verify language containment, we found this kind of experimentation to lead fairly quickly to an acceptable reduction. One reason for this is that in case of task failure, *COSPAN* produces a sequence of states which cannot be continued to a sequence accepted by the task. This usually gives a direct indication of the source of the failure. In describing the process we went through with the interlock model, we will show several of these error traces. In case of failure, we use the error trace to assess the failure, then choose new reduction parameters accordingly, and repeat the procedure. In general, our approach is to begin with fairly large $\Delta V$ (*i.e.*, a coarse grid) and reduce the size as necessary to achieve task performance. Sometimes, reducing the grid size for only one or two circuit nodes is sufficient, especially in the case where the analog behavior for these nodes is critical, as it is for $x_1$ and $x_2$ in the interlock circuit. Sometimes, of course, the model simply does not perform the task. In this case, the error track can be of use in correcting the circuit, either by changing the specified ranges of the circuit parameters, or redesigning the circuit entirely. In some cases, it is useful to use an error track as a guide to produce a simulation run of the circuit-level model using, for example, *SPICE* [Nag75] to demonstrate the failure. (This may be easy or difficult to accomplish. For example, using the error track as a guide, it was easy to simulate the metastability failure of the starvation freeness task.) If we succeed in simulating the error, then we may be assured that $M$ fails to perform $T$, and we search no farther for a homomorphism to prove the contrary. Often, however, especially in the early design stages of a circuit, the error track by itself is a sufficient indication of a design problem to motivate a change in the design or restriction of the design space, or to suggest a more constrained task formulation.

Our analysis proceeds as follows. The reduction is defined by the set of rectilinear regions comprising the states of $M'$. We fix these regions by choosing values of $\Delta V_i$ for each circuit node $i$, defining the grid spacing in each dimension. At this point, we also choose ranges for the circuit parameters. Then we carry out the reduction from $M$ to $M'$. This is done incrementally by *COSPAN* at the same time that it tests the language containment $\mathcal{L}(M') \subset \mathcal{L}(T')$. Thus there is no need to construct the entire reachable state space of $M'$ if the test

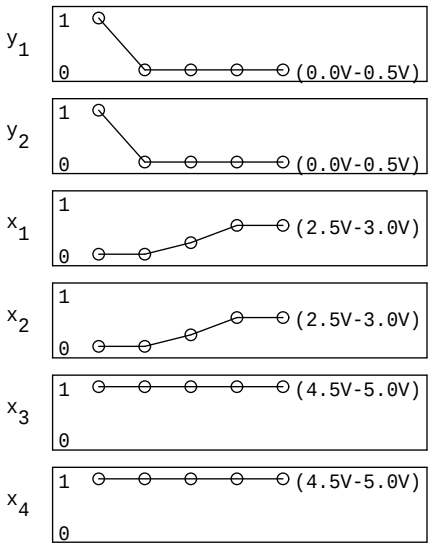


Figure 11: Error trace showing metastability.

fails before all reachable states have been analyzed.

In our first analysis of the interlock circuit, we allowed no variation of the transistor model parameters from nominal values, and set $\Delta V_i$ to be $0.5V$ for all of the circuit nodes (yielding ten grid spaces in the power supply range). A suitable $\Delta t$ value was determined by computing the minimum rise and fall times for each node. The resulting reduced model $M'$ was shown by *COSPAN* to perform the mutual exclusion task and fairness tasks, through a search of fewer than 600 states in both cases. To put this in perspective, the *COSPAN* system is capable of searching up to about 10 million states on a large computer, at a rate of around 100 to 10,000 states per second (depending upon the complexity of the system). On the other hand, $M'$ failed to perform the starvation freeness task (since it had no exception for metastable behavior), producing the error trace depicted in figure 11. This trace shows the two request signals being asserted simultaneously, causing the model to enter (a region containing) the metastable state. Note that although the figure depicts a finite trace, the last state should be understood to recur infinitely. Next, we sought to verify the starvation freeness task with the exception for metastability. To do this we set $R_m$ to be a region containing the last state in figure 11, and performed the analysis again.

With this addition, the starvation freeness task again failed, however, producing the trace of figure 12. In this trace, ACK1 is asserted by the interlock, but REQ1 is never released by the client (again, understand the last state of the trace to repeat forever). As a result, client 2 never can receive an acknowledge.

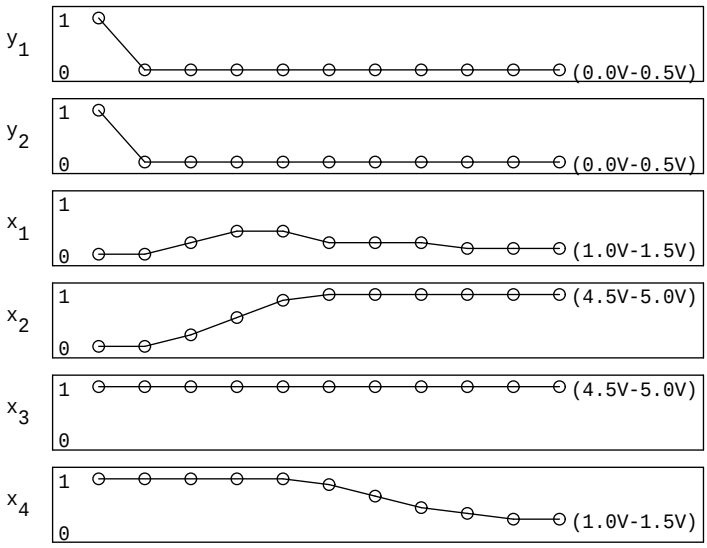


Figure 12: Error trace showing "unfair" behavior of $E_2$.

According to our assumptions, however, traces of this kind should be excepted from the analysis. We did this, by modifying the task automaton, adding a cycle-set to except the case in which REQ1 and ACK1 are both asserted forever after some point. Given this last exception, the proof of task performance succeeded after a search of 609 states.

At this point, we introduced into the model some variation in the circuit parameters of the transistors. We allowed the threshold voltage ($V_t$) parameters to vary (independently) by $\pm 0.2V$ and the transconductance parameters ($\beta$) to vary (independently) by $\pm 10\%$. In this second analysis, the mutual exclusion and fairness tasks were again performed, although this time *COSPAN* searched approximately twice as many states. (Note here that the size of the state space of $M'$ has not increased, but the number of reachable states has increased due to the weaker bounds on the transitions possible in $M'$.) The starvation freeness task failed, however, producing the error trace of figure 13. The difficulty here relates to the uncertainty in rise and fall times discussed in section 4.2. In this case, the minimum fall rate of $x_3$ was smaller than $\Delta V_3/\Delta t$ in a certain region, allowing the model $M'$ to "hang" in that region; this uncertainty in the fall time coupled with the nondeterminism built into $M'$, caused the modeled fall time to become effectively infinite. The most straightforward remedy we found was to decrease $\Delta V_3$ from $0.5V$ to $0.25V$, without decreasing the grid spacing for any other nodes. Doing this enabled us to prove that the starvation freeness task was performed, after a search of 1310 states. This illustrates the sensitivity of the reduction to the task being verified.

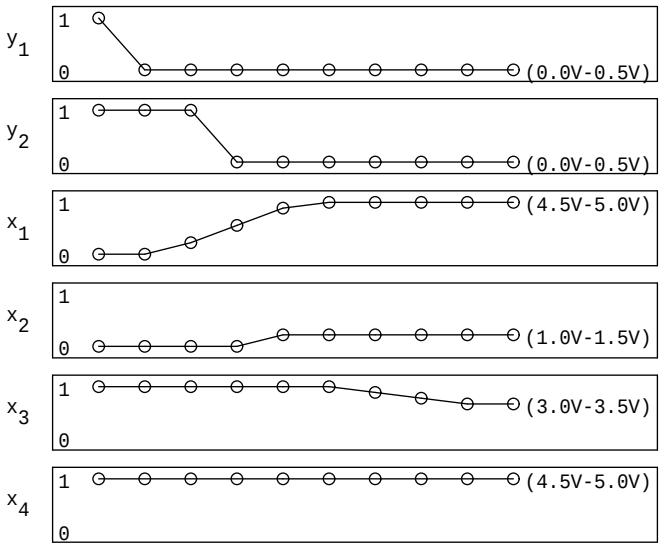


Figure 13: Error trace showing slope problem.

# 5 Analyzing larger circuits

In this section, we consider the analysis of circuits larger than the interlock example. We will need more powerful methods for this purpose, since directly reducing a substantially larger circuit to a discrete model would be impossible, due to the geometric growth in the size of the state space. Instead, we reduce the components of a larger circuit separately using a method called component-wise reduction.

## 5.1 Bottom-up analysis of circuit-level models

Breaking a continuous model into components is accomplished simply by partitioning the differential equations governing the model. Each partition is an underconstrained system, so its solutions for a given input function $\mathbf{y}(t)$ are not unique. The unconstrained variables can be treated in the same way as inputs in the reduction process. By definition, the trajectories of the system $M$ are the intersection of the trajectories of the components, since this is what is meant by solving the system of equations simultaneously. If we use the same language map to reduce all of the components separately, we obtain a very useful result: the product of the reduced models is a reduction of the composite system of equations. This follows directly from the language intersection property 1.

Constructing the product of the reduced models is not our goal, however. Instead, we further reduce these automata, using a common language homomor-

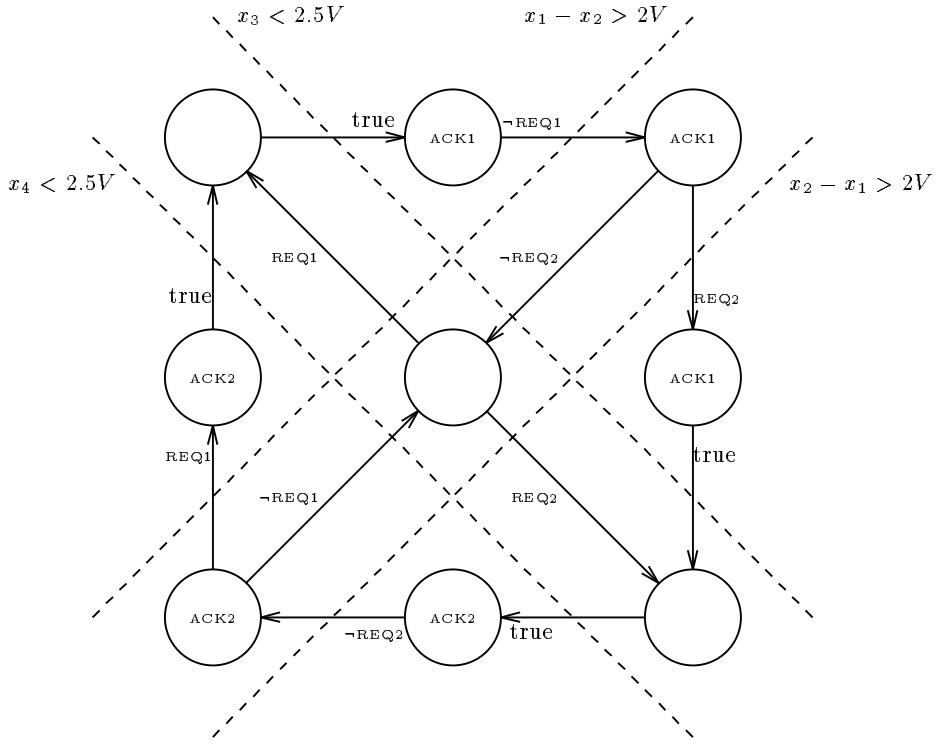


Figure 14: Reduced interlock model ($M''$). Labels on arcs represent transition conditions in terms of Boolean inputs REQ1 and REQ2. Labels on nodes represent assertion of Boolean outputs ACK1 and ACK2. There is an implicit self-loop at each state with transition condition *true*. That is, the model may wait an arbitrary amount of time before making a non-self transition. The dotted lines represent voltage region boundaries in the state map from $M'$ to $M''$, and are labeled with inequalities on the state space components.

phism, to a set of second-level reductions, which may be much more abstract, but still sufficient to show correct interaction of the components. To illustrate this, we return to the interlock circuit. This circuit is intended to be a building block in speed-independent circuit designs [Sei80]. For this reason, correct interoperation of the interlock circuit with other system components should not depend on delays between its inputs and outputs. If we reduce the discrete model $M'$ to a second level model $M''$, with unconstrained delays, we obtain a substantial reduction in states. Such a reduction is pictured in figure 14.

A state map which might be used to prove this reduction is suggested by the dashed lines in the figure. No state map is needed in practice, however, since the language map induces a map from state sequences of $M'$ to state sequences of $M''$. We can exploit this relation to test the reduction automatically, using

COSPAN. By constructing the product of the second level reductions, we can verify tasks in a substantially reduced model, provided the tasks are independent of the component speeds. Once again, the reduction is relative to the task. If the task performance depends on some speed requirements, a reduction might be found which retains some of the delay information.

## 5.2 Top-down design hierarchies

In addition to this bottom-up analysis of circuits, component-wise reduction also may be applied to top-down refinement in the design process. Instead of beginning with low-level models of the components, one begins with a very abstract model of the system, and refines the design through successive levels until the system is fully specified in terms of low-level component models. Each level of the design is proved to be a reduction of the level below. As a result, a task which is verified at a high level is guaranteed to carry through to lower levels. Verification begins at the architectural level, before design details are filled in, saving wasted design time due to propagation of errors from high-level design to low-level design. Generally, the reductions in this design hierarchy are proved relative to assumptions about the low-level behavior of the system. These assumptions are carried through and must be discharged at a lower level of the design hierarchy. For example, variables at the abstract levels usually have types such as integer, scalar, or Boolean. At the lower levels, however, the only available signal type is $\{0, 1, X\}$. The language map defines the encoding of the high-level types in terms of $\{0, 1\}$. Since it is inconvenient to maintain the $X$ value at the higher levels, we perform the reductions relative to the assumption that X never occurs in the language. This assumption is represented by an automaton, which becomes a task to be verified at a lower level (the level of $M'$, for example). In general, this task is proved by constructing the product of low level component models, to verify that they interact correctly, without producing ambiguous signals.

Another example of discharging high level design assumptions by low level analysis is the way in which delay constraints are commonly handled. In producing the design, one may make some assumptions about the relative speeds of some components. One expresses these assumptions, for example, in terms of an automaton which excepts sequences in which component $A$ changes state some number of times while component $B$ continues to delay. This assumption then is verified as a task using the product of the low level models of A and B.

Top-down design and bottom-up analysis are easily seen to be complimentary. As one progresses to lower levels, a design becomes increasingly dependent on the underlying technology. At some point, it is no longer possible to push the design down. Instead, one must use the primitives provided by the technology. This is the point where design and analysis meet. The second-level reduction $M''$, which was derived by bottom-up analysis, becomes a primitive to be used in top-down design.

### 5.3 Reduction strategies

The second-level reduction of figure 14 is simple enough that the product of five to ten such automata could be constructed directly with reasonable memory resources. Certain reduction strategies allow us to analyze much larger systems, however. For example, low level models generally determine the behavior of a device under certain interface requirements on the device's operating environment. Showing that a collection of components mutually satisfy each other's interface requirements cannot be accomplished by analyzing the components separately (*cf.* section 4). Instead, we use a hierarchy of reductions. Small groups of components are collected into modules. A reduction of each module is produced, whose requirements are relative only to the interface signals of the module. These reductions in turn are combined into larger modules, until a level is reached where the global product can be constructed, and it can be proved that all the high-level requirements are met. As an example, most circuit components require that no ambiguous signals occur at their inputs (at least at certain times). A reduction hierarchy might be used to prove ambiguous signals never occur in the system.

On the other hand, localized tasks often can be proved using an unbalanced reduction hierarchy. In this strategy, a detailed model of one part of the system is used, while the rest of the system is greatly abstracted. As an example, consider a system containing the interlock element, in which we wish to prove that collisions in the use of a resource such as a memory never occur. The two clients of the resource can be greatly abstracted in the analysis, since the operations they perform other than requesting and accessing the resource are not relevant to the task. Thus a simple four-state model (idle, requesting, accessing, releasing) probably would be sufficient regardless of the complexity of the function performed by the clients.

## 6 Summary and Conclusions

In the introduction, we stated two criteria important for a successful formal verification method for switching circuits: the accuracy with which real circuit behavior is modeled, and the tractability of the decision algorithms. With regard to the first criterion, we have demonstrated that formal verification methods can be extended into the domain of analog transistor circuit models such as the ODE models used in circuit simulation. As to the second criterion, we have proposed that a hierarchy of formal abstractions be used as a means of reducing the complexity of analysis to tractable levels. Homomorphic reduction forms the mathematical basis of both of these methods.

## References


[BCDM85] M. Browne, E. Clarke, D. Dill, and B. Mishra. Automatic verification of sequential circuits using temproal logic. In *Conf. on Hardware Description Languages, (Tokyo)*, 1985.

[Boc82] G. V. Bochmann. Hardware specification with temporal logic: An example. *IEEE Trans. Comput.*, 31:223–231, 1982.

[Bry80] R. E. Bryant. An algorithm for mos logic simulation. *Lambda*, 1:46–53, 1980.

[Cho74] Y. Choueka. Theories of automata on $\omega$-tapes: A simplified approach. *J. Comp. Sys. Sci.*, 8:117–141, 1974.

[CM73] T. J. Chaney and C. E. Molnar. Anomalous behavior of synchronizer and arbiter circuits. *IEEE Trans. Comput.*, 22:421–422, 1973.

[CW75] G. R. Couranz and D. F. Wann. Theoretical and experimental behavior of synchronizers operating in the metastable region. *IEEE Trans. Comput.*, 24:604–616, 1975.

[DC85] D. L. Dill and E. M. Clarke. Automatic verification of asynchronous circuits using temporal logic. In *Chapel Hill Conf. on VLSI*. Computer Sci. Press, 1985.

[Fos81] M. J. Foster. Syntax-directed verification of circuit function. In H. T. Kung et al., editor, *VLSI Systems and Computations*, pages 203–212. Computer Science Press, 1981.

[FST84] A. Fusaoka, H. Seki, and K. Takahashi. Description and reasoning of vlsi circuit in temporal logic. *New Generation Computing*, 2:79–90, 1984.

[HK88] Z. Har'El and R. P. Kurshan. Software for analysis of coordination. In *Proc. Internat. Conf. Syst. Sci.*, pages 382–385. Pergamon, 1988.

[Kur87] R. P. Kurshan. Reducibility in analysis of coordination. In *LNCIS*, volume 103, pages 19–39. Springer-Verlag, 1987.

[Kur90] R. P. Kurshan. Analysis of discrete event coordination. In *LNCS*, volume 430, pages 414–453. Springer-Verlag, 1990.

[MC80] C. A. Mead and L. Conway. *Introduction to VLSI Systems*. Addison-Wesley, 1980.

[MO81] Y. Malachi and S. S. Owicki. Temporal specifications of self-timed systems. In H. T. Kung et al., editor, *VLSI Systems and Computations*, pages 203–212. Computer Science Press, 1981.

[Nag75] A. W. Nagel. *SPICE 2, A Computer Program to Simulate Semiconductor Circuits*. Univ. Calif., Berkeley, 1975.

[Rue81] A. E. Ruehli. Survey of analysis, simulation and modelling for large scale logic circuits. In *18th IEEE Design Automation Conf.*, pages 124–129, 1981.

[Sei80] C. L. Seitz. System timing. In Carver Mead and Lynn Conway, editors, *Introduction to VLSI Systems*, pages 218–262. Addison-Wesley, 1980.

[Sho83] R. E. Shostak. Formal verification of circuit designs. In T. Uehara and M. Barbacci, editors, *Computer Hardware Description Languages and Their Applications*, pages 13–30. North-Holland, 1983.