Abstract Models for Concrete Constructions

Roger Bishop Jones

Abstract

Some preliminary explorative modelling of electronic circuits.

Created 2010/08/20

Last Change Date: 2014/11/08 19:43:27

Id: t040.doc,v 1.11 2014/11/08 19:43:27 rbj Exp

http://www.rbjones.com/rbjpub/pp/doc/t040.pdf

Contents

1 Prelude

This document develops some elementary theory about represenations of electronic circuits, intended to support later work on the semantics of languages for system design. It might possibly later be used in exemplifying techniques for addressing semantic problems in HOL.

Discussion of what might become of this document in the future may be found the postscript (Section 8).

In this document, phrases in coloured text are hyperlinks, like on a web page, which will usually get you to another part of this document (the blue parts, the contents list, page numbers in the Index) but sometimes take you (the red bits) somewhere altogether different (if you happen to be online) like online copy of this document.

[1]

2 Introduction

Text dumped to file t040i.tex

Last Change $ $*Date*: 2014/11/08 19:43:27 $ $

$ $*Id*: *t040.doc,v 1.11 2014/11/08 19:43:27 rbj Exp* $ $

I explore here in a fairly abstract way aspects of the theory of combining models of concrete components into larger systems.

This is conceived of as the application of formal techniques to design automation, but is also an exploration of possible connections between this kind of problem and techniques for denotation semantics, and the use of semantic methods analogous to shallow embeddings in formal analysis.

2.1 Provoking Problem

This document is the result of a though process provoked by a problem in the design of languages for hardware or co-design, i.e. one relevant to the design of a possible successor to languages like *verilog* and *VHDL*.

I am not myself interested in undertaking such a design, but am interested in the applicability of formal methods to such a design process.

Peter Flake identified to me a problem which he considered important in relation to the design of such a language, which was specifically, the reconciliation of cycle based and event based models of hardware. I though about this for while (with a rather limited understanding both of the domain in general and of these two kinds of model) and I had what seemed to me an insight which might help me to come up with "semantic primitives" which might be appropriate to the design of such a language. I started this document within a day of the discussion with Peter which provoked my supposed insight, with the intention of turning the insight into a formal model and some theorems capturing the insight, but found the desired result rather more difficult to obtain than I had anticipated. I have not come to doubt the idea, or that it can be formalised, but I have not yet found a satisfactory way of doing it, and because of the unexpected awkwardness, I have decided to make an attempt to describe the relevant matters informally.

2.1.1 Some Methodological Preliminaries

The methodological context is of some importance in understanding what I was hoping to achieve.

I have been exploring the utility of techniques similar to 'shallow' *semantic embedding* in other domains, and am interested in their applicability to the language design problem. This is a variant on the refrain which I associate with research on denotational semantics back in the '70s, that language design should begin with semantics rather than syntax (and abstract semantics before concrete, and possibly also, semantic *domains* first).

To this you add the idea that the earliest parts of this process (i.e. the exploration of the semantic domains) might benefit from being undertaken formally using an interactive proof tool.

If the design of a hardware description language were to be undertaken in this way, then the stages would be along the following lines.

1. semantic domains and primitive operations over those domains

2. develop the theory to evaluate whether these are appropriate

3. consider the constructors in some putative abstract syntax for the language, define these in terms of the primitive operators and evaluate

For this domain, because of my own limited experience in hardware verification, I though of the definition of circuits from components (or subsystems) using something likem a netlist, which one might hope to model as a system of equations. This is unlike the programming language situation in which one is not typically allowed to define a procedure by some arbitrary set of equations, and gives rise to a particular problem with consistency (that the system of equations might have no solution).

[HERE]

2.2 Languages and Models

I will expand on these connections here a little, because some features of the methods adopted may be easier to understand for a reader who has some understanding of the route by which I came to these methods.

We may consider formal analysis in general to consist in the development of *theories*, of one kind or another, generally in a logical or mathematical sense of that term. There are two distinct ways of approaching such a task which can usefully be constrasted before we then fudge the contrast. These correspond informally to universalist and pluralistic conceptions of language, or the question whether to have one large and complex language in which everything can be done, or a large number of specialised languages to chose between according to the needs of the problem. Russell was in this sense a universality, and Carnap became a pluralist (after a brief period following Russell's iniversalism). The idea of developing a theory provides a kind of fudge between these two idioms, the placement of the fudge depending on how you develop the theory.

In the case of a first order axiomatic theory, the mathematician or logician is supplied with a kit of parts (first order logic) with which to construct a special language in which a first order theory can be developed. To this kit the mathematician has to supply the details specific to his problem and he then gets 'a' first order language in which to develop the particular theory he has in mind. The things he has to supply may be conveniently described as:

- A signature.
- A set of axioms.

The 'signature' lists the names which the mathematician wishes to use in the theory, together with the kind of thing which they are names of (individuals, functions, predicates, relations). The axioms tell us what the mathematician requires of his subject matter as his starting point, and the mathematician will then be regarded as developing 'the' theory of those structures having features corresponding to the names in the signature and satisfying all the axioms.

That's a lightweight pluralism. You can get exactly the same theory working in the univesalistic context of first order set theory. This is a single first order language in which you define the kind of structures (things satisfying a signature) your theory is about, and then reason about them in first order set theory. In that universalist context you can also do things which you can't do as a first order pluralist without stepping up into the metalanguage, like comparing different structures of the kind you are reasoning about.

Developing a subject in set theory is very like working in set theory as a metalanguage for first order logic. However, you can forget lots of aspects of the language (you don't have to do metatheory), and focus on the subject matter of the language.

What we have here is, roughly parallel to the distinction between universalism and pluralism, a choice in how much machinery you invent for a new subject. You need to have some vocabulary for talking about the subject matter which will be particular to the subject. You might invent a new language so that not only are there special words, but there may be special notations which are not purely verbal. Alternatively, you can just stick with the new vocabulary and use it in some well established more or less universal context. The former involves a lot of extra work, for you not only have quite a complex task of designing a language, but you will also need to get tools, such as proof tools or compilers which can do the things you need to have done with the language, so it will be rare for this step to be taken. Universal languages also often include features which make it possible for you to get additional flexibility in what syntax to use for your theory without actually having to design a new language. For example, you might be able to chose the fixity of a function name (which controls whether the function goes before after or in between its arguments).

There are other kinds of fudge which we can do, convenient compromises on the question whether to devise a new language for a new problem domain.

One of these comes from semantics and may be associated with the idea of a "shallow (semantic) embedding". There is a particular way of organising the semantics of a language which is associated closely with a style of semantics called "denotational semantics". One feature of this kind of semantics is that they are *compositional*, a term which refers to a principle which Frege put forward in his thinking about logic, viz that the meaning of some structure should be a function of the meanings of its constituents. The function in question will vary according to the kind of structure involved. These ways of 'constructing' a syntactic complex from its parts may be call constructions, and when an abstract rendition of the syntax of a language is given they are the constuctors of the abstract syntax.

A compositional semantics can be rendered parallel to the abstract syntax by supplying a semantic function corresponding to each constructor in the abstract syntax. Such a semantic function maps the meanings of the syntactic components of a complex to the meaning of the whole complex. When semantics is organised in this way we get a nice mathematical relationship between the syntax on the one hand, and the semantics on the other, as viewed through the respective constructors. The semantics becomes an algebraic homomorphism.

If pluralistic semantics is thought of in this way, a means is provided for further systematic fudging of the distinction between universalism and pluralism. If you have such a semantics, and a universal context in which the semantic operators can be

3

In the first instance this is approached here by considering the case of electronic circuits, while keeping an eye open for possible generalisations beyond that domain.

The initial purpose of this document is to explore some ideas on theoretical aspects of the design of hardware description languages intended for use in the larger context of co-design. The language may ultimately be an aspect of a more broadly scope language for co-design of concurrent systems intended to facilitate the development both of digital hardware and of software which together form some kind of system, but in this document only the underpinnings of a hardware description language are considered.

Undertaking the semantics of the language first consists in considering what the language is intended to be about, coming up with general abstract mathematical models for the kinds of entities which the language talks about, and describing the kinds of operators over these entities which are expected to be expressed in the language. Eventually these abstract entities and the operations we define over them might form a part of a formal semantics for the hardware description language, in which case (supposing that the semantics is some kind of denotational semantics) the meanings of the constructs of the language will be defined in terms of the operations here considered.

In the case of hardware description languages, we are concerned with the construction of complex electronic systems by connecting together a selection of smaller subsystems or components, and it is the distinctive nature of this composition by wiring together which occupies most of our energy here.

We seek (at least initially) to do little more than give an account of wiring diagrams, because one of the issues to be addressed is the integration of diverse approaches to the treatment of other aspects of these systems.

4 Functional Models

The aim here is to probe the difficulty in obtaining a formal model of electronic hardware which is abstract end general enough to underpin the semantics of a fairly wide range of harware description languages.

It is not based on a good knowledge of existing languages, and the probability of success is not high, but if initial exploration go well I may look a little closer at existing hardware description languages to see whether the theory provides an adequate for their semantics.

The idea I propose to explore is that a circuit be modelled by a function from a histories of values on input lines to values on output lines. The principal initial aim is to chose a type for representing such functions which is convenient for the description of putting together circuits in a manner analogous to a wiring diagram to form larger circuits and to prove a result about when such compositions of circuits yields a new circuit (ideally always).

I begin therefore with some type abbreviations (later I might possibly substitute definitions of type constructors, if this seems likely to be advantageous).

First we need a notion of signature for circuits, which tells us what the input and output ports are.

In the following *'a* is a type of tags, a list of which is the name of a port. The signature is a pair of sets of port names, input ports on the left, output ports on the right.

SML

```
declare_type_abbrev ("CIRCSIG", ["'a"], ⌜:('a LIST SET × 'a LIST SET)⌝);
```

Next the type of a circuit:

In the following the type variables are for:

- 'a a type of tags
- 'b a type of signal values
- 'c a type of time values

Circuits are modelled by functions. The domain and codmain of the function consists of indexed sets of signals where signals are total functions over tune. The the indexes are names, which are lists of tags.

Domains of the indexed sets in the domain and codomain of the function are the sets of input port names and output port names respectively.

SML

```
declare_type_abbrev ("CIRC", ["'a", "'b", "'c"],
                ⌜:('a LIST, 'c → 'b)IX → ('a LIST, 'c → 'b)IX⌝);
```

When a circuit are connected by a wiring diagram it is natural to treat the connections in the wiring as equations between the signals at the points of connection. Together with equations which express the function of the connected circuits these give a system of equations which determines the behaviour of the resulting circuit. In the real world some behaviour will always result, but when we are working with mathematical models these models may involve idealisations which result in some of these systems of equations having no solution. An example of this would be if combinatory logic components were modelled as having no propagation delay and a circuit is constructed from such components. If the diagram contains no cylic paths then all will be well, but if cycles are present, as in the trivial case of wiring the output of a logical negation to its input, then there will be no solution to the resulting equations.

There are two strategies which are admitted here for avoiding such contradictions. The first is the avoidance of cycles in the wiring diagram, the second is the avoidance of zero delay components. In either case it is important that the models of components and hence of circuits made from them, have outputs which do not anticipate future inputs.

This leads me to define two similar properties, one for the case of components which might have zero delay, and one for the case that zero delays are excluded. We then anticipate a general result to the effect that diagrams made from no-zero-delay components (there must be a word for this) always yield circuits consistent with all the expected equations, whether or not the diagram is acyclic.

For reasoning generally over partial orders I introduce the infix name $\leqslant_p$ (to be used as a variable name):

SML

```
declare_infix (210, "≤p");
```

In following definitions, the strictly version is relevant to circuits which may not react instantaneously to a change on the input, the other version applies to all circuits including those with zero-delay.

HOL Constant

$$\textbf{SimilarHistory} : ('c\ SET \times ('c \to 'c \to BOOL)) \to ('a\ LIST, 'c \to 'b)IX \to ('a\ LIST, 'c \to 'b)IX \to BOOL$$

$$\forall D\ \$\leqslant_p\ g\ h \bullet SimilarHistory\ (D, \$\leqslant_p)\ g\ h \Leftrightarrow IxDom\ g = IxDom\ h \wedge \forall t\ a \bullet a \in IxDom\ g \Rightarrow \forall u \bullet u \in D \wedge u \leqslant_p t \Rightarrow ValueOf\ (g\ a)\ u = ValueOf\ (h\ a)\ u$$

HOL Constant

$$\textbf{Strict2} : ('c \to 'c \to BOOL) \to ('c \to 'c \to BOOL)$$

$$\forall \$\leqslant_p \bullet Strict2\ \$\leqslant_p = \lambda x\ y \bullet x \leqslant_p y \wedge \neg\ y \leqslant_p x$$

HOL Constant

$$\begin{array}{l} \textbf{\textit{StrictlySimilarH}} : (\prime c\ SET \times (\prime c \to \prime c \to BOOL)) \\ \qquad\qquad \to (\prime a\ LIST, \prime c \to \prime b)IX \to (\prime a\ LIST, \prime c \to \prime b)IX \to BOOL \\ \hline \forall D\ \$\leqslant_p\ g\ h \bullet\ StrictlySimilarH\ (D, \$\leqslant_p)\ g\ h \Leftrightarrow SimilarHistory\ (D, Strict2\ \$\leqslant_p)\ g\ h \end{array}$$

HOL Constant

$$\begin{array}{l} \textbf{\textit{Historical}} : (\prime c\ SET \times (\prime c \to \prime c \to BOOL)) \to (\prime a,\prime b,\prime c)CIRC \to BOOL \\ \hline \forall D\ \$\leqslant_p\ c \bullet\ Historical\ (D, \$\leqslant_p)\ c \Leftrightarrow \forall g\ h \bullet\ SimilarHistory\ (D, \$\leqslant_p)\ g\ h \Rightarrow c\ g = c\ h \end{array}$$

HOL Constant

$$\begin{array}{l} \textbf{\textit{StrictlyHistorical}} : (\prime c\ SET \times (\prime c \to \prime c \to BOOL)) \to (\prime a,\prime b,\prime c)CIRC \to BOOL \\ \hline \forall D\ \$\leqslant_p\ c \bullet\ StrictlyHistorical\ (D, \$\leqslant_p)\ c \Leftrightarrow \forall g\ h \bullet\ StrictlySimilarH\ (D, \$\leqslant_p)\ g\ h \Rightarrow c\ g = c\ h \end{array}$$

5 Composition of Circuits

The method of composition is intended to provide a semantic analogue of wiring components together.

There is a tenuous analogy with the ideal of the colimit of a diagram in a category, which might provide the basis of a further abstraction to a more general conception of how systems can be combined to yield more complex systems. The general scheme is that a diagram corresponds to a set of instructions for building a complex structure from components and may be thought of as abstract syntax in some appropriate language. The taking of the colimit of the diagram may then be seen as offering the semantics of the diagram.

I do however attempt a separation of the instructions for assembly from the actual components to be used in the construction, and this I have not seen in the category theoretic notion of diagram, in which the specific objects to be assembled are mentioned, and the ways in which the assembly is constrained are rather limited (taking the arrows in a category theoretic diagram to be such constraints). So the analogy is weak, but a generalisation from circuit diagrams remains of interest, and it remains possibly that category theory might help with that[1].

The information necessary to the construction of a composite circuit is as follows:

(A) A set of named occurrences of components circuits.

(B) A set port names.

(D) A set of connections.

Of these we may observe that the first may be thought of as the things to which the construction described by the diagram are applied, and confine our conception of the diagram to the manner of interconnection. Some aspects of (A) are needed for the diagram, details of the port configuration of

[1] I am looking for general perspectives on formal modelling and trying to understand whether or not category theory might be helpful in this.

each component. This information is a bit like a type for the component, and the notion of *signature* may be thought of as a generalisation of the concept of *type*, which we will employ for this purpose.

The content of a diagram then becomes:

(A) A signature for the circuit to be constructed.

(B) An indexed set of signatures for the components to be used in the construction.

(D) A set of connections.

We combine these items together into an object of the following labelled product type. The port names element is not explicit but can be recovered from the connections, as can the type of the ports, provided that certain well-formedness constraints are satisified.

HOL Labelled Product

CDIAG

CSig	$: ('a)\,CIRCSIG;$
CSigs	$: ('a,\ ('a)\,CIRCSIG)\,IX;$
CCons	$: ('a\ LIST \times 'a\ LIST)\ SET$

The well-formedness conditions are as follows:

1. names which appear in the the signatures in *CSigs* will be augmented by adding the component name in front and then may appear in one or more of the connections in *CCons*.
2. such names will only appear on the left of a connection if they are input ports names (on the relevant component circuit) and will only appear on the right if they are output port names
3. other names appearing in *CCons* are the names of ports on the circuit constructed by the diagram, and will only appear on one side of the connections, from which the type of port may be inferred.
4. *CCons* will be a many-one relationship (no input port is connected to more than one output port).
5. all the input ports from CSigs will be in the domain of *CCons*.

HOL Constant

$$\textbf{\textit{DiagNames}} : (('a)\ CIRCSIG \to 'a\ LIST\ SET) \to ('a)\ CDIAG \to 'a\ LIST\ SET$$

$$\forall f\ cd \bullet\ DiagNames\ f\ cd =$$
$$\bigcup\{is \mid \exists x \bullet\ x \in IxDom\ (CSigs\ cd)$$
$$\wedge\ is = \{y \mid \exists z \bullet\ z \in f\ (ValueOf\ (CSigs\ cd\ x)) \wedge y = Cons\ x\ z\}\}$$

HOL Constant

$$\boldsymbol{WellFormedDiag} : ('a)\ CDIAG \rightarrow BOOL$$

$$\forall cd \bullet\ WellFormedDiag\ cd \Leftrightarrow \exists inps\ outps \bullet\ inps \cap outps = \{\}$$
$$\wedge\ DiagNames\ Fst\ cd \subseteq inps$$
$$\wedge\ DiagNames\ Snd\ cd \subseteq outps$$
$$\wedge\ Dom\ (CCons\ cd) = inps$$
$$\wedge\ Ran\ (CCons\ cd) \subseteq outps$$
$$\wedge\ (CCons\ cd) \in Functional$$

To give the semantics of such a construction we treat the subsidiary circuits as imposing constraints on the possible values on the wires and output ports in the context of the input ports. It may be helpful to think of the wiring diagram as a sentence of first order logic in which each circuit is a relation between its signals on its ports. An interpretation of such a formula will be an assignment of values to the signal names. Each equation, either expressing the functionality of a component or the effect of a connection in the wiring, eliminates all those interpretations which do not comply with it. One hopes that the set of equations will have exactly one solution for each assignment of values to the input ports of the whole diagram. The functionality of the whole will then be the the relation between input and output values obtained by projecting onto the external ports. A general result should be obtainable for strictly historical components, provided that the time is well-founded. Diagrams using components modelled with continuous time are more difficult to prove consistent.

The following definition expresses the conditions for a circuit to be a correct construction of a circuit from an indexed set of circuits following a particular wiring diagram.

This is expressed by asserting the existence of a set of signals, one for each name in the signature of the diagram which satisfy all the equations in the signature of the diagram and match the functionality of the circuits being composed.

The required satsifaction conditions are compounded as follows.

First the condition for the collection of signals to satisfy a single circuit, which is that the output signals are obtainable from the input signals by application of the function representing the circuit.

HOL Constant

$$\boldsymbol{CircSat} : ('a)\ CIRCSIG \rightarrow ('a\ LIST, 'c \rightarrow 'b)IX\ \rightarrow ('a, 'b, 'c)\ CIRC \rightarrow BOOL$$

$$\forall cs\ ixs\ c \bullet\ CircSat\ cs\ ixs\ c \Leftrightarrow (Snd\ cs) \lhd ixs = c\ ((Fst\ cs) \lhd ixs)$$

Then the condition for satisfaction of all the circuits, viz. that the conditions for satisfying each component circuit are all satisfied.

HOL Constant

$$\boldsymbol{CircsSat} : ('a)\ CDIAG \rightarrow ('a\ LIST, 'c \rightarrow 'b)IX\ \rightarrow ('a, ('a, 'b, 'c)\ CIRC)IX \rightarrow BOOL$$

$$\forall cd\ ixs\ ixc \bullet\ CircsSat\ cd\ ixs\ ixc \Leftrightarrow$$
$$\forall n \bullet\ n \in IxDom\ ixc \Rightarrow CircSat\ (ValueOf(CSigs\ cd\ n))\ ixs\ (ValueOf\ (ixc\ n))$$

An equation arising from connection of two signals by a wire is satisfied, naturally, by the equality of the two signals connected.

HOL Constant

$$\boldsymbol{EqSat} : ('a\ LIST, 'c \to 'b)IX \to 'a\ LIST \times 'a\ LIST \to BOOL$$

$$\forall ixs\ n\ m \bullet\ EqSat\ ixs\ (n, m) \Leftrightarrow n \in IxDom\ ixs \wedge m \in IxDom\ ixs \wedge ixs\ n = ixs\ m$$

The condition for satisfying all the wiring connections being that each connection is satisfied.

HOL Constant

$$\boldsymbol{EqsSat} : ('a\ LIST, 'c \to 'b)IX \to ('a\ LIST \times 'a\ LIST)SET \to BOOL$$

$$\forall ixs\ nms \bullet\ EqsSat\ ixs\ nms \Leftrightarrow \forall nm \bullet\ nm \in nms \Rightarrow EqSat\ ixs\ nm$$

The following expresses the relationship between a circuit diagram, a collection of components which fit into that diagram, and the circuit which is obtained by composing those circuits in the manner prescribed by the diagram. The result will be a projection from the complete collection of signals to the subset which corrsponds to the signature of the resulting circuit.

HOL Constant

$$\boldsymbol{DiagCircRel} : ('a)\ CDIAG \to ('a, ('a, 'b, 'c)\ CIRC)IX \to ('a, 'b, 'c)\ CIRC \to BOOL$$

$$\forall cd\ isc\ c \bullet\ DiagCircRel\ cd\ isc\ c \Leftrightarrow \exists six{:}('a\ LIST, 'c \to 'b)IX \bullet$$
$$CircsSat\ cd\ six\ isc \wedge EqsSat\ six\ (CCons\ cd) \wedge CircSat\ (CSig\ cd)\ six\ c$$

We could define the construction using a choice function as follows:

HOL Constant

$$\boldsymbol{Diag2Circ} : ('a)\ CDIAG \to ('a, ('a, 'b, 'c)\ CIRC)IX \to ('a, 'b, 'c)\ CIRC$$

$$\forall d{:}('a)\ CDIAG;\ c \bullet\ Diag2Circ\ d\ c = \epsilon x \bullet\ DiagCircRel\ d\ c\ x$$

But in order to make use of such a definition we would first have to show that the relationship is satisfiable, and in order to do that we will find it necessary to come up with a more constructive definition.

In fact, more than one may be necessary. Two distinct cases may be envisaged. In the case that we have a well-founded ordering on time, relative to which our signals are strictly historical, then we may be able to obtain our result as a fixed point of a functor which respects that ordering. This however will not be the case if time is continuous, and some other conditions (perhaps continuity) will be necessary to establish the existence of a fixed point for the equations.

5.1 When Time is Well-Founded

We are looking for a functor fixed points of which are circuits satisfying *DiagCircRel*, and an ordering on circuits, presumably derived from the ordering on time, which is respected by the functor.

The functor will operate over the complete selection of signals, and a projection will be taken once the fixed point is obtained.

We need the operands over which the functor operates to be themselved functions over time, so we convert an indexed set of traces to a single trace whose codomain is a type of indexed sets as follows:

HOL Constant

$$\boldsymbol{tr22tr1} : (('a\ LIST, 'c \to 'b)IX) \to ('c \to ('a\ LIST, 'b)IX)$$

$\forall t2 \bullet\ tr22tr1\ t2 = \lambda t\ n \bullet$
$\quad if\ n \in IxDom\ t2$
$\quad then\ Value\ (ValueOf\ (t2\ n)\ t)$
$\quad else\ Undefined$

HOL Constant

$$\boldsymbol{tr12tr2} : ('c \to ('a\ LIST, 'b)IX) \to (('a\ LIST, 'c \to 'b)IX)$$

$\forall t1 \bullet\ tr12tr2\ t1 = \lambda n \bullet$
$\quad if\ n \in IxDom\ (t1\ (\epsilon x \bullet T))$
$\quad then\ Value\ \lambda t \bullet\ ValueOf\ (t1\ t\ n)$
$\quad else\ Undefined$

In order to obtain a set of values for the signals at some time we have to extract from the diagram various signals. Signals are either inputs or outputs to the diagram (lets call these external), or inputs or outputs to a component (which I will call internal).

In order to obtain the value of an internal input or an external output, we locate the internal output or external input signal to which it is connected. In order to obtain the value of an internal output we locate the input signals to the component of which it is an output, and apply the function modelling the component to those signals.

Since inputs are in the domain of the connection function in the diagram, and outputs are in its range, it is easy to establish which case applies.

These two steps have to be combined in one recursion of our functor to facilitate the proof that the recursion is well-founded. So obtain the value of an output port signal we must apply the circuit function to the signals to which its inputs are connected, taking care to label these signals with the name of the relevant input port, and then to supplement the output signals obtained from the circuit with the component name from the diagram.

This function determines which set of signals are required and returns a function which projects the required signals from a larger collection renaming as necessary for input to the relevant circuit.

HOL Constant

$$\boldsymbol{InputSignals} : ('a)\ CDIAG \to 'a \to (('a\ LIST, 'c \to 'b)IX) \to (('a\ LIST, 'c \to 'b)IX)$$

$\forall cd\ cn \bullet\ InputSignals\ cd\ cn =$
$\quad let\ csig = ValueOf\ (CSigs\ cd\ cn)$
$\quad in\ let\ cins = FunImage\ (Cons\ cn)\ (Fst\ csig)$
$\quad in\ let\ map = \ cins \lhd (CCons\ cd)$
$\quad in\ \lambda is\ sn \bullet\ if\ (Cons\ cn\ sn) \in Dom\ map\ then\ is\ (Cons\ cn\ sn)\ else\ Undefined$

The following definition gives a parameterised functor. The parameters are:

- a diagram

- an indexed set of components suitable for composition according to the diagram
- a indexed set of input signals matching the input ports to the composite circuit

The functor then returns transformed traces in which the trace values are complete indexed sets of values, one for each signal in the compound circuit external and internal. This functor defines values of the output trace in terms of temporally preceding values of the input trace. Provided that the component circuits are strictly historical relative to the (possibly partial) ordering on time, and that ordering is well-founded, then the recursion implicit in the functor will be well-founded and the functor will have a fixed point, so we can define the required composite circuit in terms of the fixed point of the functor.

The principal result we need specifies conditions under which a diagram yields a circuit satisfying DiagCircRel. This will be, certain conditions for well formedness of the diagram and that the constituent circuits are all strictly historical relative to a well-founded ordering on time.

The proof of the required result will use a recursion theorem to the effect that a functor which respects some well-founded ordering has a fixed point. The first step in the proof is to define this functor. The function which we wish to obtain as a fixed point of this functor is obtained by a process which essentially involves obtaining the values of internal signals even though their values are then discarded. The recursion must therefore take place in defining a function which does not discard the internal values, which will be discarded only when we have the required fixed point of the more informative function.

In the following definition the functor defined operates on circuits represented as if all internal wires were output ports.

6 Alloy

This section attempts an embedding of the Alloy relational calculus in HOL.

6.1 Lightweight Semantics as Shallow Embedding

This is based on the semantics for Alloy given in Emina Torlak's thesis [2] Figure 2-1. Since it is a 'shallow semantic embedding' the semantics is not given as a map, we simply define operators corresponding to the constructions and try to arrange the syntax as closely as possibly so as to permit Alloy to be expressed in HOL. This could be a prelude to use of the relevant operations in giving a 'deep' semantics, or to rendering some Alloy model in HOL, or to reasoning about Alloy in general or some model using HOL.

SML

```
declare_type_abbrev("tuple", [], ⌜:ℕ LIST⌝);
declare_type_abbrev("rel", [], ⌜:tuple SET⌝);
```

The admission of LISTs in general as tuples may not be satisfactory since this includes the empty list which would be the 0-tuple. I don't know whether it is intended to admit or exclude 0-tuples, but I suspect that the effect is detrimental (i.e. that it delivers no important benefits but complicates the theory).

6.1.1 Problem (P)

A problem is the conjunction of its relation bounds and formulae, which we can just spell out. The domain of discourse is just a set. We need to constrain the range of the variables in the specification to range over relations confined to this domain.

6.1.2 Restriction (T)

Can be expressed using the subset relation.

6.1.3 Formulae (F)

HOL Constant

$$\boldsymbol{no} : rel \to BOOL$$

$$\forall r \bullet\ no\ r \Leftrightarrow r = \{\}$$

HOL Constant

$$\boldsymbol{lone} : rel \to BOOL$$

$$\forall r \bullet\ lone\ r \Leftrightarrow \exists x \bullet\ r \subseteq \{x\}$$

HOL Constant

$$\boldsymbol{one} : rel \to BOOL$$

$$\forall r \bullet\ one\ r \Leftrightarrow \exists x \bullet\ r = \{x\}$$

HOL Constant

$$\boldsymbol{some} : rel \to BOOL$$

$$\forall r \bullet\ some\ r \Leftrightarrow \exists x \bullet\ x \in r$$

Subset, equality and the boolean operations are already suitably defined in HOL. We do not have suitable restricted quantifiers but the effect can be obtained by writing out in full using the normal HOL quantifiers.

6.1.4 Expressions (E)

Variables are rendered as HOL variables of type rel.

Operations over expressions are operations over sets. Many of these are already defined in HOL, and may be aliased for the present context.

SML

```
declare_prefix(50, "~a");
```

HOL Constant

$\$^{\sim}_{\boldsymbol{a}} : rel \to rel$

$\forall p\bullet\ (^{\sim}_{a}\ p) = \{x \mid \exists y\ z\bullet\ x = [y;z] \wedge [z;y] \in p\}$

SML

$declare\_prefix(50,\ "^{\frown}_{a}");$

HOL Constant

$\$^{\frown}_{\boldsymbol{a}} : rel \to rel$

$\forall p\bullet\ (^{\frown}_{a}\ p) = \{x \mid \exists y\ z\bullet\ x = [y;z] \wedge tc\ (\lambda v\ w\bullet\ [v;w] \in p)\ z\ y\}$

SML

$declare\_alias\ ("^{\frown}",\ \ulcorner\$^{\frown}_{a}\urcorner);$

SML

$declare\_prefix(50,\ "*");$

HOL Constant

$\$^{*} : rel \to rel$

$\forall p\bullet\ (^{*}\ p) = (^{\frown}_{a}\ p) \cup \{x \mid \exists y\bullet\ x = [y;y]\}$

Union, intersection and difference of sets is already suitably defined.

This brings us to the join operation. The definition here is not the same as the same in Emina Torlak's thesis [2], since I am guessing that she has made a small error.

SML

$declare\_infix(310,\ ".{}_{a}");$

HOL Constant

$\$._{\boldsymbol{a}} : rel \to rel \to rel$

$\forall p\ q\bullet\ p\ ._{a}\ q = \{x{:}tuple \mid \exists(u{:}tuple)\ v\ (w{:}tuple)\bullet\ u@[v] \in p \wedge [v]@w \in q \wedge x = u@w\}$

Product.

SML

$declare\_alias\ (".",\ \ulcorner\$._{a}\urcorner);$

SML

$declare\_infix(310,\ "\to_{a}");$

HOL Constant

$$\$\rightarrow_{\boldsymbol{a}} : rel \rightarrow rel \rightarrow rel$$

$$\forall p\ q \bullet\ p \rightarrow_a q = \{x{:}tuple \mid \exists (u{:}tuple)\ (w{:}tuple) \bullet\ u \in p \wedge w \in q \wedge x = u@w\}$$

SML

$$declare\_alias\ (\texttt{"}\rightarrow\texttt{"},\ \ulcorner \$\rightarrow_a \urcorner);$$

We will use 'if .. then .. else ..' syntax for conditionals.

HOL set comprehension will suffice for Alloy comprehension.

6.2 Some Theorems

6.3 Sample Specification

This is the one from Emma Torlak's thesis [2] Figure 4-1.

It is presented as a set. Note that in some places I have had to add subscript a to disambiguate aliases. When printed in the theory listing (See Appendix A.2). Note also that function application associates to the right (though it is in fact semantically more like relational composition and is associative so long as the things applied are functions).

HOL Constant

$$\boldsymbol{ToyList} : (rel \times rel \times rel \times rel \times rel \times rel \times rel \times rel)\ SET$$

$$\begin{array}{l}
ToyList = \{(list,\ nil,\ cons,\ thing,\ car,\ cdr,\ equivTo,\ prefixes) \mid \\
\quad list = cons \cup nil \\
\wedge \quad no\ (cons \cap nil) \\
\wedge \quad car \subseteq (cons \rightarrow thing) \\
\wedge \quad (\forall a \bullet\ a \in cons \Rightarrow one\ (\{a\}.car)) \\
\wedge \quad cdr \subseteq (cons \rightarrow list) \\
\wedge \quad (\forall a \bullet\ a \in cons \Rightarrow one\ (\{a\}.cdr)) \\
\wedge \quad (\forall a \bullet\ a \in list \Rightarrow \exists e \bullet\ e \in nil \wedge e \in \{a\}.(\hat{}_a\ cdr)) \\
\wedge \quad equivTo \subseteq (list \rightarrow list) \\
\wedge \quad (\forall a\ b \bullet\ a \in list \wedge b \in list \Rightarrow \\
\qquad\qquad (\{a\} \subseteq \{b\}.equivTo \\
\qquad\qquad \Leftrightarrow\ \{a\}._a\ car = \{b\}._a\ car \\
\qquad\qquad\qquad \wedge \{a\}._a\ cdr.equivTo = \{b\}._a\ cdr.equivTo)) \\
\wedge \quad prefixes \subseteq (list \rightarrow list) \\
\wedge \quad (\forall e\ a \bullet\ e \in nil \wedge a \in list \Rightarrow (\{e\} \subseteq \{a\}.prefixes)) \\
\wedge \quad (\forall e\ a \bullet\ e \in nil \wedge a \in cons \Rightarrow \neg\ (\{a\} \subseteq \{e\}.prefixes)) \\
\wedge \quad (\forall a\ b \bullet\ a \in cons \wedge b \in cons \Rightarrow \\
\qquad\qquad (\{a\} \subseteq \{b\}.prefixes \\
\qquad\qquad \Leftrightarrow \{a\}._a\ car = \{b\}._a\ car \\
\qquad\qquad\qquad \wedge \{a\}.cdr \subseteq \{b\}.cdr._a\ prefixes)) \\
\}
\end{array}$$

To reason about this specification in HOL it would probably be most convenient first to demonstrate the consistency of the specification (the non-emptyness of the above set), and then to introduce new constants complying with the specifications and reason about these constants.

7 Conclusions

8 Postscript

A Theory Listings

A.1 The Theory circuit

Parents

ordered_sets

Constants

SimilarHistory
$'c\ \mathbb{P} \times ('c \to 'c \to BOOL)$
$\quad \to ('a\ LIST, 'c \to 'b)\ IX$
$\quad \to ('a\ LIST, 'c \to 'b)\ IX$
$\quad \to BOOL$

Strict2 $('c \to 'c \to BOOL) \to 'c \to 'c \to BOOL$

StrictlySimilarH
$'c\ \mathbb{P} \times ('c \to 'c \to BOOL)$
$\quad \to ('a\ LIST, 'c \to 'b)\ IX$
$\quad \to ('a\ LIST, 'c \to 'b)\ IX$
$\quad \to BOOL$

Historical $'c\ \mathbb{P} \times ('c \to 'c \to BOOL) \to ('a, 'b, 'c)\ CIRC \to BOOL$

StrictlyHistorical
$'c\ \mathbb{P} \times ('c \to 'c \to BOOL) \to ('a, 'b, 'c)\ CIRC \to BOOL$

CCons $'a\ CDIAG \to 'a\ LIST \leftrightarrow 'a\ LIST$

CSigs $'a\ CDIAG \to ('a, 'a\ CIRCSIG)\ IX$

CSig $'a\ CDIAG \to 'a\ CIRCSIG$

MkCDIAG $'a\ CIRCSIG$
$\quad \to ('a, 'a\ CIRCSIG)\ IX$
$\quad \to 'a\ LIST \leftrightarrow 'a\ LIST$
$\quad \to 'a\ CDIAG$

DiagNames $('a\ CIRCSIG \to 'a\ LIST\ \mathbb{P}) \to 'a\ CDIAG \to 'a\ LIST\ \mathbb{P}$

WellFormedDiag
$'a\ CDIAG \to BOOL$

CircSat $'a\ CIRCSIG$
$\quad \to ('a\ LIST, 'c \to 'b)\ IX$
$\quad \to ('a, 'b, 'c)\ CIRC$
$\quad \to BOOL$

CircsSat $'a\ CDIAG$
$\quad \to ('a\ LIST, 'c \to 'b)\ IX$
$\quad \to ('a, ('a, 'b, 'c)\ CIRC)\ IX$
$\quad \to BOOL$

EqSat $('a\ LIST, 'c \to 'b)\ IX \to 'a\ LIST \times 'a\ LIST \to BOOL$

EqsSat $('a\ LIST, 'c \to 'b)\ IX \to 'a\ LIST \leftrightarrow 'a\ LIST \to BOOL$

DiagCircRel $'a\ CDIAG$
$\quad \to ('a, ('a, 'b, 'c)\ CIRC)\ IX$
$\quad \to ('a, 'b, 'c)\ CIRC$
$\quad \to BOOL$

Diag2Circ $'a\ CDIAG$
$\quad \to ('a, ('a, 'b, 'c)\ CIRC)\ IX$
$\quad \to ('a, 'b, 'c)\ CIRC$

tr22tr1 $('a\ LIST, 'c \to 'b)\ IX \to 'c \to ('a\ LIST, 'b)\ IX$
tr12tr2 $('c \to ('a\ LIST, 'b)\ IX) \to ('a\ LIST, 'c \to 'b)\ IX$
InputSignals $'a\ CDIAG \to 'a \to ('a, 'b, 'c)\ CIRC$

Types

$'1$ ***CDIAG***

Type Abbreviations

$'a$ ***CIRCSIG*** $'a\ CIRCSIG$
$('a, 'b, 'c)$ ***CIRC***
$('a, 'b, 'c)\ CIRC$

Fixity

Right Infix 210:
$\leqslant_{p}$

Definitions

SimilarHistory
$\vdash \forall\ D\ \$\leqslant_p\ g\ h$
• $SimilarHistory\ (D, \$\leqslant_p)\ g\ h$
$\Leftrightarrow IxDom\ g = IxDom\ h$
$\wedge\ (\forall\ t\ a$
• $a \in IxDom\ g$
$\Rightarrow\ (\forall\ u$
• $u \in D\ \wedge\ u \leqslant_p t$
$\Rightarrow ValueOf\ (g\ a)\ u = ValueOf\ (h\ a)\ u))$

Strict2 $\vdash \forall\ \$\leqslant_p \bullet\ Strict2\ \$\leqslant_p = (\lambda\ x\ y \bullet\ x \leqslant_p y \wedge \neg\ y \leqslant_p x)$

StrictlySimilarH
$\vdash \forall\ D\ \$\leqslant_p\ g\ h$
• $StrictlySimilarH\ (D, \$\leqslant_p)\ g\ h$
$\Leftrightarrow SimilarHistory\ (D, Strict2\ \$\leqslant_p)\ g\ h$

Historical $\vdash \forall\ D\ \$\leqslant_p\ c$
• $Historical\ (D, \$\leqslant_p)\ c$
$\Leftrightarrow\ (\forall\ g\ h$
• $SimilarHistory\ (D, \$\leqslant_p)\ g\ h \Rightarrow c\ g = c\ h)$

StrictlyHistorical
$\vdash \forall\ D\ \$\leqslant_p\ c$
• $StrictlyHistorical\ (D, \$\leqslant_p)\ c$
$\Leftrightarrow\ (\forall\ g\ h$
• $StrictlySimilarH\ (D, \$\leqslant_p)\ g\ h \Rightarrow c\ g = c\ h)$

CDIAG $\vdash \exists\ f \bullet\ TypeDefn\ (\lambda\ x \bullet\ T)\ f$
MkCDIAG
CSig
CSigs
CCons $\vdash \forall\ t\ x1\ x2\ x3$
• $CSig\ (MkCDIAG\ x1\ x2\ x3) = x1$

```
                ∧ CSigs (MkCDIAG x1 x2 x3) = x2
                ∧ CCons (MkCDIAG x1 x2 x3) = x3
                ∧ MkCDIAG (CSig t) (CSigs t) (CCons t) = t
DiagNames       ⊢ ∀ f cd
                  • DiagNames f cd
                    = ⋃
                      {is
                       |∃ x
                        • x ∈ IxDom (CSigs cd)
                          ∧ is
                            = {y
                            |∃ z
                             • z ∈ f (ValueOf (CSigs cd x))
                               ∧ y = Cons x z}}
WellFormedDiag
                ⊢ ∀ cd
                  • WellFormedDiag cd
                    ⇔ (∃ inps outps
                    • inps ∩ outps = {}
                      ∧ DiagNames Fst cd ⊆ inps
                      ∧ DiagNames Snd cd ⊆ outps
                      ∧ Dom (CCons cd) = inps
                      ∧ Ran (CCons cd) ⊆ outps
                      ∧ CCons cd ∈ Functional)
CircSat         ⊢ ∀ cs ixs c
                  • CircSat cs ixs c ⇔ Snd cs ◁ ixs = c (Fst cs ◁ ixs)
CircsSat        ⊢ ∀ cd ixs ixc
                  • CircsSat cd ixs ixc
                    ⇔ (∀ n
                    • n ∈ IxDom ixc
                      ⇒ CircSat
                        (ValueOf (CSigs cd n))
                        ixs
                        (ValueOf (ixc n)))
EqSat           ⊢ ∀ ixs n m
                  • EqSat ixs (n, m)
                    ⇔ n ∈ IxDom ixs ∧ m ∈ IxDom ixs ∧ ixs n = ixs m
EqsSat          ⊢ ∀ ixs nms
                  • EqsSat ixs nms ⇔ (∀ nm• nm ∈ nms ⇒ EqSat ixs nm)
DiagCircRel     ⊢ ∀ cd isc c
                  • DiagCircRel cd isc c
                    ⇔ (∃ six
                    • CircsSat cd six isc
                      ∧ EqsSat six (CCons cd)
                      ∧ CircSat (CSig cd) six c)
Diag2Circ       ⊢ ∀ d c• Diag2Circ d c = (ϵ x• DiagCircRel d c x)
tr22tr1         ⊢ ∀ t2
                  • tr22tr1 t2
                    = (λ t n
                    • if n ∈ IxDom t2
                      then Value (ValueOf (t2 n) t)
```

```
                    else Undefined)
tr12tr2             ⊢ ∀ t1
                      • tr12tr2 t1
                        = (λ n
                        • if n ∈ IxDom (t1 (ϵ x• T))
                          then Value (λ t• ValueOf (t1 t n))
                          else Undefined)
InputSignals        ⊢ ∀ cd cn
                      • InputSignals cd cn
                        = (let csig = ValueOf (CSigs cd cn)
                          in let cins = FunImage (Cons cn) (Fst csig)
                            in let map = cins ◁ CCons cd
                              in λ is sn
                              • if Cons cn sn ∈ Dom map
                                then is (Cons cn sn)
                                else Undefined)
```

A.2 The Theory alloy

Parents

fixp *rbjmisc*

Constants

no	$rel \to BOOL$
lone	$rel \to BOOL$
one	$rel \to BOOL$
some	$rel \to BOOL$
$\$\sim_{\boldsymbol{a}}$	$rel \to rel$
$\$\hat{}_{\boldsymbol{a}}$	$rel \to rel$
$\$*$	$rel \to rel$
$\$.\!_{\boldsymbol{a}}$	$rel \to rel \to rel$
$\$\to_{\boldsymbol{a}}$	$rel \to rel \to rel$
ToyList	$rel \leftrightarrow (rel \times rel \times rel \times rel \times rel \times rel \times rel)$

Aliases

$\hat{}$	$\$\hat{}_a : rel \to rel$
.	$\$.\!_a : rel \to rel \to rel$
$\to$	$\$\to_a : rel \to rel \to rel$

Type Abbreviations

tuple	*tuple*
rel	*rel*

Fixity

Right Infix 310:

$\cdot_{\boldsymbol{a}}$ $\quad \to_{\boldsymbol{a}}$

Prefix 50: $\hat{}_{\boldsymbol{a}}$ $\quad *$ $\quad \sim_{\boldsymbol{a}}$

Definitions

no	$\vdash \forall\ r\bullet\ no\ r \Leftrightarrow r = \{\}$
lone	$\vdash \forall\ r\bullet\ lone\ r \Leftrightarrow (\exists\ x\bullet\ r \subseteq \{x\})$
one	$\vdash \forall\ r\bullet\ one\ r \Leftrightarrow (\exists\ x\bullet\ r = \{x\})$
some	$\vdash \forall\ r\bullet\ some\ r \Leftrightarrow (\exists\ x\bullet\ x \in r)$
$\sim_{\boldsymbol{a}}$	$\vdash \forall\ p\bullet\ (\sim_a\ p) = \{x \mid \exists\ y\ z\bullet\ x = [y;\ z] \wedge [z;\ y] \in p\}$
$\hat{}_{\boldsymbol{a}}$	$\vdash \forall\ p$ $\bullet\ \$\hat{}\ p$ $= \{x$ $\mid \exists\ y\ z\bullet\ x = [y;\ z] \wedge tc\ (\lambda\ v\ w\bullet\ [v;\ w] \in p)\ z\ y\}$
$*$	$\vdash \forall\ p\bullet\ (*\ p) = \$\hat{}\ p \cup \{x \mid \exists\ y\bullet\ x = [y;\ y]\}$
$\cdot_{\boldsymbol{a}}$	$\vdash \forall\ p\ q$ $\bullet\ p\ .\ q$

$= \{x$
$|\exists\ u\ v\ w\bullet\ u\ @\ [v] \in p \wedge [v]\ @\ w \in q \wedge x = u\ @\ w\}$

$\rightarrow_a$ $\vdash \forall\ p\ q\bullet\ (p \rightarrow q) = \{x|\exists\ u\ w\bullet\ u \in p \wedge w \in q \wedge x = u\ @\ w\}$

ToyList $\vdash$ *ToyList*
$= \{(list,\ nil,\ cons,\ thing,\ car,\ cdr,\ equivTo,$
$prefixes)$
$|list = cons \cup nil$
$\wedge\ no\ (cons \cap nil)$
$\wedge\ car \subseteq (cons \rightarrow thing)$
$\wedge\ (\forall\ a\bullet\ a \in cons \Rightarrow one\ (\{a\}\ .\ car))$
$\wedge\ cdr \subseteq (cons \rightarrow list)$
$\wedge\ (\forall\ a\bullet\ a \in cons \Rightarrow one\ (\{a\}\ .\ cdr))$
$\wedge\ (\forall\ a$
$\bullet\ a \in list \Rightarrow (\exists\ e\bullet\ e \in nil \wedge e \in \{a\}\ .\ \$\hat{}\ cdr))$
$\wedge\ equivTo \subseteq (list \rightarrow list)$
$\wedge\ (\forall\ a\ b$
$\bullet\ a \in list \wedge b \in list$
$\Rightarrow (\{a\} \subseteq \{b\}\ .\ equivTo$
$\Leftrightarrow \{a\}\ .\ car = \{b\}\ .\ car$
$\wedge\ \{a\}\ .\ cdr\ .\ equivTo$
$= \{b\}\ .\ cdr\ .\ equivTo))$
$\wedge\ prefixes \subseteq (list \rightarrow list)$
$\wedge\ (\forall\ e\ a$
$\bullet\ e \in nil \wedge a \in list \Rightarrow \{e\} \subseteq \{a\}\ .\ prefixes)$
$\wedge\ (\forall\ e\ a$
$\bullet\ e \in nil \wedge a \in cons \Rightarrow \neg\ \{a\} \subseteq \{e\}\ .\ prefixes)$
$\wedge\ (\forall\ a\ b$
$\bullet\ a \in cons \wedge b \in cons$
$\Rightarrow (\{a\} \subseteq \{b\}\ .\ prefixes$
$\Leftrightarrow \{a\}\ .\ car = \{b\}\ .\ car$
$\wedge\ \{a\}\ .\ cdr \subseteq \{b\}\ .\ cdr\ .\ prefixes))\}$

Bibliography

[1] Roger Bishop Jones. Introduction to Work in Progress. *RBJones.com*, 2010. http://www.rbjones.com/rbjpub/pp/doc/t000.pdf.

[2] Emina Torlak. A Constraint Solver for Software Engineering: Finding Models and Cores of Large Relational Specifications. 2009. http://alloy.mit.edu/kodkod/pubs/emina-phd-thesis.pdf.

Index