Toward the Rigorous Use of Diagrams in Reasoning about Hardware

Steven D. Johnson
sjohnson@indiana.edu
Department of Computer Science
Indiana University

Jon Barwise
barwise@indiana.edu
Departments of Philosophy, Mathematics, and Computer Science
Indiana University

Gerard T. Allwein
Visual Inference Laboratory
Indiana University

Indiana University Logic Group
Preprint No. IULG-93-23
May 1993

Abstract

We propose that the concept of "formal methods" be expansive enough to include a rigorous, integrated use of diagrams for automated reasoning tasks. Applications to hardware design provide an especially rich domain to explore visually oriented extensions of traditional logic. This domain has an established methodology embracing the use of diagrams; it promises to light the way toward modernizing the foundations of formal logic. In this paper we examine some of the issues exposed when one attempts to develop a rigorous basis for heterogeneous and visually oriented reasoning. .

Toward the Rigorous Use of Diagrams in Reasoning about Hardware

Steven D. Johnson*, Jon Barwise, and Gerard T. Allwein.

*Johnson's research supported, in part, by the National Science Foundation under grants numbered MIP89-21842 and MIP92-08745.

Introduction

The logician's conventional notion of proof has grown increasingly anachronistic through the twentieth century as computing capabilities have advanced. Classical proof theory provides a partial model of actual mathematical reasoning. When we move away from mathematics toward reasoning in engineering and computation, its limitations are even more pronounced. The standard idea of a formal system seems frozen in the information technology of Frege's time; it is decidedly quaint in the presence of today's desk-top computer.

Contrary to formalists' dogma, experience suggests that pictures, graphs, charts, and diagrams are important tools in human reasoning, not mere illustrations as traditional logic would have us believe. Nor is the computer merely an optimized Turing machine. The computer's graphical capabilities have advanced to the point that diagrams can be manipulated in sophisticated ways, and it is time to exploit this capability in the analysis of reasoning, and in the design of new reasoning aids.

In this paper we propose a new understanding of the role of various sorts of diagrams in the specification and design of computational hardware. This proposal stems from a larger project, initiated by Barwise and Etchemendy [3], the goals of which are to develop a mathematical basis from which to understand the substantive logical relationships between diagrams and sentences, and to develop a new generation of automated reasoning tools from that basis.

Microelectronic CAD systems are among the supreme examples of visualized reasoning environments. Their tools are highly oriented toward diagrams, are quite sophisticated, and are comparatively well integrated. These systems also integrate logical and physical design, providing a strong coherence between specification and implementation views. Formalized reasoning meshes poorly with these working frameworks. Although it provides needed rigor for today's highly complex design challenges, its preoccupation with formulas at the expense of diagrams is simply too cumbersome. We should attempt to draw lessions from these advanced design environments, making the reasoning rigorous without subverting their character.

This paper is built around two simple design examples: a synchronizing circuit and the "Mead-Conway" traffic-light controller. Our purposes are, first, to illustrate heterogeneous use of pictoral "formalisms" in design, and second, to expose basic questions for the logical analysis that follows. Between the examples, we will develop a common mathematical basis in which the examples can be analyzed. These are admittedly modest beginnings, but we hope that they start to put to rest the idea that only formulas can be used in formal reasoning.

The CircuitProof project

The foundations of the research described here arose from previous work in information theoretic aspects of logic: *Turing's World*, a program for building and running Turing machines using only graphical representations for them [2], and *Tarski's World*, a program to teach the language of first-order logic [4]. The success of these programs shows the power of graphical designs in reasoning in logic, but it also raises a host of new theoretical issues and possibilities.

Exploring these has led to the development of *Hyperproof*, [1] a specialized heterogeneous reasoning system developed under the guidance of Barwise and Etchemendy by Allwein, Greaves, and Lenz [1, 5]. This program is now being used at Stanford, Notre Dame, and Indiana to teach basic reasoning skills to students in a way that makes crucial use of visualization of information.

Hyperproof is a formalism for use in reasoning about blocks worlds by using a combination of diagrams and sentences of first-order logic. Figure shows a *Hyperproof* session. In it, partial knowledge is used to discover facts about a situation. There is significant interplay between the information system represented by the picture and the proof manager. For instance, information in the picture is used to determine whether case analyses are exhaustive; in fact, cases themselves may be represented by hypothetical board configurations.

Hyperproof is very specialized and could not be applied to hardware description. However, there are some important lessons to be learned from *Hyperproof*, lessons which have informed the project initiated here, and which we summarize very briefly now.

The first lesson—which is hardly new but seems to be ignored in the standard logical formalisms—is that finding the best representation of a problem is often the most important step in solving the problem. The reason engineers use various sorts of diagrams is because they are highly efficacious representational schemes. The second lesson learned from *Hyperproof* is to explore "heterogeneous" systems, systems in which two or more different types of representation are used side by side. Typically different formalisms will be better for talking about different aspects of the problem or device under con-

[1] *Hyperproof* is a joint project of the Visual Inference Lab at Indiana University and CSLI at Stanford University.

Figure 1: A Hyperproof display

sideration. As long as there is a common subject matter, this subject matter will tie the various formalisms together, giving us a notion of valid inference. Finally, *Hyperproof* shows that we need a "hyperstructure," the place were proofs are given using a mixture of the various formalisms.

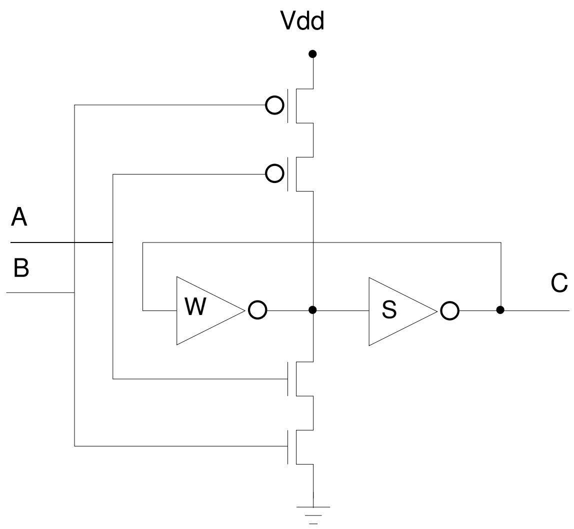
CircuitProof is a pilot study with three goals. The first is to examine engineering practice in order to develop a suite of integrated reasoning tools, some based on sentences and some based on diagrams, which are useful in circuit-design applications. In this paper we will configure such a suite containing finite state diagrams, circuit schematics, timing diagrams, as well as second order logic. The second goal is to engage research in the formal-methods area, where there is an abundance of visually oriented design tools for logicians to study. Finally, we want to inform the engineering community that automated reasoning of the kind that is entailed in hardware design is not *necessarily* textual. Logicians bear as much responsibility as tool developers to bridge the gap between practice and formalism.

Diagrams and hardware description

It is hard to write, talk, or think about circuitry without using diagrams. Virtually all articles about formalizing hardware description make heavy, though usually intuitive, use of schematics, physical drawings, timing diagrams, state machines, and so on. It is so common that we may regard it as vital. Structural qualities of the circuits may justify the use of diagrams, but the correlation between a physical circuit and its schematic is difficult to pin down. Often, these differ dramatically.

There must be other reasons for the ubiquitous use of diagrams. As in many realms of mathematical discourse, the interaction of a narrative argument and a circuit diagram yields an efficient mode of explanation. The following excerpt from [7] is typical:

"... When $A = B$ either the N-types will be turned on or the P-types. They will clamp the input to the strong (S) inverter, overpower the weak (W) inverter, and the output will change to the proper level. When A and B are different neither a pull-down or a pull-up path will exist and the previous output will be maintained by the latch formed by the W and and S inverters. ...



A conventional formalization of this argument represents the diagram as a sentence. In predicate logic for example, one would employ relations for devices, variables for wires, and conjunction for composition [8]. The narrative sketches a proof, "this circuit is a C-element," listing relevant facts that can be extraced from such a formula and which, taken together, should imply the behavioral properties of a C-element. This regimen for verification raises several questions, but foremost is its denial at the outset that the diagram itself is serving any purpose. We can see that it serves at least as a referent to the narrative. Once everything is distilled to sentences it is hard to see how to recover this manner of reference. Discovered properties like "pull-down path" and "latch" emerge as syntactic qualities of the diagram, but would otherwise have to be indirectly developed in the course of a formal deduction.

Hunt offers a radical statement of purist objectives [12]:

> "We envision providing a mathematical statement which we call a *formula manual* [Hunt's italics], that completely specifies the operation of a hardware component. With respect to digital systems, we want to: [1] Completely replace programmer's manuals, timing diagrams, interface specifications, power requirements, tec. with clear precise formulas. [2] Provide a perfectly clear foundation upon which systems can be built."

Surely this is not the world to which we really aspire, for while the programmer's manual may convey information imperfectly, it conveys that imperfect information rather efficiently in human terms. Hunt is justifiably calling for greater rigor in the description of hardware, but his formula manual, as described, would be of marginal benefit to the human user.

Cohn poses the central problem in her summary of the VIPER verification project [6]:

> "The first task in the [block level] verification effort is to derive a functional expression of the block model in a formal logic which is suitable for reasoning and proof. This is necessary because it is difficult to reason formally about a schematic diagram indicating [information flow]. It is possible to imagine doing this by reasoning about sequences of annotated pictures, but the real problem is not so much the obvious awkwardness of such a method as the lack of a formal semantics of pictures. ..."

A reasoning system based purely on pictures would, of course, be awkward, but on the other hand, so is purely textual reasoning, judging from the experience gained to date. Given the human tendencies for using both forms, it is worth investigating a middle ground.

A notable example of work connecting diagramatic representation with formal proof is the *LAMBDA* system of Fourman and others [9]. A schematic-entry facility called *DIALOG* associates graphical symbols with HOL proof tactics, which are invoked as new components are connected. These tactics generate proof obligations, which may be resolved by either graphical or textual interactions with the system. We think the claim that such tools make "[formal methods] accessible to design engineers" is a valid one. For this reason, serious attention should be paid by logicians—and the implementers of automated reasoning systems—to what have heretofore been regarded as mere human-interface issues.

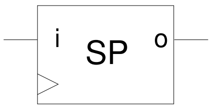
In the remainder of this paper we examine some of the questions that arise when diagrams are used in hardware design. We cannot offer comprehensive answers to any of these questions, nor yet a system to implement these paradigms. However, we strongly believe that the formal-methods community should engage itself with questions of this nature, in order to help establish a new foundation for 21st century logic.

The Unit-Pulser Example

To illustrate a perspective on design, let us consider a simple example, drawn from the textbook by Winkel and Prosser [14]. We will present the example as it might be developed in an actual design class, with all the diagrams one would naturally use. Our claim is relatively simple: *properly understood, this design process is a logically rigorous piece of reasoning it as it stands.* There is no need to (and every reason not to) reduce the diagrams used in the reasoning into some more standard formal system.

In this paper, we assume a globally clocked, synchronous design technology.

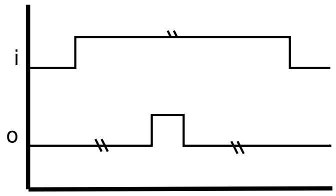
Suppose one wants to design a device with a single one-bit input and a single one-bit output:



SP's external behavior is stated informally as follows:

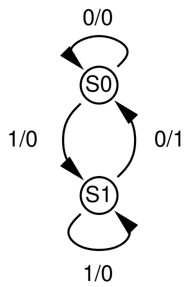
"SP emits a unit-pulse on o for each pulse received on i."

This specification, of course, leaves many things open, things which would have to be settled in the design of an actual device. The timing diagram given below provides a somewhat more exacting specification of SP's behavior by requiring that the output pulse occur in the neighborhood of its corresponding input pulse.



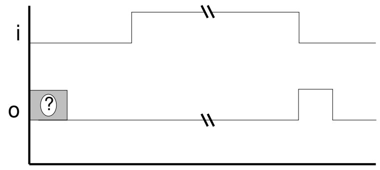
This specification is still fairly open to interpretation. Any simple implementation in hardware would determine where in the interval to generate the output pulse.

Let us think of this timing diagram as a "requirements specification." The state-machine diagram below can be thought of as a design specification:



Using a standad interpretation as an ω-automaton of some kind a device is described with two states and a specific input/output behavior in each of these states.

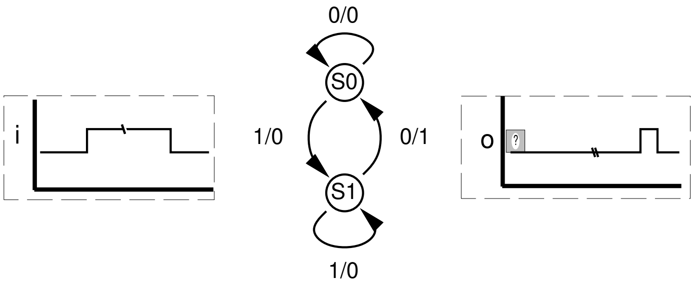
Our first observation is that any device described by this Mealy machine diagram has an input-out behavior that satisfies the requirement specification. Indeed, we can say more, since it will have a behavior that is described by the following timing diagram.



At a purely logical level, this can be made perfectly rigorous, every bit as rigorous as the fact that $\mathsf{A}(\mathsf{f}(3))$ follows from $\forall \mathsf{x} \mathsf{A}(\mathsf{x})$, as we will show. This diagram reflects an undetermined choice of the starting state for the machine: we cannot know whether a pulse is issued at "time zero." But given the condition that input i is initially zero, the machine settles into a desirable behavior, generating its output pulse at the end of the input pulse.

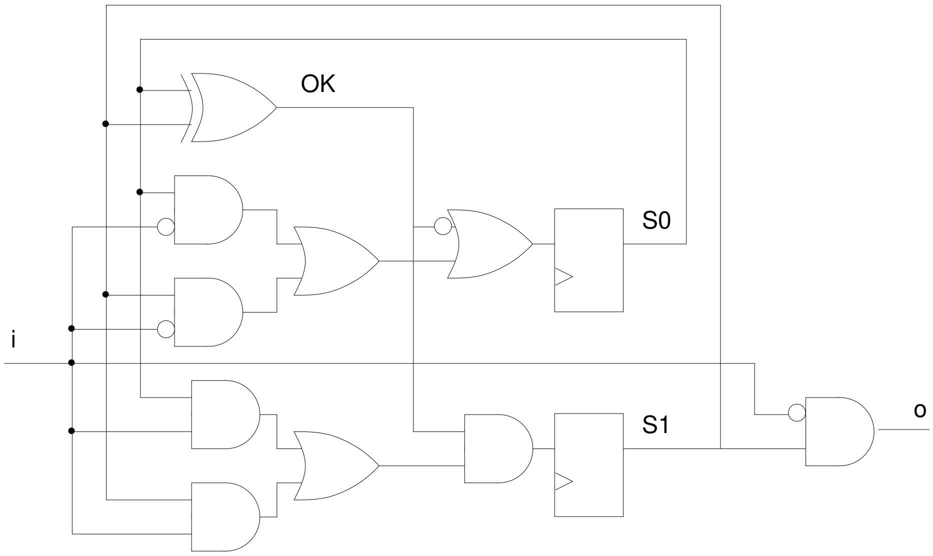
Whether this output pulse lies within the "neighborhood" specified by our earlier timing diagram can only be settled by giving a rigorous semantics; including a precise meaning to the "ellipsis" symbols in the waveforms.

So much for the purely logical aspects of this step of the exercise. Let us look ahead to a future where there is computational support for such reasoning. Perhaps we can build a simulator/verifier to certify the following relationship:

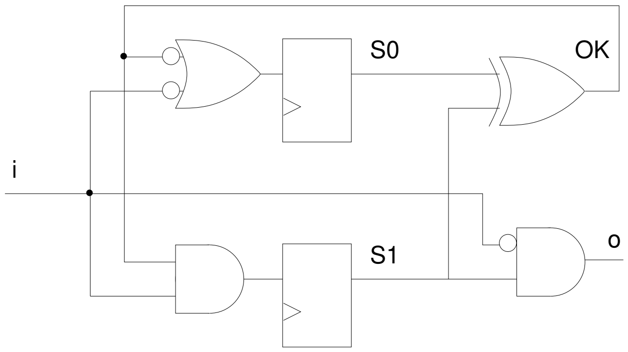


This is plausible given recent developments in symbolic simulation and inductive theorem proving. It should be at least as practical as automated proofs by induction, since there is a very simple instance of induction to verify that a simulation relationship holds.

Moving on in the design process, let us construct a circuit to implement the state machine. The circuit diagram below is obtained by a (well-known, but see Appendix A) general construction based on a "one-hot" representation of control. Briefly, in a one-hot controller there is one d-flipflop for each state of the machine.

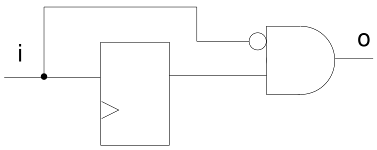


The signal OK (which in this case should be viewed as an artifact of the formalization discussed later) assures that the circuit is always in a valid state: having exactly one of S0 and S1 asserted at any time. Under the assumption OK = 1, which must obtain except possibly at "time 0," the circuit can be simplified a great deal. First, we can simplify the selection of next-state inputs part of the diagram to:

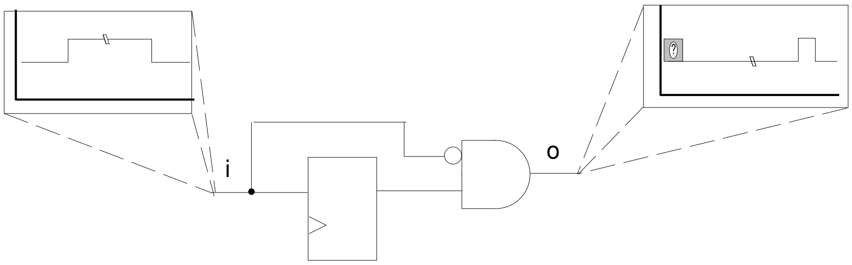


We have obtained this circuit with equivalence preserving transformations, subject to valid side-conditions; and since the initial circuit has the input-output behavior specified by the Mealy-machine diagram, this derivation is a proof of implementation.

Now observe that if signal OK is invariantly equal to 1, then the two gates feeding the registers aren't needed, and if those gates aren't needed, neither is the signal OK; and if OK isn't needed, neither is S0. This suggests looking at a possible optimization:



This reduction is not always valid but it is justified in this case because condition OK is equivalent to $\mathsf{S0} = \overline{\mathsf{S1}}$. Rather than working out a proof of this claim, let us instead simply verify the circuit by checking that it, too, conforms to the state-machine. This might be accomplished, as before, by some form of simulation:



Analysis

This method of approaching design is, we believe, reflective of practice. Our main point is that, properly understood, it is every bit as logically rigorous as any other form of proof. Furthermore, it is computationally tractable, at least for this small example.

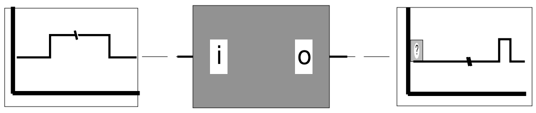
It is clear that the diagrams are integral to this particular solution. But are English sentences integral or eliminable? In this example, the English discourse is little more than a narration of the design process, although it does occasionally point out features of the diagrams it threads together. It is playing the illustrative role that is traditionally attributed to the pictures. The succession of drawings contains the substance of the design, so it is these drawings that should be connected by a rigorous semantics. However, it is almost certain that an optimal suite of formalisms would also include classical formal systems, logic corresponding to some of what we have used English for in the example.

Obviously we are dealing several distinct graphical idioms, and also—although it may not be as obvious from this example—with a number of possible language dialects. These must be united in a way that secures a relationship among them. What relates them, of course, is that they are all ways of describing various aspects of the same object. That is, they have a common semantics.

One way to approach a semantics would be to insist on a translation of all our diagrams into a common-denominator formalism with a well understood semantics. Higher-order logic is often proposed for this purpose. Under this approach a timing diagram, for example, would just be a convenient encoding for this formula. An alternative is to treat all the participating formalisms as on a par. This is the approach we have chosen. Although there are a number of reasons for this decision, we will only discuss two of the more important, connected to two of our originally stated goals.

The first reason is that a reduction to a linguistic formalism unjustifiably elevates sentential calculus to a preferred status. Nothing forces one to do this, and doing it violates the basic engineering practice we seek to understand. This practice uses a system in which textual formalisms interact with diagramtic formalisms as peers. We want our theoretical development to reflect this balance. Although a proper formula might specify the same relation, we do not think it shares all the inferential properties as a diagram.

The second reason has to do with our objective for diagram-based design tools. We do not regard formal systems—as they are currently understood—to be sufficiently powerful engines to drive automated reasoning. A lesson learned from *Hyperproof* is that systems that treat diagrams and sentences on a par, admitting rules of inference that explicitly relate the two, are more natural and powerful than either alone. Purely syntactic rules of inference are not enough to prove things proficiently. For example, it would be unreasonable to perform the generalized simulation



as a purely textual deduction.

A mathematical basis

Let us develop a possible basis to support these preliminary explorations. We do not prescribe a model and, in fact, believe that a practical system might have to support a variety of models for different levels of description. However, we do ground our semantics in a particular domain, giving a basis for relating the various visual idioms. The formalization in this section develops toward a globally clocked, synchronous implementation technology, and involves other simplifying assumptions along the way.

To reason about any device, we need at least to know how it is connected to its environment. All of our diagrams implicitly or explicitly refer to this external view, or *signature*. The Single-pulser had external ports I and O, and sometimes there was and an internal signal X, too.

Definition 1 A *signature* is a finite set $\Sigma = \{N_1, \ldots, N_n\}$ of *names*.

For the remainder of this section, let Σ be a fixed general signature,

$$\Sigma = \{N_1, \ldots, N_n\}$$

For generality we will develop our semantics in ways that do not impose a direction on information flow. However, directionality is an important quality, so we shall introduce terminology for it along the way. When directionality is considered, a given signature will be composed of two disjoint, possibly empty, subsets: Σ^{inp} of *inputs*, and Σ^{out} of *outputs*.

$$\Sigma = \{ \overbrace{I_1, \ldots, I_p}^{\Sigma^{\text{inp}}}; \overbrace{O_1, \ldots, O_q}^{\Sigma^{\text{out}}} \}$$

The partitioning could involve other attributes, such as "internal" and "bidirectional," but these two are enough for our present discussions.

For now assume that each signal can hold only a 1 or a 0 at any time. Let us represent the contents of a signal as a function from Z to $\{0, 1\}$, Z being the set of (both positive and negative) integers. Such a function is called a *simple sequence*. We allow time to flow infinitely in two directions in order to avoid talking about initialization, resets, and so forth.

Definition 2 Let $\Sigma = \{N_1, \ldots, N_n\}$ A Σ-*sequence* is a collection of simple sequences $\{F_{N_1}, \ldots, F_{N_n}\}$. A Σ-*profile* is a set of Σ-sequences.

A Σ-sequence represents a single, infinite run of some device, $F_{N_i}(t)$ being the value on signal named N_i at time t. If the context permits it we will use the same variable for both the name of a signal and the function associated with that name, and so write $N_i(t)$. Intuitively, a given device will admit certain sequences, not others, and so determine a unique profile.

We frequently need to define an association, or *binding*, between names in Σ and values in some other set.

Definition 3 A Σ-*binding in S* is a mapping from Σ to a set of values, S.

Thus, a Σ-sequence is just a Σ-binding to the class of simple sequences over $\{1, 0\}$.

In many of the defintions that follow, bindings are represented by a string of values corresponding to some implicit ordering of Σ. If Σ is given as $\{N_1, \ldots, N_n\}$, then the string $V_1 \cdots V_n$ binds each N_i to V_i.

A partial ordering on signatures can be based on set containment, and extends in the usual way to sequences and profiles.

Definition 4 A subset $\Sigma \subseteq \Sigma'$ is called a *reduct of* Σ'. A Σ-sequence W is a *reduct* of a Σ'-sequence W' iff the functions in W are identical to those in W' to which they correspond by name. Finally, a class P of Σ-sequences is the *projection* (onto Σ) of a class P' of Σ'-sequences if P is the set of all reducts of sequences in P. For P we write $P'|\Sigma$.

We will require that projections preserve directionality; so if Σ' is a reduct of Σ, we must also have $\Sigma'^{\mathrm{inp}} \subseteq \Sigma^{\mathrm{inp}}$, and $\Sigma'^{\mathrm{out}} \subseteq \Sigma^{\mathrm{out}}$.

Σ-profiles will provide the semantic domain for assigning meaning to diagrams. If diagrams D_i have signatures Σ_i and profiles $\mathcal{D}_i$, then D_1 is *consistent* with D_2 if $\mathcal{D}_1 \cap \mathcal{D}_2 \neq \emptyset$. D_1 *entails* D_2 iff $\mathcal{D}_1 \subseteq \mathcal{D}_2$. D_1 is *logically equivalent* to D_2 iff $\mathcal{D}_1 = \mathcal{D}_2$. A transformation from one diagram to another is *universally valid* iff the two diagrams are logically equivalent.

None of the logical qualities above captures the notion of *implementation*. As a first attempt, we might say that a diagram D_1 is an implementation of Σ-diagram D_2 if $\mathcal{D}_2 = \mathcal{D}_1|\Sigma_2$. This is too strong if we want to say, for example, that a particular unit-pulser circuit implements the unit-pulser specification, since there would be sequences that fit the specification but which would not be accepted by the circuit.

Alternatively, we might say that $\mathcal{D}_1|\Sigma_2 \subseteq \mathcal{D}_2$, corresponding to the standard representation of "implements" by "implies" in formal methods. But as Weise [13] and many others have explained, this is too weak, since, for example, the empty set becomes an implementation profile of any specification.

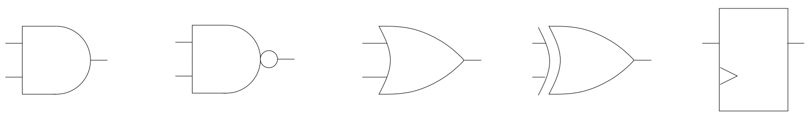
"Implementation" involves both entailment and consistency. In the directional case, we can say that any input accepted by the specification must also be accepted by the implementation. Formally, D_1 implements D_2 if Σ_2 is a reduct of Σ_1, $\mathcal{D}_1|\Sigma_2 \subseteq \mathcal{D}_2$, and for all σ in $\mathcal{D}_2$ there exists a σ' in $\mathcal{D}_1$ such that

$$\sigma'|\Sigma_2^{\mathrm{inp}} = \sigma|\Sigma_2^{\mathrm{inp}}$$

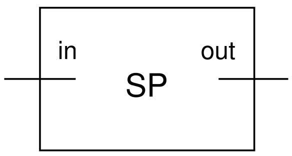
In the following sections we develop what might be called an abstract syntax for the various forms of diagrams we have seen earlier. We are not specifically concerned with how diagrams are rendered but with their underlying information structures. However, it should of course be possible to render diagrams without too much human intervention.

Circuit diagrams

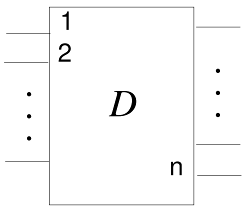
We are given a number of icon types, D, each having an arity, $\ell(D) > 0$. For each icon type there is an infinite set of icon *tokens*, each associated with a signature Σ_D of size $\ell(D)$. Examples of tokens include the gates and storage elements seen in the previous section:



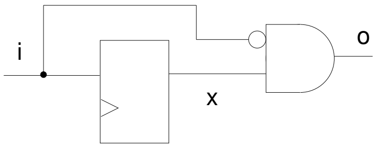
as well as higher level symbols, such as



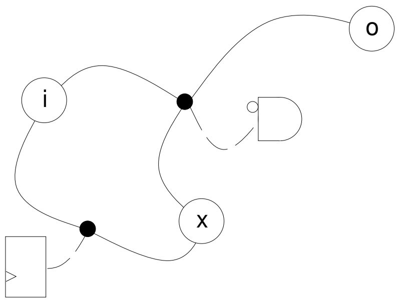
Each token must have a signature, whether or not it is revealed by its graphics. Where necessary, we will let the set $\{1, 2, \ldots, n\}$ stand for the generic signature of arity $n = \ell(D)$. A generic token of type D could be drawn



Circuit diagrams are represented as hypergraphs in which the nodes stand for connections and the hyperedges are associated with components. For example our unit-pulser diagram,



would have node set $\{\mathsf{i}, \mathsf{x}, \mathsf{o}\}$, with two hyperedges: one for the **d-flipflop** connecting i and x; and the other for the **and**-gate connecting i, x, and o:



Definition 5 A Σ-*circuit graph* is a hypergraph structure $\langle \Sigma, E, R \rangle$, where E is a set of icon tokens, and the relation $R \subseteq E \times \Sigma^*$ respects arities. That is, if $(e, N_1 N_2 \cdots N_n) \in R$ and e is a token of type D, then $\ell(D) = n$.

Our semantics will interpret the string $N_1 N_2 \cdots N_n$ as a Σ_D-binding (Defintion 3). Intuitively, we want to associate a circuit graph with the set of Σ sequences that is simultaneously determined by its components. For example, define the *and-profile* to be the set of $\{A, B; Z\}$-sequences in which $Z(t) = A(t) \cdot B(t)$ for all t; and similarly for other Boolean gates (Recall that we are using the same variable for the name and its function). A storage element with signature $\{D;\ Q\}$ has a profile whose sequences satisfy $Q(t+1) = D(t)$.

Composite profiles for circuit graphs are built from the families of profiles associated with the graph's hyperedges. Thus, a consistent family of sequences for our unit-pulser, shown above, is $\{I; X, O\}$ where for all $t \in Z$, $X(t+1) = I(t)$ and $O(t) = X(t) \cdot \overline{I(t)}$. The set of all such $\{I, X, O\}$-sequences would comprise a "fine-grained" profile for the circuit; and the $\{I, O\}$-projection would be a "course grained" profile, which hides internal signals.

For device type D let $\Sigma_D = \{1,\ 2,\ \ldots,\ \ell(D)\}$ represent the class of all signatures that can be affiliated with D. The semantics assigns as given a Σ_D*-profile*, written *Profile*(D).

Definition 6 Let $\Sigma = \{N_1,\ \ldots,\ N_n\}$ and let G be a Σ-circuit graph. A Σ-sequence $\{F_1,\ \ldots,\ F_n\}$ is *consistent* with G if whenever G contains an edge $(e, N_{e_1} \cdots N_{e_k})$, in which e has device type D, then the Σ_D-sequence

$$\{F_{e_1},\ \ldots,\ F_{e_k}\} \in \mathit{Profile}(D)$$

The G*-profile* is the collection of all Σ-sequences that are consistent with G.

By calling $\{F_{e_1},\ \ldots,\ F_{e_k}\}$ a Σ_D-binding we affiliate each F_{e_j} with j in the generic signature. In other words, the connection of component e to the signal nodes N_{e_j} is fully determined.

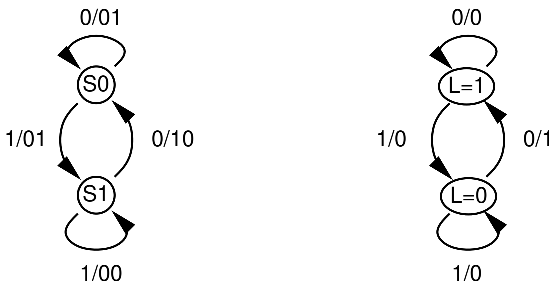
When directionality is considered, we would like circuit-graphs to act as functions from (the individual sequences associated with) inputs to outputs. We can develop this notion by generalizing to a *directional* Σ-circuit graph in which icons have fixed input and output arities and these are respected by the relation R of Definition 5. The storage elements of a digital circuit induce a directionality with respect to time. To be *properly formed*, several structural conditions would have to be met. Initial nodes come from Σ^{in}; terminal nodes must come from Σ^{out} (but outputs may participate in cycles and hence not be terminal); in R, each node is associated with at most one component output; and every cycle includes an edge associated with a storage-element icon.

A standard result—though not a trivial one to prove considering these structural conditions—is that a directional circuit graph is *functional*: that is, it's profile represents a well-defined, total function from its input signals to its output signals, as desired.

Proposition 1 *If directed Σ-circuit graph G is properly formed and if components are functional, then so is G.*

State diagrams

For behavioral specification we will use finite-state automata (FSAs, or "state machines") whose inputs and outputs fit some particular signature. Our diagrams will associate some outputs with machine states ("Moore outputs") and others with state transitions ("Mealy outputs"), although our formalization will include only the latter. For example, the Unit-pulser state machine in the previous section had only Mealy outputs, but suppose we add a light to indicate when the machine is waiting for a pulse. We might specify this enhanced single pulser in one of two ways, the difference being a matter of notation.



The diagrams need an explicit association to tell which values go with which terminals. In this example the state labels hold the value of the light, L, and the edge labels hold the values of I and O. We can eliminate the state labels by adding them to all edges leaving the state, as shown in the diagram on the left above. Let us adopt this simplification in defining state machines.

Definition 7 A *state-machine graph* for signature $\Sigma = \{N_1, \ldots, N_n\}$ is a directed graph, with edges labeled by Σ-bindings to $\{0, 1\}$.

For example, the state machine shown to the left above refers to signature $\{I;\ O,\ L\}$, and the $\{I;\ O,\ L\}$-binding associates the first digit of each label with input I, the second with output O, and the third with output L.

The notion a machine M accepting a Σ-sequence is simply that a series of states exists whose transition labels agree with the sequence. For example, a $\{I;\ O,\ L\}$-sequence is accepted by the machine shown above provided the following conditions are satisfied:

1. $O(t) = 1$ iff $I(t-1) = 1$ and $I(t) = 0$; that is, the machine is in state S1 and detects a 0 on I.
2. For all $t \in Z$, if $I(t) = 0$ then $L(t+1) = 1$

These are simply the sequences that accept complete (infinite in both directions) runs of the machine. In general, Given any Σ state-machine M, we associate a Σ-profile with it as follows.

Definition 8 A Σ-sequence $\{N_1, \ldots, N_n\}$ is *accepted* by state machine M provided there is a sequence of states s_i, for $i \in Z$, such that for each i, there is a unique edge from s_i labeled by the string $N_1(i) \cdots N_n(i)$ and leading to s_{i+1}. By the *profile* of a machine M we mean the set of all Σ-sequences accepted by M.

We want machines which at all times respond to all possible input stimuli. Assuming $\Sigma^{\text{inp}} = \{I_1, \ldots, I_k\}$, the state-machine graph is said to be *reactive* if for every state s and string $b_1 \cdots b_k$, there is an edge whose label maps I_j to b_j for $i \leq j \leq k$.

Proposition 2 *Deterministic reactive machines are functional.*

Timing diagrams

In contrast to the circuits and state machines we have looked at so far, a semantics for timing diagrams seems more subtle. One reason for this subtlety may be that the diagrams are partial descriptions, while, typically, circuit and machine diagrams fully determine the objects they describe. Another reason may be the context-sensitive quality of these kinds of diagrams, which often express controlling relationships among events.

We begin with the notion that a conventional timing diagram is really a collection of diagrams which happen to share a common reference to the time line.

Definition 9 *A* time scale $\delta = \langle V, \preceq, s \rangle$ *consists of a set* V *of* time variables, *a linear ordering* $\preceq$ *of* V*, and a partial successor function* s *on* V*. Whenever* $\mathsf{t} \in \mathrm{dom}(s)$ *then* $s(\mathsf{t})$ *must be the immediate successor of* t *in the ordering* $\preceq$ *(but the converse is not required).*

We sometimes refer to the time variables as "ticks" on the time line, since this is the way they are rendered in simple diagrams. Intuitively, the ticks stand for arbitrary times as constrained by the ordering and the successor function. Temporal orderings assigned to V must preserve $\preceq$; and in addition, the successor function constrains two consecutive times to be just one unit apart (Recall that we are modeling time with the set of integers, Z). These notions are formalized in the next definition.

Definition 10 Let $\delta = \langle V, \preceq, s \rangle$ be a time scale. A *timing assignment* for δ is a function $\tau : V \to Z$ satisfying the following two conditions:

1. τ is strictly order preserving: $\mathsf{t} \preceq \mathsf{t}'$ implies $\tau(\mathsf{t}) \leq \tau(\mathsf{t}')$,
2. for all $\mathsf{t} \in \mathrm{dom}(s)$ $\tau(s(\mathsf{t})) = \tau(\mathsf{t}) + 1$.

Definition 11 A *wave form* consists of a time scale δ together with a partial function ν from V_δ into the values $\{0, 1, \phi\}$. A timing assignment for a wave form is just a timing assignment for its time scale.

If $\nu(\mathsf{t}) = 1$, this represents a value of 1 on the wave form over the interval starting at time t and ending at (just before) the next tick on the scale. Similarly for $\nu(\mathsf{t}) = 0$. We will use $\nu(\mathsf{t}) = \phi$ to mean that the value over the interval starting at tick t is undetermined. This is made precise by the following definition.

Definition 12 Let $W = \langle \delta, \nu \rangle$ be a wave form, and let τ be a timing assignment for W. A function $X \in 2^Z$ is said to *conform to* W *relative to* τ provided the following conditions hold:

1. For all ticks $\mathsf{t} \in V_\delta$ $X(\tau(\mathsf{t})) = \nu(\mathsf{t})$ unless $\nu(\mathsf{t}) = \phi$.
2. if t is a non-final tick, and t' is its immediate successor under the ordering $\preceq$, then X is constant on the the half-open interval $[\tau(\mathsf{t}), \tau(\mathsf{t}'))$.

A Σ-timing diagram will contain one waveform for each name in Σ. Σ-sequence conforms to a Σ-timing diagram if its individual sequences conform to their associated wave forms *for a particular set of timing assignments.* The subtlety of the definition lies in how valid timing assignments are chosen.

Timing diagrams contain dependence information which gives a notion of relevance or "control". It is this implicative information that allows us to specify which Σ-sequences lie in the profile of a diagram. The function R in the next definition represents the "control arrows" that one might find in a timing diagram, in that $R(\sigma)$ denotes the set of all wave forms that participate in controlling σ.

Definition 13 A Σ-*timing diagram* $D = \langle \delta, f, R \rangle$ is a time scale δ, a Σ-binding f to wave forms on δ, and a Σ-binding R to 2^Σ.

For $\sigma \in \Sigma$, we will write D_σ for the wave form $f(\sigma)$. Given a Σ-sequence X, we will write X_σ for the elementary sequence associated with the name σ. We will say that σ *depends on* σ' if $\sigma' \in R(\sigma)$.

We now extend the notion of conforming from a single wave form to an arbitrary timing diagram.

Definition 14 Let D be a timing diagram with time scale δ. A Σ-sequence X *conforms to* D provided the following holds for each $\sigma \in \Sigma$: for each timing assignment τ, if X_{σ_0} conforms to D_{σ_0} relative to τ for each σ_0 on which σ depends, then X_σ conforms to D_σ relative to τ. The *profile* of a timing diagram is the set of all Σ-sequences that conform to it.

In fact, a weaker definition may be needed in practice, based on a natural partial ordering on time scales. Given time scales δ_1, δ_2, we say that δ_2 is an *extension* of δ_1 if $V_1 \subseteq V_2$, $\preceq_1$ is the restriction of $\preceq_2$ to V_1, and s_1 is a subfunction of s_2. Write $\delta_1 \sqsubseteq \delta_2$ for this relationship, which is a partial ordering. We may say that various time scales are *compatible* if they have an upper bound in this ordering. Intuitively, this means that the constraints imposed by each are compatible with one another. If we need to build timing diagrams not only by "stacking up" wave forms, but also by concatenating them, then it may be better to allow for compatable time scales, rather than identical time scales, in Definition 13.

A Formal Language for Hardware

Our representative formal language is a version of two-sorted second order logic, with sorts *Integer* (Z, representing time) and *Boole* (B). It is composed from the following alphabet:

1. individual variables of sort Z.
2. Binary constants 0 and 1 of sort B. No constants of sort Z.
3. unary function constant '$-$' of sort $B \to B$.
4. binary function constants '$*$' and '$+$' of sort $B \times B \to B$.
5. unary signal function symbols $F, G, H, \ldots$ of sort $Z \to B$. Some of these are treated as constants, others as variables.
6. function constants $f, g, \ldots$ of sort $Z \to Z$. (All we need for our examples is the successor function.)

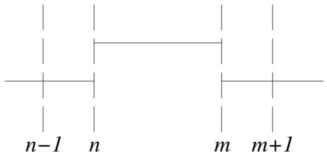
Definition 15 *Time terms* will consist of variables and expressions of form $f(t_1, \ldots, t_n)$ where $t_1 \ldots t_n$ are time terms and f is a function symbol of sort $Z \to Z$.

Definition 16 A *value term* is either a binary constant, a term of the form $F(t)$ where F is a signal function symbol and t is a time term, or a compound term involving $*$, $+$, and $-$, whose proper subterms are value terms.

Definition 17 The *atomic wffs* are of the form: $t_1 \leq t_2$ and $t_1 = t_2$, where t_1 and t_2 are time terms; and $b_1 = b_2$, where b_1 and b_2 are value terms.

Definition 18 *Wffs* include the atomic wffs and are closed under the standard connectives $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\forall x$, $\exists x$, $\forall F$, and $\exists F$, where x is a time variable and F is a function variable.

We write $\exists x \in [y, z] : \mathcal{G}$ for $\exists x : (y \leq x \wedge x \leq z) \wedge \mathcal{G}$, and similarly for $\forall$. And we will permit metalinguistic definitions. As an example, we will develop some sentences about Single Pulser. First, we capture the notion of a "pulse."

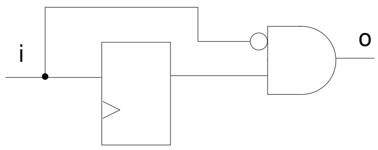


$$\begin{aligned}\mathsf{pulse}(F,n,m) \ \equiv_{\mathsf{df}} \quad & n < m \\ & \wedge\ F(n-1) = 0 \\ & \wedge\ F(m) = 0 \\ & \wedge\ \forall x \in [n, m-1] \ : \ (F(x) = 1)\end{aligned}$$

The Single-pulser specification below narrows the original English description, "For every input pulse there is an unit pulse on output," which was inadequate in many respects. Like the initial timing-diagram, it requires the output pulse to occur in the neighborhood of the input pulse. The other conjuncts exclude with spurious output pulses and infinite input pulses.

$$\begin{aligned}\mathsf{spec}(I,O) \ \equiv_{\mathsf{df}} \quad & \forall n, m \ : \ \mathsf{pulse}(I,n,m) \Rightarrow (\exists!\, k \in [n,m] \ : \ O(k) = 1) \\ & \wedge\ \forall k \ : \ O(k) = 1 \ \Rightarrow \ (\quad \mathsf{pulse}(O,k,k+1) \\ & \qquad\qquad\qquad\qquad \wedge\ (\exists n, m \ : \ n \le k \le m \ \wedge \ \mathsf{pulse}(I,n,m)))\end{aligned}$$

Of course, we can also associate a formula with the SP circuit:



$$\mathsf{imp}_1(I,O) \ \equiv_{\mathsf{df}} \ \exists X \ : \ \mathsf{delay}(I,X) \ \wedge \ \mathsf{and}^{\circ}_{\bullet}(X,I,O)$$

Expanding the component defintions yields

$$\mathsf{imp}_2(I,O) \ \equiv_{\mathsf{df}} \ \exists X \ (\forall n \ : \ I(n) = X(n+1) \ \wedge \ O(n) = -I(n) * X(n))$$

To carry out a proof of this implementation we must make the additional assumption that there have been no infinite pulses. This is a consequence of the fact that our circuit is not really a pulse detector but is instead an "edge" detector. Define

$$\begin{aligned}\mathsf{env}(I,O) \ \equiv_{\mathsf{df}} \quad & \forall t \ \exists t' < t : I(t') = 0 \\ & \wedge\ \forall t \ \exists t' > t : I(t') = 0\end{aligned}$$

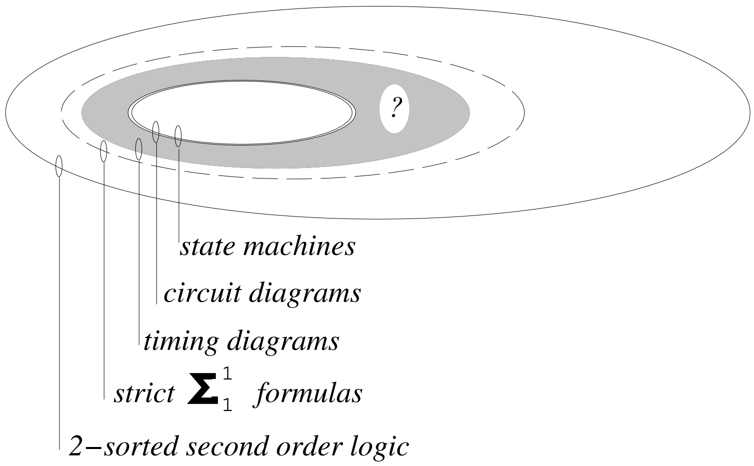


Figure 2: Relative expressiveness of the visual idioms

Our implementation theorem[2] is

$$\forall\, I,\, O \;:\; (\mathsf{env}(I,O) \wedge \mathsf{imp}_2(I,O)) \;\Rightarrow\; \mathsf{spec}(I,O)$$

A *wff* W which is closed with respect to its time variables, can be associated with a signature Σ_W, consisting of W's free signal variables. Any Σ_W-sequence provides an assignment under which W can be interpreted. Thus,

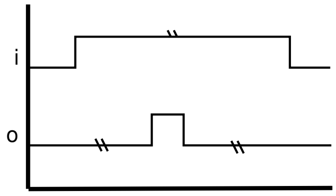
Definition 19 *The* profile *for wff W is the set of all Σ_W-sequences which make W true.*

Relating the idioms

Figure illustrates the relative expressiveness of the visual idioms we have developed so far. Since there are constructions that produce circuits from state machines and state machines from circuits these two diagramantic forms are equivalent with respect to the class of profiles they can describe.

[2] Proved by Paul Miner using *PVS*[11]

Timing diagrams are properly more expressive than the circuit diagrams and machine diagrams. For instance, our Single-pulser *specification*,



would include a device which generates its output pulse at the midpoint of every input pulse; and no circuit could realize that implementation.

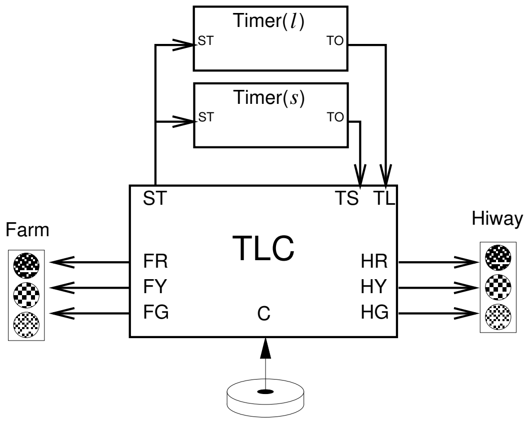
It is also apparent that the full second-order logic is properly more expressive than any of the visual forms because restricted, and properly less expressive, dialects of this language can characterize each kind of diagram. This is well known for circuit diagrams, hence also state machines. We believe that the following sublanguage encompasses timing diagrams.

Definition 20 *The* strict Σ_1^1 *formulas form the smallest collection of wffs containing all atomic and negated atomic wffs and closed under* $\wedge$, $\vee$, $\forall x$, $\exists x \in [t_1, t_2]$, $\exists F$.

Our conjecture is that every timing diagram (in our current model of these) specifies a profile that can be defined by a strict Σ_1^1 wff. However, as we shall see after the next example, these initial attempts at assigning semantics will have to be developed further. So it is premature to pursue these relationships very far.

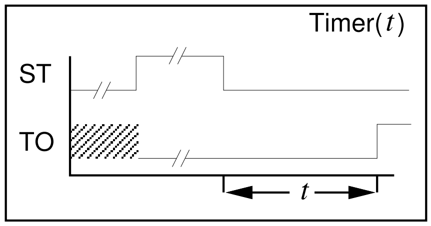
Second example: a traffic light controller

The second illustration—the Mead-Conway traffic-light controller [MEADCONWAY]—provides a sharper delineation between specification levels. In particular, the notion of verification is tied together with the notion of the design serving as a model for its "requirements specification." We start this time with the general idea of an architecture. The design problem is to produce a controller, TLC, for the system.

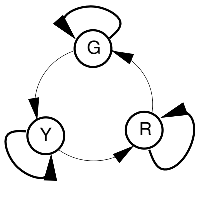


There are two traffic lights, one for a heavily traveled hiway [sic], and the other for a lightly traveled farm road. There is a sensor on the farm road, which signals the presence of a vehicle at the intersection. The hiway does not need a sensor because the default setting of its light is *green*.

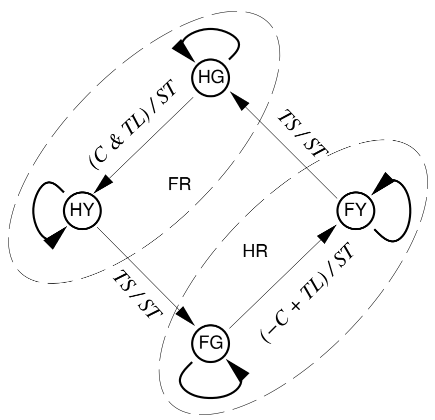
Two interval timers determine the duration of the light cycle. A Timer is parametrically specified by the timing diagram:



We make the environmental assumption that vehicles obey a light that follows the sequence: *green*, *red*, *yellow*, *green*, . . . ; so that with respect to a single light, the controller must conform to the behavior



Before going over some of the properties that that need to be verified, let us propose the following design for the controller:



The state-chart [Harel] makes it easy to see that this machine contains two instances of the single-light machine. Using a one-hot implementation method again, we could obtain a circuit schematic like the one shown in Figure below: Notice that this circuit diagram incorporate formulas to represent portions of circuitry. There are many common subexpressions and our circuit could be simplified, but this level of description is adequate for this discussion.

The circuit also implements the intended property that the one light is *red* exactly when the other is *green* or *yellow*. The saftey condition—that the lights are never both *green*—could be established by proving that under invariant that OK is always true (which is guaranteed by the construction), $\mathsf{FG} \cdot \mathsf{HG}$ is always false.[3]

$$\mathsf{OK} \Rightarrow \overline{\mathsf{FG} \cdot \mathsf{HG}}$$

[3] Although our narrative blurs the distinction between logical truth-values and boolean

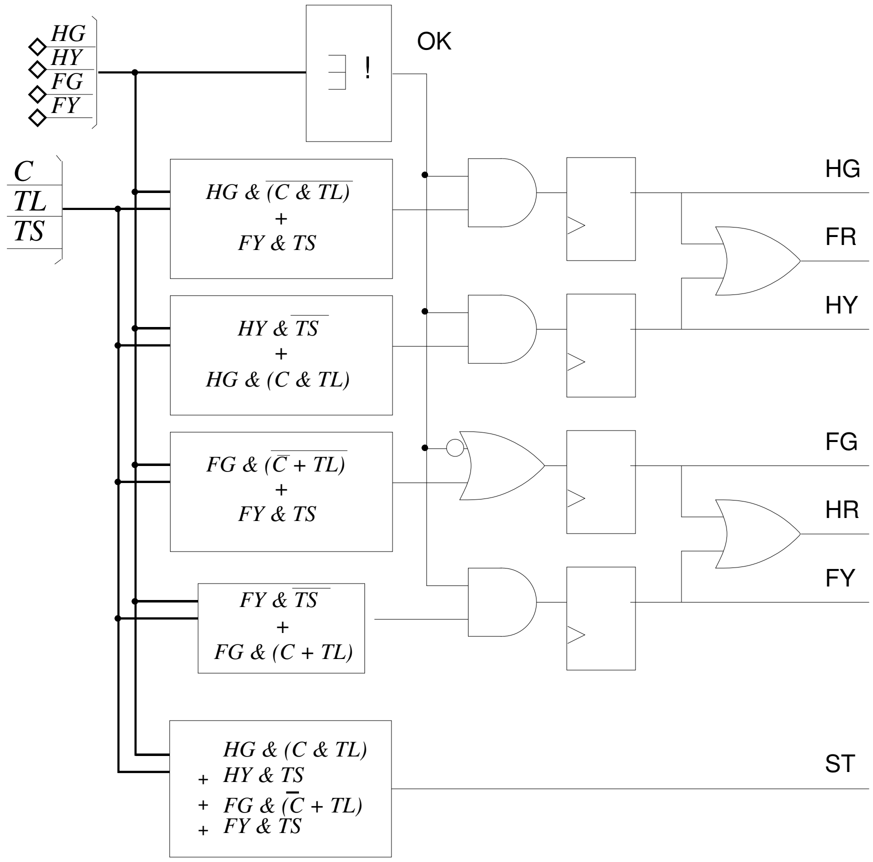
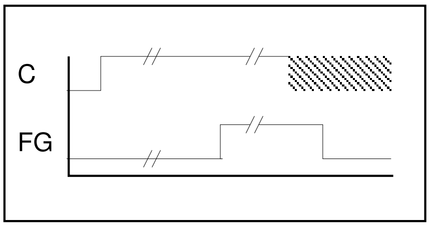


Figure 3: Circuit diagram constructed from the TLC state machine.

Of course, this can also be seen by inspection of the state-machine diagram assuming we implement the outputs properly according to that figure.

Let us now consider the liveness condition that every vehicle that reaches the intersection eventually goes through it. For the highway light, we must prove that under any conditions, HG will be true in a future state. For the farm light, we must prove



This diagram says that if a vehicle is detected on the farm road it will eventually get a green light. At least this is what it says if we associate those conditions with true values on C and FG, respectively. It is not obvious how best to approach this proof. A argument by cases, even by model checking, would have to consider not only the state of the controller, but also the initial condition and finite duration of the interval timers.

Discussion

This illustration generalizes in several ways from the Single-pulser example. Every visual idiom has been extended, as has the interaction among these forms. The extensions entail not only a refinement of semantics (which we shall not go into at this time) but also serve to point out the need for rigorously developed interconnections between the formalizations.

Extensions within an idiom included the use of higher-level symbols in circuit diagrams, the use of hyperstates in state-machine diagrams, the use of a time parameter in the timing-diagrams for the interval timers. It may be interesting to see what collection of classical formal mechanisms is needed to account for these (and other) forms of generalization used in engineering.

At least as challenging is the interplay between diagrams. Timer specifications, in the form of timing diagrams, served as components in the architectural description of the traffic light system. These diagrams provided all

bit-values, these should be regarded differently. We should really write: $(\mathsf{OK} = 1) \Rightarrow \overline{(\mathsf{FG} = 1) \wedge \mathsf{HG} = 1)}$

the information that was needed to get at the safety and liveness properties. In the controller circuit diagram we labeled component icons with formulas expressing their internal function. Some reasoning tools, such as optimizers, might just as well apply to these formulas.

The invariant safety property, $\neg(\mathsf{FG} \wedge \mathsf{HG})$, might have been proved either by a path analysis of the state machine or a static anaysis of the implementing cirucit. On the other hand, liveness seemed simpler to consider with the device of simulation. It is unlikely that the generalized notion of simulation, extended over undetermined intervals, can be fully automated. In the case of traffic light controller, the proof that a light turns green eventually reduces to a complicated case analysis, which is especially frustrating when the the conclusion is so clearly visible in the state-machine diagram.

Conclusions

We have presented two illustrations to advance the case for the use of diagrams in reasoning systems. Hardware design is a rich domain to consider these questions; our examples reveal a deeply woven interplay among visual idioms. It is relatively easy to assign a semantics to the visual forms, a semantics which ties them to a common domain of meaning as profiles. However, this is only the first step toward getting how the diagramatic reasoning systems to interact.

Our visual forms reflect three of the four principal aspects of hardware design [10]. Circuit diagrams deal with the design *structure*; State machines deal with design *behavior*; and timing diagrams deal with the aspect of *timing*, which includes not only real-time constraints but also protocol and communcation. The missing aspect is *typing*, or data abstraction, which would inevitably arise as larger examples are considered. The examples differ from conventional formal treatments in that the visual idioms, although related by a common design instance, are not required to coexist in a common linguistic framework.

Maintaining distinct views of an evolving design, and reasoning in a fashion that integrates them, are not just problems for visual-inference research. They confront any attempt to apply formalized reasoning to nontrivial design problems. An important contribution of visual-inference research is its attention to human reasoning processes, not only to the form of expression but also to underlying organization of thought.

One of the advantages of a diagram is that it is similar to what it represents along some dimensions, enough so that one can simply "read off" facts about the represented object from the representation. A diagram $\mathcal{D}$ of an object $\mathcal{S}$ is useful if there is a *structure preserving* map $h\colon \mathcal{D} \to \mathcal{S}$ where what we mean by "structure preserving" is support for logical inference. In the *Hyperproof* system (discussed in Section 1) for example, an observe inference rule ($|\!\approx$) is used to justify facts that can be directly extracted from the visual scene:

$$\begin{tikzcd} \mathcal{S} \arrow[dr, "\models"] & \\ \mathcal{D} \arrow[u, "h"] \arrow[r, "|\!\approx"'] & \Phi \end{tikzcd}$$

The representations we have developed are bear similar relationships to the real circuits they refer to—as we should expect—and therefore permit certain facts to be inferred by observation. An important question is the following: among our four kinds of representations, S for state-machine, C for circuit, T for timing, and F for formulas, for which pairs do we have an observe relation between them: F does not stand on the left of any such relation. $T \models\!\approx F$ makes sense and can be defined for a wide class of W's and all T's. $C \models\!\approx S$ makes sense: we can read off facts about the state diagram from the circuit, under the assumption that each state has a known assignment of values to the storage elements. $S \models\!\approx C$ does not seem to work, although we do have a form of inference which allows us to construact a C from an S. $S \models\!\approx T$, and by transitivity broadly construed, $C \models\!\approx F$ should make sense.

On the other hand, there are significant differences between *Hyperproof* and *Circuitproof* as it is developed so far. The notion of proof in *Circuitproof* is essentially a sequence of reasoning steps about a process from the outside, we look at the circuit as being the closed form of the transfer function from inputs to outputs. We tend to prove global properties about the transfer function, not local properties. This is somewhat at variance with *Hyperproof* where one is concerned with local properties of the situation. *Circuitproof* has no situational aspect as yet and this greatly complicates the comparison of the two systems.

One way to support the notion of a local semantics is to involve hierarchy. Our examples are at too low a level to excercise much hierarchical decomposition. Again, the presense of several distinct diagramatic forms, each contributing its own sense of a "black box", greatly complicates the organization of an argument. However, it is evident that the human designer is adept at managing these contending hierarchies: and this skill deserves support in the reasoning tools and study by logicians.

Our tentative formalization of diagrams is deficient in its handling of abstraction, as discussed briefly in the previous section. It is also lacking a coherent treatement of relevance. Perhaps the simplest notion of relevance is directionality: our attempt to develop a nondirectional semantics faltered on at least three occasions. The first was our attempt to characterize "implementation," the purely logical notions of entailment and consistency seeming to bracket a range of possible relationships. The second, minor problem arose in the discussion of state diagrams. Finally, our attempt to fix a semantics for timing diagrams exposed a substantive issue of relevant control.

Of course, directionality is the simplest manifestation of relevance. The single-faceted view provided by one of the diagram idioms represses the fact that there are other dependent views. Similarly, focusing on a single design module supresses the constraints offered by the rest of the design. Conversely we do not expect one part of a specification containing a contradiction to induce logical nonsense in another, unconnected part. Relevant connections can be handled at a logical level and ought to be if rigorous formalism is to contribute to engineering practice.

Acknowledgments

We thank Kathryn Fisler and Paul Miner for many helpful discussions and other contributions to this paper.

References

[1] Jon Barwise. Hyperproof: an exercise in situated logic. In *Proceedings of the North Americal Logic Programming Conference*, 1990.

[2] Jon Barwise and John Etchemendy. *Turing's World*. Academic Courseware Exchange, 1986.

[3] Jon Barwise and John Etchemendy. Visual information and valid reasoning. In W. Zimmerman and S. Cunningham, editors, *Visualization in Teaching and Learning Mathematics*, pages 9–24. Mathematical Association of America, 1991. *MAA Notes*, No. 19.

[4] Jon Barwise and John Etchemendy. *The Language of First-order logic, with Tarski's World 3.0.* CLSI Lecture Notes, 2nd edition, 1992.

[5] Jon Barwise and John Etchemendy. *Hyperproof*. University of Chicago Press, 1993. In press.

[6] Avra Cohn. Correctness properties of the viper block model: the second level. In G. Birtwistle and P. A. Subrahmanyam, editors, *Current Trends in Hardware Verification and Automated Theorem Proving*, pages 1–91, New York, 1989. Springer.

[7] A. L. Davis. What do computer architects design anyway? In G. Birtwistle and P. A. Subrahmanyam, editors, *Current Trends in Hardware Verification and Automated Theorem Proving*, pages 463–479. Springer, New York, 1989.

[8] Mike Gordon. Why higher-order logic is a good formalism for specifying and verifying hardware. In G. J. Milne and P. A. Subrahmanyam, editors, *Formal Aspects of VLSI Design*, pages 153–177, Amsterdam, 1986. North-Holland.

[9] E. M. Mayger, M. D. Francis, R. L. Harris, G. Musgrave, and M. P. Fourman. Dialog – linking formal proof to the design environment. Abstract Hardware Ltd. report, dated 1989. Uxbridge, U. K.

[10] Thomas F. Melham. *Formalizing Abstraction Mechanisms for Hardware Verification in Higher Order Logic*. PhD thesis, University of Cambridge, August 1990. Computer Laboratory Technical Report No. 201.

[11] S. Owre, J. M. Rushby, and N. Shankar. PVS: a prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, pages 748–752. Springer, 1992. Volume 607 of Springer *Lecture Notes in Artificial Intelligence*.

[12] Jr. Warren A. Hunt. a formal HDL and its use in the FM9001 verification. Second NASA Formal Methods Workshop 1992, NASA Conference Publication 10110, Langley Research Center, Hampton Virginia. From an oral presentation.

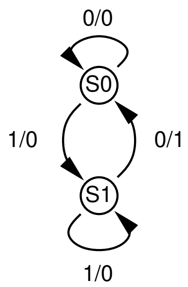
[13] Daniel Weise. Constraints, abstraction and verification. In M. Leeser and G. Brown, editors, *Hardware Specification, Verification and Synthesis: Mathematical Aspects*, pages 25–39, Berlin, 1989. Springer. Mathematical Sciences Institute Workshop, Cornell University.

[14] David Winkel and Franklin Prosser. *The Art of Digital Design*. Prentice-Hall, Englewood Cliffs, New Jersey, 2nd edition, 1989.

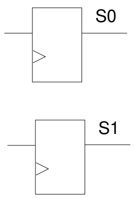
A Building a Circuit for a Machine

There are various ways to construct circuits from state diagrams. The *one-hot* technique developed below is one of them. We will use the fact that we can build a *combinational* circuit diagram for any boolean term. That is, if ϕ is a term over variables $I_1, \ldots, I_n$, and the function expressed by ϕ is $f\colon \{1,0\}^n \to \{1,0\}$, then we can inductively build circuit with signature $\{I_1, \ldots, I_n;\ O\}$ whose sequences are exactly those in which $O(t) = f(I_1(t), \ldots, I_n(t))$ for all $t \in Z$. We sometimes use the term as a name for its circuit-diagram representation.

Fix a machine M with signature $\{I_1, \ldots, I_n;\ O_1, \ldots O_m\}$. As an example to illustrate the construction, consider the unit-pulser machine:



We will construct a diagram D, beginning with an array of storage elements whose outputs are $\{R_s \mid s \text{ is a state of } M\}$. The idea is make R_s become 1 exactly when M is in state s. For our unit-pulser example, we have



Next, we will develop the inputs for each R_s.

1. In the degenerate case that there are no transitions into s, the input of R_s should be connected to a (diagram with output) 0.

2. Otherwise, develop a circuit-diagram representation of the term δ_s, defined below:

(a) For each transition t from s' to s with label $x_1 \cdots x_n / y_1 \cdots y_m$ define $\gamma_t \equiv S' \wedge \hat{I}_1 \wedge \cdots \hat{I}_n$. The fresh variable S' refers to the output of register $R_{s'}$; and

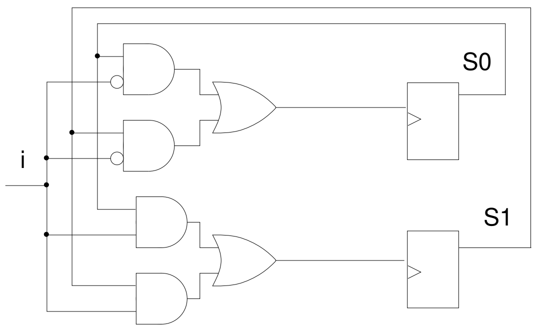
$$\hat{I}_k = \begin{cases} I_k & \text{if } x_k = 1 \\ \neg I_k & \text{if } x_k = 0 \end{cases}$$

(b) For the set of all such transitions, build the disjunction

$$\delta_s \equiv \bigvee \{\gamma_t \mid \; s' \xrightarrow{t} s \text{ for some state } s'\}$$

In the corresponding circuit diagram, each I_i represents a connection to the input terminal I_i, each new variable S represents a connection to the output of register R_s, and the sole output of δ_s is connected to the input of R_s.

The δ_{S0} and δ_{S1} terms are added to the unit-pulser circuit below. In the schematic we use the "mixed logic" convention that a line with a bubble at just one end logically negates its signal.

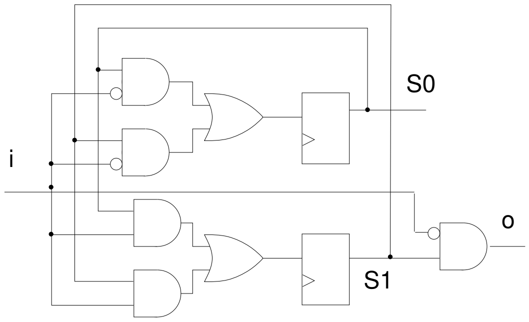


In a similar fashion, we will build a logical function making each output $O_j = 1$ exactly when the Mealy-machine specifies that it should be. That is, we implement the term

$$\beta_j \equiv \bigvee \{\alpha_t \mid \; s \xrightarrow{t[y_j=1]} s' \text{ for some states } s \text{ and } s'\}.$$

Where each $t[y_j = 1]$ refers to a transition in which $y_j = 1$; $\alpha_t \equiv S \wedge \hat{I}_1 \wedge \cdots \hat{I}_n$; and the S and $\hat{I}$'s are developed as before.

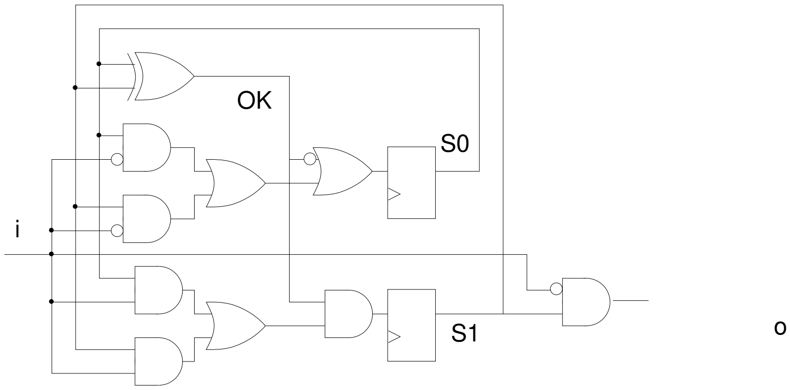
The circuit diagram for β_j is connected to input terminals and registers just as δ_s was; and its output is connected to output terminal O_j. For our unit-pulser example we get:



Now fix a machine M and any diagram D for M constructed as just described. The first thing to observe about D is that it does not have the same wave profile as M. We must restrict to those waveforms in which *exactly* one state-register is "on," for that condition is associated with M being in one of its states.

Now it is easy to prove that once D enters a valid state it will remain in a valid state forever. This invariant is maintained by the δ terms. But we cannot show that D is ever in a valid state, so let us add circuitry to assure it. We will develop a signal OK which is 1 exactly when for some i, $S_i(n) = 1$ and $S_k(n) = 0$ for $1 \leq k \neq i \leq p$; and conjoin OK with the input to every state register but one. This one exception will be that state that the circuit enters should it ever be in an invalid state.

These additions complete our unit-pulser example, where we have elected S0 to be the "reset" state:



Notice that in the profile model, where time is infinite in both directions, no Σ-sequence accepted by the circuit can ever have an invalid state. For if it did, then the previous state would have to be invalid, in which case the OK circuit would have corrected the problem. In practice, of course, circuits of this kind do fall into invalid states, due to external effects like cosmic rays.