Using HOL to study *Sugar 2.0* semantics

Michael J. C. Gordon

University of Cambridge Computer Laboratory
William Gates Building, JJ Thomson Avenue, Cambridge CB3 0FD, U.K.
`mjcg@cl.cam.ac.uk` `http://www.cl.cam.ac.uk/~mjcg`

July 5, 2002

Abstract. The Accellera standards-promoting organisation selected *Sugar 2.0*, IBM's formal specification language, as a standard that it says "will drive assertion-based verification". *Sugar 2.0* combines aspects of Interval Temporal Logic (ITL), Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) into a property language suitable for both formal verification and use with simulation test benches. As industrial strength languages go it is remarkably elegant, consisting of a small kernel conservatively extended by numerous definitions or 'syntactic sugar' (hence the name).
We are constructing a semantic embedding of *Sugar 2.0* in the version of higher order logic supported by the HOL system. To 'sanity check' the semantics we tried to prove some simple properties and as a result a few small bugs were discovered. We hope eventually to obtain a formal semantics that, with high confidence, matches the official 'golden' semantics issued by Accellera.
We are contemplating a variety of applications of the semantics, including building a semantics-directed Sugar model checker inside HOL. We also hope to investigate generating checkers by executing proof scripts that rewrite the semantics of particular constructs into an executable form. In the longer term we want to investigate the use of theorem proving to reason about models with infinite state spaces, which might involve developing extensions of *Sugar 2.0*.

1 Background on Accellera and Sugar

The Accellera organisation's website has their mission statement:

> *To improve designers' productivity, the electronic design industry needs a methodology based on both worldwide standards and open interfaces. Accellera was formed in 2000 through the unification of Open Verilog International and VHDL International to focus on identifying new standards, development of standards and formats, and to foster the adoption of new methodologies.*
> *Accellera's mission is to drive worldwide development and use of standards required by systems, semiconductor and design tools companies, which enhance a language-based design automation process. Its Board of Directors guides all the operations and activities of the organisation and is comprised of representatives from ASIC manufacturers, systems companies and design tool vendors.*

Faced with several syntactically and semantically incompatible formal property languages, Accellera initiated a process of selecting a standard property language to "drive assertion-based verification".
Four contributions were initially considered

- Motorola's CBV language;
- IBM's Sugar (the language of its RuleBase FV toolset);
- Intel's ForSpec;
- Verisity's *e* language (the language of the Specman Elite test-bench).

After a combination of discussion and voting, some details of which can be viewed on the web[1], attention was narrowed down to Sugar and CBV, and then in April 2002 a vote[2] selected IBM's submission, *Sugar 2.0*. *Sugar 2.0* is primarily an LTL-based language that is a successor to the CTL-based *Sugar 1* [1]. A key idea of both languages is the use of ITL-like [4] constructs called *Sugar Extended Regular Expressions*. *Sugar 2.0*

[1] http://www.eda.org/vfv/hm/
[2] http://www.eda.org/vfv/hm/0795.html

retains CTL constructs in its *Optional Branching Extension* (OBE), but this is de-emphasised in the defining document.

Besides moving from CTL to LTL, *Sugar 2.0* supports clocking and finite paths. Clocking allows one to specify on which clock edges signals are sampled. The finite path semantics allows properties to be interpreted on simulation runs, as in test-bench tools like Vera and Specman[3]

The addition of clocking and finite path semantics makes the *Sugar 2.0* semantics more than twice as complicated as the *Sugar 1* semantics. However, for a real 'industry standard' language *Sugar 2.0* is still remarkably simple, and it was routine to define the abstract syntax and semantics of the whole language in the logic of the HOL system [3].

In Section 2 we discuss the point of embedding Sugar in HOL. In Section 3, semantic embedding is reviewed and illustrated on simplified semantics of fragments of *Sugar 2.0*. In Section 4, the semantics of full *Sugar 2.0* is discussed, including finite paths and clocking. Due to space limitations, the complete semantics of *Sugar 2.0* is not given here, but can be found on the web.[4] In Section 5, progress so far in analysing the semantics using the HOL system is discussed. Finally, there is a short section of conclusions.

2 Why embed Sugar in HOL?

There are several justifications for the work described here. This project started in April 2002 and its goals are still being defined. Current motivations include the following.

2.1 Sanity checking and proving meta-theorems

By formalising the semantics and passing it through a parser and type-checker one achieves a first level of sanity checking of the definition. One also exposes possible ambiguities, fuzzy corner cases etc (e.g. see Section 4.2). The process is also very educational for the formaliser and a good learning exercise.

There are a number of meta-theorems one might expect to be true, and proving them with a theorem prover provides a further and deeper kind of sanity checking. In the case of *Sugar 2.0*, such meta-theorems include showing that expected simplifications to the semantics occur if there is no non-trivial clocking, that different semantics of clocking are equivalent and that if finite paths are ignored then the standard 'text-book semantics' results. Such meta-theorems are generally mathematically shallow, but full of tedious details – i.e. ideal for automated theorem proving. See Section 5 for what we have proved so far.

2.2 Validating definitional extensions

A key feature of the Sugar approach – indeed the feature from which the name "Sugar" is derived – is to have a minimal kernel augmented with a large number of definitions – i.e. syntactic sugar – to enhance the usability (but not the expressive power) of the language.

The definitions can be validated by proving that they achieve the correct semantics. See the end of Section 5.3 for some examples.

2.3 Machine processable semantics

The current *Sugar 2.0* document is admirably clear, but it is informal mathematics presented as typeset text. Tool developers could benefit from a machine readable version. One might think of using some standard representation of mathematical content, like MathML[5], however there is currently not much mathematically sophisticated tool support for such XML-based representations. See the end of Section 5.4 for a bit more discussion.

Higher order logic is a widely used formalisation medium (versions of higher order logic are used by HOL, Isabelle/HOL, PVS, NuPrl and Coq) and the semantic embedding of model-checkable logics in HOL is standard [6, 5]. Once one has a representation in higher order logic, then representations in other formats should be straightforward to derive.

[3] There is a 'Sugar2e' tool available from NoBug Consulting.

[4] http://www.cl.cam.ac.uk/~mjcg/Sugar/

[5] http://www.w3.org/Math/

2.4 Basis for research

We hope to develop semantically-based reasoning and checking infrastructure in HOL to support *Sugar 2.0*, and a prerequisite for this is to have a 'golden semantics' to which application-specific semantics can be proved equivalent.

We are interested in the development of property languages that support data operations and variables ranging over infinite data-types like numbers (e.g. including reals and complex numbers for DSP applications). Some sort of mixture of Hoare Logic and *Sugar 2.0* is being contemplated. Incrementally developing constructs by extending an existing semantics of *Sugar 2.0* is a way to ensure some backward compatibility with industry-standard language. Also, we might wish to prove sanity checking meta-theorem about our extended language, e.g. that it collapses to *Sugar 2.0* when there are no infinite types.

Sugar 2.0 is explicitly designed for use with simulation as well as formal verification. We are interested in using the HOL platform to experiment with combinations of execution, checking and theorem-proving. To this end we are thinking about implementing tools to transform properties stated in Sugar to checking automata. This is inspired by IBM's FoCs project[6], but uses compilation by theorem proving to ensure semantic equivalence between the executable checker and the source property.

2.5 Education

Both semantic embedding and property specification are taught as part of the Computer Science undergraduate course at Cambridge University, and being able to illustrate the ideas on a real example like *Sugar 2.0* is pedagogically valuable. Teaching an industrial property language nicely complements and motivates academic languages like ITL, LTL and CTL.

The semantic embedding of *Sugar 2.0* in the HOL system is an interesting case study. It illustrates some issues in making total functional definitions, and the formal challenges attempted so far provide insight into how to perform structural induction using the built-in tools. Thus *Sugar 2.0* has educational potential for training HOL users. In fact, the semantics described in this paper is an example distributed with HOL.[7]

3 Review of semantic embedding in higher order logic

Higher order logic is an extension of first-order predicate calculus that allows quantification over functions and relations. It is a natural notation for formalising informal set theoretic specifications (indeed, it is usually more natural than formal first-order set theories, like ZF). We hope that the HOL notation we use in what follows is sufficiently close to standard informal mathematics that it needs no systematic explanation.

We use Church's λ-notation for denoting functions: a 'lambda-term' like $\lambda x.\ t$, where x is a variable and t a term, denotes the function that maps a value v to the result of substituting v for the variable x in t (the infix notation $x \mapsto t$ is sometimes used instead of $\lambda x.\ t$). If P is a function that returns a truth-value (i.e. a predicate), then P can be thought of a set, and we write $x \in P$ to mean $P(x)$ is true. Note that $\lambda x.\ \cdots x \cdots$ corresponds to the set abstraction $\{x \mid \cdots x \cdots\}$. We write $\forall x \in P.\ Q(x)$, $\exists x \in P.\ Q(x)$ to mean $\forall x.\ P(x) \Rightarrow Q(x)$, $\exists x.\ P(x) \wedge Q(x)$, respectively.

To embed[8] a language in HOL one first defines constructors for all the syntactic constructs of the language. This is the 'abstract syntax' and provides a representation of parse trees as terms in the logic. The semantics is then specified by defining a semantic function that recursively maps each construct to a representation of its meaning.

For *Sugar 2.0*, a model $\mathtt{M}$ is a quintuple $(\mathtt{S}_\mathtt{M}, \mathtt{S0}_\mathtt{M}, \mathtt{R}_\mathtt{M}, \mathtt{P}_\mathtt{M}, \mathtt{L}_\mathtt{M})$, where $\mathtt{S}_\mathtt{M}$ is a set of states, $\mathtt{S0}_\mathtt{M}$ is the subset of initial states, $\mathtt{R}_\mathtt{M}$ is a transition relation (so $\mathtt{R}_\mathtt{M}(s, s')$ means s' is a possible successor state to s), $\mathtt{P}_\mathtt{M}$ is a set of atomic propositions, and $\mathtt{L}_\mathtt{M}$ is a valuation that maps a state to the set of atomic propositions that hold at the state (so $\mathtt{L}_\mathtt{M}\ s\ \mathtt{p}$ is true iff atomic proposition $\mathtt{p}$ is true in state s).

[6] http://www.haifa.il.ibm.com/projects/verification/focs/
[7] http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/hol/hol98/examples/Sugar2/
[8] We shall only be concerned with so called 'deep embeddings' here [2].

3.1 Boolean expressions in Sugar

The syntax of boolean expressions (ranged over by b, b_1, b_2 etc.) is built from atomic propositions (ranged over by p) using negation ($\neg$) and conjunction ($\wedge$):

$$
\begin{array}{lll}
\mathtt{b} ::= & \mathtt{p} & \text{(Atomic formula)} \\
 & \mid \neg\mathtt{b} & \text{(Negation)} \\
 & \mid \mathtt{b}_1 \wedge \mathtt{b}_2 & \text{(Conjunction)}
\end{array}
$$

This is defined in HOL by a recursive type definition of a type that represents the syntax of boolean expressions. Other boolean expressions are added via definitions (e.g. see Section 5.3 for the definition of disjunction: $\mathtt{b}_1 \vee \mathtt{b}_2$).

Let l range over predicates on $\mathtt{P_M}$, called "truth assignments" in the Sugar documentation. The semantics of boolean expressions is given by defining a semantic function B_SEM such that B_SEM M l b if true iff b is built from propositions in $\mathtt{P_M}$ and it is true with respect to the truth assignment l.

If we write (M, l $\models$ b) for B_SEM M l b then the semantics is given by

$$
\begin{array}{l}
((\mathtt{M},\ \mathtt{l} \models \mathtt{p}) \quad = \mathtt{p} \in \mathtt{P_M} \wedge \mathtt{p} \in \mathtt{l}) \\
\wedge \\
((\mathtt{M},\ \mathtt{l} \models \mathtt{T}) \quad = \mathtt{T}) \\
\wedge \\
((\mathtt{M},\ \mathtt{l} \models \neg\mathtt{b}) \quad = \neg(\mathtt{M},\ \mathtt{l} \models \mathtt{b})) \\
\wedge \\
((\mathtt{M},\ \mathtt{l} \models \mathtt{b1} \wedge \mathtt{b2}) = (\mathtt{M},\ \mathtt{l} \models \mathtt{b1}) \wedge (\mathtt{M},\ \mathtt{l} \models \mathtt{b2}))
\end{array}
$$

Note that the symbol $\wedge$ is overloaded: the first occurrence in the equation above is part of the boolean expression syntax of Sugar, but the second occurrence is higher order logic.

Before looking at the full semantics of *Sugar 2.0*, we first consider a simplified semantics in which there is no clocking, and paths are always infinite. We consider separately the parts of *Sugar 2.0* corresponding to Interval Temporal Logic (ITL), Linear Temporal Logic (LTL) and Computation Tree Logic (CTL).

3.2 ITL: Sugar Extended Regular Expressions (SEREs)

Interval Temporal Logic (ITL) provides formulas that are true or false of intervals of states. Here we just consider finite intervals, though recent formulations of ITL[9] allow intervals to be infinite. For Sugar we only need to consider ITL formulas, as there are no constructs corresponding to ITL expressions (expressions map intervals to values). Providing more elaborate ITL constructs in Sugar strikes us as an interesting research topic.

The Sugar subset corresponding to ITL is called *Sugar Extended Regular Expressions* (SEREs). If r, r_1, r_2 etc. range over SEREs and p ranges over the set $\mathtt{P_M}$ of atomic propositions, then the syntax is given by:

$$
\begin{array}{lll}
\mathtt{r} ::= & \mathtt{p} & \text{(Atomic formula)} \\
 & \mid \{\mathtt{r}_1\}\ \mathtt{|}\ \{\mathtt{r}_2\} & \text{(Disjunction)} \\
 & \mid \mathtt{r}_1\ \mathtt{;}\ \mathtt{r}_2 & \text{(Concatenation)} \\
 & \mid \mathtt{r}_1\ \mathtt{:}\ \mathtt{r}_2 & \text{(Fusion: ITL's chop)} \\
 & \mid \{\mathtt{r}_1\}\ \mathtt{\&\&}\ \{\mathtt{r}_2\} & \text{(Length matching conjunction)} \\
 & \mid \{\mathtt{r}_1\}\ \mathtt{\&}\ \{\mathtt{r}_2\} & \text{(Flexible matching conjunction)} \\
 & \mid \mathtt{r}[*] & \text{(Repeat)}
\end{array}
$$

The semantics of SEREs is given by defining a semantic function S_SEM such that S_SEM M w r if true iff w is in the language of the extended regular expression r. We write (M, w $\models$ r) for S_SEM M w r.

If wlist is a list of lists then Concat wlist is the concatenation of the lists in wlist and if P is some predicate then Every P wlist means that P(w) holds for every w in wlist.

The semantics S_SEM M w r is defined in HOL by recursion on r.

[9] http://www.cms.dmu.ac.uk/~cau/itlhomepage/

```
((M, w ⊨ b) =
  ∃l. (w = [l]) ∧ (M, l ⊨ b))
∧
((M, w ⊨ r1;r2) =
  ∃w1 w2. (w = w1 w2) ∧ (M, w1 ⊨ r1) ∧ (M, w2 ⊨ r2))
∧
((M, w ⊨ r1:r2) =
  ∃w1 w2 l. (w = w1 [l] w2) ∧
            (M, (w1 [l]) ⊨ r1) ∧ (M, ([l] w2) ⊨ r2))
∧
((M, w ⊨ {r1}|{r2}) =
  (M, w ⊨ r1) ∨ (M, w ⊨ r2))
∧
((M, w ⊨ {r1}&&{r2}) =
  (M, w ⊨ r1) ∧ (M, w ⊨ r2))
∧
((M, w ⊨ {r1}&{r2}) =
  ∃w1 w2. (w = w1 w2) ∧
          (((M, w ⊨ r1) ∧ (M, w1 ⊨ r2))
           ∨
           ((M, w ⊨ r2) ∧ (M, w1 ⊨ r1))))
∧
((M, w ⊨ r[*]) =
  ∃wlist. (w = Concat wlist) ∧ Every (λw. (M, w ⊨ r)) wlist)
```

This definition is manifestly primitive-recursive, and so is automatically proved total by HOL [7]. The intuitive semantics of SERE's is explained in the *Sugar 2.0* documentation [8].

3.3 LTL: Sugar Foundation Language (FL)

Sugar 2.0 has a kernel combining standard LTL notation with a less standard `abort` operation and some constructs using SEREs. The suffix "`!`" found on some constructs indicates that these are 'strong' (i.e. liveness-enforcing) operators. The distinction between strong and weak operators is discussed and motivated in the *Sugar 2.0* literature (e.g. [9, Section 4.11]).

$\mathtt{f} ::= \mathtt{p}$	(Atomic formula)
$\mid \neg\mathtt{f}$	(Negation)
$\mid \mathtt{f}_1 \wedge \mathtt{f}_2$	(Conjunction)
$\mid \mathtt{X!f}$	(Successor)
$\mid [\mathtt{f}_1\ \mathtt{U}\ \mathtt{f}_2]$	(Until)
$\mid \{\mathtt{r}\}(\mathtt{f})$	(Suffix implication)
$\mid \{\mathtt{r}_1\}\ \mathtt{\vert\text{-}>}\ \{\mathtt{r}_2\}\mathtt{!}$	(Strong suffix implication)
$\mid \{\mathtt{r}_1\}\ \mathtt{\vert\text{-}>}\ \{\mathtt{r}_2\}$	(Weak suffix implication)
$\mid \mathtt{f\ abort\ b}$	(Abort)

Numerous additional notations are introduced as syntactic sugar. These are easily formalised as definitions in HOL. Some examples are given in Section 5.3.

Being LTL, the semantics of FL formulas is defined with respect to a path π, which (in the simplified semantics here) is a function from the natural numbers to states.

We define a semantic function `F_SEM` such that `F_SEM M` π `f` means FL formula `f` is true of path π. We write (`M`, π ⊨ `r`) for `F_SEM M` π `f`.

Note that in the semantics below it is not assumed that paths π are necessarily computations of `M` (i.e. satisfy `Path M` π, as defined in Section 3.4). This is important for the `abort` construct (where the $\exists\pi'$ quantifies over all paths).

The notation π_i denotes the i-th state in the path (i.e. $\pi(i)$); π^i denotes the 'i-th tail' of π – the path obtained by chopping i elements off the front of π (i.e. $\pi^i = \lambda n.\ \pi(n{+}i)$); $\pi^{(i,j)}$ denotes the finite sequence of states from i to j in π, i.e. $\pi_i\pi_{i+1}\cdots\pi_j$. The juxtaposition $\pi^{(i,j)}\pi'$ denotes the path obtained by concatenating the finite sequence $\pi^{(i,j)}$ on to the front of the path π'.
The function $\hat{\mathtt{L}}_\mathtt{M}$ denotes the point-wise extension of $\mathtt{L}_\mathtt{M}$ to finite sequences of states (i.e. `MAP` $\mathtt{L}_\mathtt{M}$ in HOL and functional programming notation).
The definition of `F_SEM M` π `f` is by recursion on `f`.

$$
\begin{array}{l}
((\mathtt{M},\ \pi\ \models\ \mathtt{b}) = (\mathtt{M},\ \mathtt{L}_\mathtt{M}(\pi_0)\ \models\ \mathtt{b}))\\
\wedge\\
((\mathtt{M},\ \pi\ \models\ \neg\mathtt{f}) = \neg(\mathtt{M},\ \pi\ \models\ \mathtt{f}))\\
\wedge\\
((\mathtt{M},\ \pi\ \models\ \mathtt{f1} \wedge \mathtt{f2}) = (\mathtt{M},\ \pi\ \models\ \mathtt{f1})\ \wedge\ (\mathtt{M},\ \pi\ \models\ \mathtt{f2}))\\
\wedge\\
((\mathtt{M},\ \pi\ \models\ \mathtt{X!\,f}) = (\mathtt{M},\ \pi^1\ \models\ \mathtt{f}))\\
\wedge\\
((\mathtt{M},\ \pi\ \models\ [\mathtt{f1\ U\ f2}]) =\\
\quad \exists\mathtt{k}.\ (\mathtt{M},\ \pi^\mathtt{k}\ \models\ \mathtt{f2})\ \wedge\ \forall\mathtt{j}.\ \mathtt{j} < \mathtt{k} \Rightarrow (\mathtt{M},\ \pi^\mathtt{j}\ \models\ \mathtt{f1}))\\
\wedge\\
((\mathtt{M},\ \pi\ \models\ \{\mathtt{r}\}(\mathtt{f})) =\\
\quad \forall\mathtt{j}.\ (\mathtt{M},\ (\hat{\mathtt{L}}_\mathtt{M}\ (\pi^{(0,\mathtt{j})}))\ \models\ \mathtt{r}) \Rightarrow (\mathtt{M},\ \pi^\mathtt{j}\ \models\ \mathtt{f}))\\
\wedge\\
((\mathtt{M},\ \pi\ \models\ \{\mathtt{r1}\}\mathtt{|\text{-}>}\{\mathtt{r2}\}\mathtt{!}) =\\
\quad \forall\mathtt{j}.\ (\mathtt{M},\ (\hat{\mathtt{L}}_\mathtt{M}\ (\pi^{(0,\mathtt{j})}))\ \models\ \mathtt{r1})\\
\qquad\quad \Rightarrow \exists\mathtt{k}.\ \mathtt{j} \le \mathtt{k} \wedge (\mathtt{M},\ (\hat{\mathtt{L}}_\mathtt{M}\ (\pi^{(\mathtt{j},\mathtt{k})}))\ \models\ \mathtt{r2}))\\
\wedge\\
((\mathtt{M},\ \pi\ \models\ \{\mathtt{r1}\}\mathtt{|\text{-}>}\{\mathtt{r2}\}) =\\
\quad \forall\mathtt{j}.\ (\mathtt{M},\ (\hat{\mathtt{L}}_\mathtt{M}\ (\pi^{(0,\mathtt{j})}))\ \models\ \mathtt{r1})\\
\qquad\quad \Rightarrow (\exists\mathtt{k}.\ \mathtt{j} \le \mathtt{k} \wedge (\mathtt{M},\ (\hat{\mathtt{L}}_\mathtt{M}\ (\pi^{(\mathtt{j},\mathtt{k})}))\ \models\ \mathtt{r2}))\\
\qquad\qquad \vee\\
\qquad\qquad \forall\mathtt{k}.\ \mathtt{j} \le \mathtt{k} \Rightarrow \exists\mathtt{w}.\ (\mathtt{M},\ (\hat{\mathtt{L}}_\mathtt{M}\ (\pi^{(\mathtt{j},\mathtt{k})}))\mathtt{w}\ \models\ \mathtt{r2}))\\
\wedge\\
((\mathtt{M},\ \pi\ \models\ \mathtt{f\ abort\ b}) =\\
\quad ((\mathtt{M},\ \pi\ \models\ \mathtt{f}\ )\\
\quad\ \vee\\
\quad\ \exists\mathtt{j}\ \pi'.\ (\mathtt{M},\ \pi^\mathtt{j}\ \models\ \mathtt{b})\ \wedge\ (\mathtt{M},\ \pi^{(0,\mathtt{j}-1)}\pi'\ \models\ \mathtt{f})))
\end{array}
$$

In this semantics, paths π are infinite, as in the classical semantics of LTL for model checking. A version that also handles finite paths, suitable for evaluation on simulation runs, is given in Section 4.2.

3.4 CTL: Sugar Optional Branching Extension (OBE)

The syntax of the *Sugar 2.0* OBE is completely standard. The syntax of the OBE formulas is:

$\mathtt{f} ::= \mathtt{p}$	(Atom)
$\mid \neg\mathtt{f}$	(Negation)
$\mid \mathtt{f}_1 \wedge \mathtt{f}_2$	(Conjunction)
$\mid \mathbf{EX}\mathtt{f}$	(Some successors)
$\mid \mathbf{E}[\mathtt{f}_1\ \mathbf{U}\ \mathtt{f}_2]$	(Until – along some path)
$\mid \mathbf{EG}\mathtt{f}$	(Always on some path)

For the semantics, define `Path M` π to be true iff π is a computation of `M`:

```
Path M π = ∀n. R_M(π_n, π_{n+1})
```

The semantic function O_SEM is defined so that O_SEM M s f is true iff f is true of M at state s. Write (M, s $\models$ f) for O_SEM M s f, which is defined by recursion on f by:

$$
\begin{array}{l}
((\mathtt{M},\ \mathtt{s} \models \mathtt{b}) = (\mathtt{M},\ \mathtt{L_M(s)} \models \mathtt{b})) \\
\wedge \\
((\mathtt{M},\ \mathtt{s} \models \neg\mathtt{f}) = \neg(\mathtt{M},\ \mathtt{s} \models \mathtt{f})) \\
\wedge \\
((\mathtt{M},\ \mathtt{s} \models \mathtt{f1} \wedge \mathtt{f2}) = (\mathtt{M},\ \mathtt{s} \models \mathtt{f1}) \wedge (\mathtt{M},\ \mathtt{s} \models \mathtt{f2})) \\
\wedge \\
((\mathtt{M},\ \mathtt{s} \models \mathtt{EX\ f}) = \\
\quad \exists\pi.\ \mathtt{Path\ M}\ \pi \wedge (\pi_0 = \mathtt{s}) \wedge (\mathtt{M},\ \pi_1 \models \mathtt{f})) \\
\wedge \\
((\mathtt{M},\ \mathtt{s} \models [\mathtt{f1\ U\ f2}]) = \\
\quad \exists\pi.\ \mathtt{Path\ M}\ \pi \wedge (\pi_0 = \mathtt{s}) \wedge \\
\qquad (\mathtt{M},\ \pi_\mathtt{k} \models \mathtt{f2}) \wedge \forall \mathtt{j}.\ \mathtt{j} < \mathtt{k} \Rightarrow (\mathtt{M},\ \pi_\mathtt{j} \models \mathtt{f1})) \\
\wedge \\
((\mathtt{M},\ \mathtt{s} \models \mathtt{EG\ f}) = \\
\quad \exists\pi.\ \mathtt{Path\ M}\ \pi \wedge (\pi_0 = \mathtt{s}) \wedge \forall \mathtt{j}.\ (\mathtt{M},\ \pi_\mathtt{j} \models \mathtt{f}))
\end{array}
$$

4 Full *Sugar 2.0* semantics in higher order logic

The full *Sugar 2.0* language extends the constructs described above with the addition of clocking and support for finite paths.

The clocking constructs allow (possibly multiple) clocks to be declared, see Section 4.1. Clocks define when signals are sampled, so the next value of a signal s with respect to a clock c is the value of s at the next rising edge of !c.

Simulators compute finite executions of a model, so to support checking whether a property holds over such a simulation run, *Sugar 2.0* defines the meaning of each construct on both finite and infinite paths.

Adding clocks and finite paths greatly complicates the language, though it is still surprisingly elegant.

We have formalised the full semantics of *Sugar 2.0* via a deep embedding in higher order logic. Corresponding to Appendix A.1 of the *Sugar 2.0* specification submitted to Accellera [9] we have defined types bexp, sere, fl and obe in the HOL logic to represent the syntax of Boolean Expressions, Sugar Extended Regular Expressions (SEREs), formulas of the Sugar Foundation Language (FL) and formulas of the Optional Branching Extension (OBE), respectively.

Corresponding to Appendix A.2 of the Sugar documentation we have defined semantic functions B_SEM, S_SEM, F_SEM and O_SEM that interpret boolean expressions, SEREs, FL formulas and OBE formulas, respectively.

Due to space constraints we do not give the semantics here, but full details are available on the web at:

http://www.cl.cam.ac.uk/~mjcg/Sugar

The semantics is evolving and we hope to keep the HOL version up to date with respect to the official version. In the next two sub-sections we discuss clocking and finite paths.

4.1 Clocking

If b is a boolean expression, then the SERE b@c recognises a sequence of states in which b is true on the next rising edge of c. Thus b@c behaves like {¬c[*]; c ∧ b}.

More generally, if r is a SERE and c a variable then r@c is a SERE in which all variables inside r are clocked with respect to the rising edges of c.

The semantics of clocked SEREs can be given in two ways:

1. by making a clocking context part of the semantic function, i.e. defining $(\mathtt{M},\ \mathtt{w} \overset{\mathtt{c}}{\models} \mathtt{r})$ instead of the unclocked $(\mathtt{M},\ \mathtt{w} \models \mathtt{r})$;
2. by translating clocked SEREs into unclocked SEREs using rewriting rules.

With the first approach (1), which is taken as the definition in the Accellera report, one defines

$$
\begin{array}{l}
(\mathtt{M},\ \mathtt{w} \overset{\mathtt{c}}{\models} \mathtt{b}) = \\
\quad \exists \mathtt{n}.\ \mathtt{n} \geq 1 \hfill \wedge \\
\qquad (\mathrm{length}\ \mathtt{w} = \mathtt{n}) \hfill \wedge \\
\qquad (\forall \mathtt{i}.\ 1 \leq \mathtt{i} \wedge \mathtt{i} < \mathtt{n} \Rightarrow (\mathtt{M},\ \mathtt{w}_{\mathtt{i}-1} \models \neg \mathtt{c}) \wedge \\
\qquad (\mathtt{M},\ \mathtt{w}_{\mathtt{n}-1} \models \mathtt{c} \wedge \mathtt{b})
\end{array}
$$

$$(\mathtt{M},\ \mathtt{w} \overset{\mathtt{c}}{\models} \mathtt{r@c1}) = (\mathtt{M},\ \mathtt{w} \overset{\mathtt{c1}}{\models} \mathtt{r})$$

together with equations like those in Section 3.2, but with $\overset{\mathtt{c}}{\models}$ replacing $\models$. Notice that an inner clock overrides an outer clock (i.e. `c1` is used to clock variables inside `r` in `r@c1`: the clock context `c` is overridden by `c1` inside `r`).

The second approach (2) is to translate clocked SEREs to unclocked SEREs using rewrites

$$
\begin{array}{lcl}
\mathtt{b@c} & \longrightarrow & \{\neg\mathtt{c[*];\ c}\wedge\mathtt{b}\} \\
\{\mathtt{r1;r2}\}\mathtt{@c} & \longrightarrow & \{\mathtt{r1@c}\}\mathtt{;}\{\mathtt{r2@c}\} \\
\{\mathtt{r1:r2}\}\mathtt{@c} & \longrightarrow & \{\mathtt{r1@c}\}\mathtt{:}\{\mathtt{r2@c}\} \\
\{\{\mathtt{r1}\}\mathtt{|}\{\mathtt{r2}\}\}\mathtt{@c} & \longrightarrow & \{\mathtt{r1@c}\}\mathtt{|}\{\mathtt{r2@c}\} \\
\{\{\mathtt{r1}\}\mathtt{\&\&}\{\mathtt{r2}\}\}\mathtt{@c} & \longrightarrow & \{\mathtt{r1@c}\}\mathtt{\&\&}\{\mathtt{r2@c}\} \\
\{\{\mathtt{r1}\}\mathtt{\&}\{\mathtt{r2}\}\}\mathtt{@c} & \longrightarrow & \{\mathtt{r1@c}\}\mathtt{\&}\{\mathtt{r2@c}\} \\
\mathtt{r[*]@c} & \longrightarrow & \{\mathtt{r@c}\}\mathtt{[*]} \\
\mathtt{r@c1@c} & \longrightarrow & \mathtt{r1@c1}
\end{array}
$$

these rewrites cannot be taken as equational definitions, but need to be applied from the outside in: e.g. one must rewrite `b@c1@c` to `b@c1` (eliminating `c`) rather than rewriting the sub-term `b@c1` first, resulting in $\{\neg\mathtt{c1[*];\ c1}\wedge\mathtt{b}\}\mathtt{@c}$. We have proved the two semantics for clocking SEREs are equivalent, see Section 5.3.

One can also clock formulas, `f@c`, and there may be several clocks. Consider: [10]

```
G(req_in -> X!(req_out@cb))@ca
```

this means that the entire formula is clocked on clock `ca`, except that signal `req_out` is clocked on `cb`. Clocks do not 'accumulate', so the signal `req_out` is only clocked by `cb`, not by both clocks. Thus `cb` 'protects' `req_out` from the main clock, `ca`, i.e.:

```
req_out@cb@ca = req_out@cb
```

As with the clocking of SEREs, this meaning of clocking prevents us simply defining:

```
req_out@cb = [¬cb U (cb ∧ req_out)]
```

since if this were the definition of `req_out@cb` then we would be forced to have:

```
req_out@cb@ca = [¬cb U (cb ∧ req_out)]@ca
```

when we actually want

```
req_out@cb@ca = req_out@cb
```

Thus, as with SEREs, we cannot just rewrite away clocking constructs using equational reasoning, but if one starts at the outside and works inwards, then one can systematically compile away clocking. The rules for doing this are given in the *Sugar 2.0* Accellera documentation as part of the implementation of formal verification [9, Appendix B.1]. We are currently in the process of trying to validate the clocking rewrites, see Section 5.3.

[10] The discussion of clocking here is based on email communication with Cindy Eisner.

The official semantics uses the approach – like (1) above – of having the currently active clock as an argument to the semantic function for formulas. In fact two semantics are given: one for 'weak' clocking and one for 'strong' clocking. The weak clocking is specified in HOL by defining

$(\mathtt{M},\ \pi\ \overset{\mathtt{c}}{\models}\ \mathtt{f})$

and the strong clocking by defining

$(\mathtt{M},\ \pi\ \overset{\mathtt{c!}}{\models}\ \mathtt{f})$

We shall not give the complete semantics here (they are available on the web), but just show the semantics of boolean expressions `b`:

$$((\mathtt{M},\ \pi\ \overset{\mathtt{c}}{\models}\ \mathtt{b}) = \\ \forall \mathtt{i} \in \mathtt{pl}\,\pi.\ (\mathtt{M},\ (\hat{\mathtt{L}}_{\mathtt{M}}\ (\pi^{(0,\mathtt{i})}))\ \overset{\mathrm{T}}{\models}\ \neg\mathtt{c[*];c})\ \Rightarrow\ (\mathtt{M},\ \mathtt{L}_{\mathtt{M}}(\pi_{\mathtt{i}})\ \models\ \mathtt{b}))$$

This says that *if* there is a first rising edge of `c` at time `i`, then `b` is true at `i`.

$$((\mathtt{M},\ \pi\ \overset{\mathtt{c!}}{\models}\ \mathtt{b}) = \\ \exists \mathtt{i} \in \mathtt{pl}\,\pi.\ (\mathtt{M},\ (\hat{\mathtt{L}}_{\mathtt{M}}\ (\pi^{(0,\mathtt{i})}))\ \overset{\mathrm{T}}{\models}\ \neg\mathtt{c[*];c})\ \wedge\ (\mathtt{M},\ \mathtt{L}_{\mathtt{M}}(\pi_{\mathtt{i}})\ \models\ \mathtt{b}))$$

This says that *there is* a first rising edge, and if it occurs at time `i`, then `b` is true at `i`.
Thus the strongly clocked semantics assumes the clock is 'live', but the weakly clocked semantics doesn't (compare the concepts of total and partial correctness).

4.2 Finite paths

Sugar 2.0 gives a semantics to formulas for both finite and infinite paths. To represent this, we model a path as being either a non-empty[11] finite list of states or a function from natural numbers to states and define a predicate `finite` to test if a path is a finite list. The function `length` gives the length of a finite path (it is not defined on paths for which `finite` is not true).
We interpret the official semantics locution

"for every $j <$ `length`(π): $\cdots\ j\ \cdots$"

as meaning

"for every j: (`finite` π implies $j <$ `length` π) implies $\cdots\ j\ \cdots$"

and we interpret the official semantics locution

"there exists $j <$ `length`(π) s.t. $\cdots\ j\ \cdots$"

as meaning

"there exists j s.t. (`finite` π implies $j <$ `length` π) and $\cdots\ j\ \cdots$"

Define `pl` π n to mean that if π is finite then n is less than the length of π, i.e. the predicate `pl` is defined by

$\mathtt{pl}\ \pi\ n\ =\ \mathtt{finite}\ \pi\ \Rightarrow\ n < \mathtt{length}\ \pi$

We can then write "$\forall i \in$ `pl` $\pi.\ \cdots\ i\ \cdots$" and "$\exists i \in$ `pl` $\pi.\ \cdots\ i\ \cdots$" for the locutions above. The name "`pl`" is short for "path length"
Here is a version of the unclocked FL semantics that allows paths to be finite.

$$((\mathtt{M},\ \pi\ \models\ \mathtt{b}) = (\mathtt{M},\ \mathtt{L}_{\mathtt{M}}(\pi_0)\ \models\ \mathtt{b})) \\ \wedge \\ ((\mathtt{M},\ \pi\ \models\ \neg\mathtt{f}) = \neg(\mathtt{M},\ \pi\ \models\ \mathtt{f})) \\ \wedge \\ ((\mathtt{M},\ \pi\ \models\ \mathtt{f1} \wedge \mathtt{f2}) = (\mathtt{M},\ \pi\ \models\ \mathtt{f1})\ \wedge\ (\mathtt{M},\ \pi\ \models\ \mathtt{f2})) \\ \wedge \\ ((\mathtt{M},\ \pi\ \models\ \mathtt{X!\,f}) = \mathtt{pl}\ \pi\ 1\ \wedge\ (\mathtt{M},\ \pi^1\ \models\ \mathtt{f}))$$

[11] The need for finite paths to be non-empty arose when trying to prove some properties. This requirement does not seem to be explicit in the Accellera specification.

$$
\begin{array}{l}
\wedge \\
((\mathtt{M},\ \pi\ \models\ \mathtt{[f1\ U\ f2]}) = \\
\quad \exists \mathtt{k} \in \mathtt{pl}\,\pi. \\
\quad\quad (\mathtt{M},\ \pi^{\mathtt{k}}\ \models\ \mathtt{f2}) \wedge \forall \mathtt{j} \in \mathtt{pl}\,\pi.\ \mathtt{j} < \mathtt{k} \Rightarrow (\mathtt{M},\ \pi^{\mathtt{j}}\ \models\ \mathtt{f1})) \\
\wedge \\
((\mathtt{M},\ \pi\ \models\ \{\mathtt{r}\}(\mathtt{f})) = \\
\quad \forall \mathtt{j} \in \mathtt{pl}\,\pi.\ (\mathtt{M},\ (\hat{\mathtt{L}}_{\mathtt{M}}\ (\pi^{(0,\mathtt{j})}))\ \models\ \mathtt{r}) \Rightarrow (\mathtt{M},\ \pi^{\mathtt{j}}\ \models\ \mathtt{f})) \\
\wedge \\
((\mathtt{M},\ \pi\ \models\ \{\mathtt{r1}\}\mathtt{|\text{->}}\{\mathtt{r2}\}\mathtt{!}) = \\
\quad \forall \mathtt{j} \in \mathtt{pl}\,\pi.\ (\mathtt{M},\ (\hat{\mathtt{L}}_{\mathtt{M}}\ (\pi^{(0,\mathtt{j})}))\ \models\ \mathtt{r1}) \\
\quad\quad\quad\quad \Rightarrow \exists \mathtt{k} \in \mathtt{pl}\,\pi.\ \mathtt{j} \leq \mathtt{k} \wedge (\mathtt{M},\ (\hat{\mathtt{L}}_{\mathtt{M}}\ (\pi^{(\mathtt{j},\mathtt{k})}))\ \models\ \mathtt{r2})) \\
\wedge \\
((\mathtt{M},\ \pi\ \models\ \{\mathtt{r1}\}\mathtt{|\text{->}}\{\mathtt{r2}\}) = \\
\quad \forall \mathtt{j} \in \mathtt{pl}\,\pi.\ (\mathtt{M},\ (\hat{\mathtt{L}}_{\mathtt{M}}\ (\pi^{(0,\mathtt{j})}))\ \models\ \mathtt{r1}) \\
\quad\quad\quad\quad \Rightarrow (\exists \mathtt{k} \in \mathtt{pl}\,\pi.\ \mathtt{j} \leq \mathtt{k} \wedge (\mathtt{M},\ (\hat{\mathtt{L}}_{\mathtt{M}}\ (\pi^{(\mathtt{j},\mathtt{k})}))\ \models\ \mathtt{r2})) \\
\quad\quad\quad\quad\quad \vee \\
\quad\quad\quad\quad\quad \forall \mathtt{k} \in \mathtt{pl}\,\pi.\ \mathtt{j} \leq \mathtt{k} \Rightarrow \exists \mathtt{w}.\ (\mathtt{M},\ (\hat{\mathtt{L}}_{\mathtt{M}}\ (\pi^{(\mathtt{j},\mathtt{k})}))\mathtt{w}\ \models\ \mathtt{r2})) \\
\wedge \\
((\mathtt{M},\ \pi\ \models\ \mathtt{f\ abort\ b}) = \\
\quad ((\mathtt{M},\ \pi\ \models\ \mathtt{f}\ ) \\
\quad\ \vee \\
\quad\ \exists \mathtt{j} \in \mathtt{pl}\,\pi. \\
\quad\quad \mathtt{0} < \mathtt{j} \wedge \exists \pi'.\ (\mathtt{M},\ \pi^{\mathtt{j}}\ \models\ \mathtt{b}) \wedge (\mathtt{M},\ \pi^{(0,\mathtt{j}-1)}\pi'\ \models\ \mathtt{f})))
\end{array}
$$

This semantics has evolved from an existing unpublished semantics[12] of unclocked FL formulas.

5 Progress on analysing the semantics

We have established a number of properties of the semantics using the HOL system. Some of these went through first time without any problems, but others revealed bugs both in the *Sugar 2.0* semantics and original HOL representation of the semantics.

5.1 Characterising adjacent rising edges

Define:

$$
\begin{array}{lll}
\mathtt{FirstRise\ M}\ \pi\ \mathtt{c\ i} & = & (\mathtt{M},\ (\hat{\mathtt{L}}_{\mathtt{M}}\ (\pi^{(0,\mathtt{i})}))\ \overset{\mathrm{T}}{\models}\ \neg\mathtt{c[*];c}) \\
\mathtt{NextRise\ M}\ \pi\ \mathtt{c\ (i,j)} & = & (\mathtt{M},\ (\hat{\mathtt{L}}_{\mathtt{M}}\ (\pi^{(\mathtt{i},\mathtt{j})}))\ \overset{\mathrm{T}}{\models}\ \neg\mathtt{c[*];c})
\end{array}
$$

The right hand sides of these definition occur in the *Sugar 2.0* semantics. We have proved that the definitions of `FirstRise` and `NextRise` give them the correct meaning, namely `FirstRise M` π `c i` is true iff `i` is the time of the first rising edge of `c`, and `NextRise M` π `c (i,j)` is true iff `j` is the time of the first rising edge of `c` after `i`.

$$
\begin{array}{l}
\vdash\ \mathtt{FirstRise\ M}\ \pi\ \mathtt{c\ i}\ = \\
\quad (\forall \mathtt{j}.\ \mathtt{j} < \mathtt{i}\ \Rightarrow\ \neg(\mathtt{M},\ \mathtt{L}_{\mathtt{M}}(\pi_{\mathtt{j}}) \models \mathtt{c}))\ \wedge\ (\mathtt{M},\ \mathtt{L}_{\mathtt{M}}(\pi_{\mathtt{i}}) \models \mathtt{c})
\end{array}
$$

$$
\begin{array}{l}
\vdash\ \mathtt{i} \leq \mathtt{j} \\
\quad \Rightarrow \\
\quad (\mathtt{NextRise\ M}\ \pi\ \mathtt{c\ (i,j)}\ = \\
\quad\quad (\forall \mathtt{k}.\ \mathtt{i} \leq \mathtt{k} \wedge \mathtt{k} < \mathtt{j}\ \Rightarrow\ \neg(\mathtt{M},\ \mathtt{L}_{\mathtt{M}}(\pi_{\mathtt{k}}) \models \mathtt{c}))\ \wedge\ (\mathtt{M},\ \mathtt{L}_{\mathtt{M}}(\pi_{\mathtt{j}}) \models \mathtt{c}))
\end{array}
$$

[12] Personal communication from Cindy Eisner.

The proof of these were essentially routine, though quite a bit more tricky than expected. Immediate corollaries are

$\vdash$ `FirstRise` $\mathtt{M}\ \pi\ \mathrm{T}\ \mathtt{i}\ =\ (\mathtt{i}=0)$
$\vdash$ $\mathtt{i} \leq \mathtt{j}\ \Rightarrow$ (`NextRise` $\mathtt{M}\ \pi\ \mathrm{T}$ `(i,j)` $=$ $(\mathtt{i}=\mathtt{j})$)

5.2 Relating the clocked and unclocked semantics

If we define `ClockFree r` to mean that `r` contains no clocking constructs (a simple recursion over the syntax of SEREs), then clocking with T is equivalent to the unclocked SERE semantics.

$$\vdash \forall\mathtt{r}.\ \mathtt{ClockFree\ r} \Rightarrow ((\mathtt{M},\ \mathtt{w} \overset{\mathrm{T}}{\models} \mathtt{r}) = (\mathtt{M},\ \mathtt{w} \models \mathtt{r}))$$

The proof of this is an easy structural induction, and shows that when the clock is T, the clocked semantics of SEREs collapses to the semantics in Section 3.2.

We tried to prove a similar result for FL formulas, but at first this turned out to be impossible. The reason was that the proof required first showing

$$\forall\mathtt{f}\ \pi.\ (\mathtt{M},\ \pi \overset{\mathrm{T}}{\models} \mathtt{f}) = (\mathtt{M},\ \pi \overset{\mathrm{T!}}{\models} \mathtt{f}))$$

However, the original semantics had the following:

$$(\mathtt{M},\ \pi \overset{\mathtt{c!}}{\models} \mathtt{b}) = \exists\mathtt{i}.\ \mathtt{FirstRise\ M}\ \pi\ \mathtt{c\ i} \wedge (\mathtt{M},\ \mathtt{L_M}(\pi_\mathtt{i}) \models \mathtt{b})$$

$$(\mathtt{M},\ \pi \overset{\mathtt{c}}{\models} \mathtt{b}) = \exists\mathtt{i}.\ \mathtt{FirstRise\ M}\ \pi\ \mathtt{c\ i} \Rightarrow (\mathtt{M},\ \mathtt{L_M}(\pi_\mathtt{i}) \models \mathtt{b})$$

Instantiating `c` to T and using the corollary about `FirstRise` yields

$$(\mathtt{M},\ \pi \overset{\mathrm{T!}}{\models} \mathtt{b}) = \exists\mathtt{i}.\ (\mathtt{i=0}) \wedge (\mathtt{M},\ \mathtt{L_M}(\pi_\mathtt{i}) \models \mathtt{b})$$

$$(\mathtt{M},\ \pi \overset{\mathrm{T}}{\models} \mathtt{b}) = \exists\mathtt{i}.\ (\mathtt{i=0}) \Rightarrow (\mathtt{M},\ \mathtt{L_M}(\pi_\mathtt{i}) \models \mathtt{b})$$

With this, clearly $(\mathtt{M},\ \pi \overset{\mathrm{T}}{\models} \mathtt{b})$ is not equal to $(\mathtt{M},\ \pi \overset{\mathrm{T!}}{\models} \mathtt{b})$. The solution, suggested by Cindy Eisner, is to replace the weak semantics by

$$(\mathtt{M},\ \pi \overset{\mathtt{c}}{\models} \mathtt{b}) = \forall\mathtt{i}.\ \mathtt{FirstRise\ M}\ \pi\ \mathtt{c\ i} \Rightarrow (\mathtt{M},\ \mathtt{L_M}(\pi_\mathtt{i}) \models \mathtt{b})$$

so that we get

$$(\mathtt{M},\ \pi \overset{\mathrm{T!}}{\models} \mathtt{b}) = \exists\mathtt{i}.\ (\mathtt{i=0}) \wedge (\mathtt{M},\ \mathtt{L_M}(\pi_\mathtt{i}) \models \mathtt{b})$$

$$(\mathtt{M},\ \pi \overset{\mathrm{T}}{\models} \mathtt{b}) = \forall\mathtt{i}.\ (\mathtt{i=0}) \Rightarrow (\mathtt{M},\ \mathtt{L_M}(\pi_\mathtt{i}) \models \mathtt{b})$$

which makes $(\mathtt{M},\ \pi \overset{\mathrm{T}}{\models} \mathtt{b})$ equal to $(\mathtt{M},\ \pi \overset{\mathrm{T!}}{\models} \mathtt{b})$. The same change of $\exists$ to $\forall$ is also needed for the semantics of weak clocking for $\mathtt{f1} \wedge \mathtt{f2}$, `X! f`, `{r}(f)`, `{r1}|->{r2}` and `f abort b`. With these changes, we used structural induction to prove:[13]

$$\vdash \forall\mathtt{f}\ \pi.\ (\mathtt{M},\ \pi \overset{\mathrm{T}}{\models} \mathtt{f}) = (\mathtt{M},\ \pi \overset{\mathrm{T!}}{\models} \mathtt{f})$$

However, we were still unable to prove

$$\vdash \forall\mathtt{f}.\ \mathtt{ClockFree\ f} \Rightarrow ((\mathtt{M},\ \pi \overset{\mathrm{T}}{\models} \mathtt{f}) = (\mathtt{M},\ \pi \models \mathtt{f}))$$

where here `ClockFree f` means that `f` contains no clocked FL formulas or SEREs. The proof attempt failed because the unclocked semantics for `[f1 U f2]` had a path length check, but the strongly clocked semantics didn't. After restricting the quantification of a variable in the strongly clocked semantics to values satisfying `pl` π, the proof went through.

[13] See Section 5.4 for further developments!

5.3 Validating the clock implementation rewriting rules

As discussed in Section 4.1, the semantics of clocked SEREs and formulas can be given in two ways:

1. by defining $\overset{\texttt{c}}{\models}$ and, for formulas, $\overset{\texttt{c!}}{\models}$;
2. by translating away clocking constructs `r@c`, `f@c` and `f@c!` using rewrites, then using the unclocked semantics $\models$.

The representation in HOL of the direct semantics (1) has already been discussed.
The definition of the translation (2) in HOL is straightforward: one just defines recursive functions `SClockImp`, that takes a clock and a SERE and returns a SERE, and `FClockImp` that takes a clock context and a formula and returns a formula. Thus roughly[14]

$$\begin{array}{l}\texttt{SClockImp : clock} \rightarrow \texttt{sere} \rightarrow \texttt{sere}\\ \texttt{FClockImp : clock} \rightarrow \texttt{fl} \rightarrow \texttt{fl}\end{array}$$

We can then attempt to prove that

$$\vdash \ \forall \texttt{r w c.}\ (\texttt{M},\ \texttt{w}\ \overset{\texttt{c}}{\models}\ \texttt{r}) \ = \ (\texttt{M},\ \texttt{w}\ \models\ \texttt{SClockComp c r})$$

which turns out to be a routine proof by structural induction on `r`. However, the results for formulas

$$\vdash \ \forall \texttt{f}\ \pi\ \texttt{c.}\ (\texttt{M},\ \pi\ \overset{\texttt{c}}{\models}\ \texttt{f}) \ = \ (\texttt{M},\ \pi\ \models\ \texttt{FClockComp c f})$$
$$\vdash \ \forall \texttt{f}\ \pi\ \texttt{c.}\ (\texttt{M},\ \pi\ \overset{\texttt{c!}}{\models}\ \texttt{f}) \ = \ (\texttt{M},\ \pi\ \models\ \texttt{FClockComp c! f})$$

are harder, and we have not yet finished proving these (as of 5 July 2002). To see the complexity involved consider the rewrite for weakly clocked conjunctions [9, page 67]:

$$\texttt{(f1} \wedge \texttt{f2)@c} \quad \longrightarrow \quad \texttt{[}\neg\texttt{c W (c} \wedge \texttt{(f1@c} \wedge \texttt{f2@c))]}$$

where `W` is the 'weak until' operator which is part of the definitional extension (i.e. syntactic sugar) defined as part of *Sugar 2.0*, namely:

$$\texttt{[f1 W f2]} \ = \ \texttt{[f1 U f2]} \vee \texttt{G f1}$$

where `U` is a primitive (part of the kernel) but $\vee$ and `G` are defined by:

$$\begin{array}{lcl}\texttt{f1} \vee \texttt{f2} & = & \neg(\neg\texttt{f1} \wedge \neg\texttt{f2})\\ \texttt{G f} & = & \neg\texttt{F}(\neg\texttt{f})\end{array}$$

and `F` is defined by

$$\texttt{F f} \ = \ \texttt{[T U f]}$$

Let us define

$$\begin{array}{lcl}\texttt{FClockCorrect M f} & = & (\forall\pi\ \texttt{c.}\ (\texttt{M},\ \pi\ \overset{\texttt{c}}{\models}\ \texttt{f}) \ = \ (\texttt{M},\ \pi\ \models\ \texttt{FClockComp c f}))\\ & & \wedge\\ & & (\forall\pi\ \texttt{c.}\ (\texttt{M},\ \pi\ \overset{\texttt{c!}}{\models}\ \texttt{f}) \ = \ (\texttt{M},\ \pi\ \models\ \texttt{FClockComp c! f}))\end{array}$$

It is relatively straightforward to prove the cases for boolean formulas `b` and negations $\neg$`f`, namely:

$$\vdash \ \forall\texttt{M. FClockCorrect M b}$$
$$\vdash \ \forall\texttt{M f. FClockCorrect M f} \ \Rightarrow \ \texttt{FClockCorrect M}\ (\neg\texttt{f})$$

For formula conjunction we want to prove:

$$\forall\texttt{M f1 f2. FClockCorrect M f1} \ \wedge \ \texttt{FClockCorrect M f2} \ \Rightarrow \ \texttt{FClockCorrect M}\ (\texttt{f1} \wedge \texttt{f2})$$

where the first $\wedge$ is in higher order logic and the one in `f1` $\wedge$ `f2` is part of the Sugar formula syntax.

[14] We are glossing over details here, like what the type `clock` exactly is.

We got bogged down in details when we tried to prove this directly, so we first established some lemmas about $\vee$ and the unclocked semantics of the defined operators `W`, `G` and `F`.

$$\vdash\ (\mathtt{M},\ \pi\ \models\ \mathtt{f1} \vee \mathtt{f2})\ =\ (\mathtt{M},\ \pi\ \models\ \mathtt{f1})\ \vee\ (\mathtt{M},\ \pi\ \models\ \mathtt{f2})$$

$$\vdash\ (\mathtt{M},\ \pi\ \models\ \mathtt{F\ f})\ =\ \exists \mathtt{i} \in \mathtt{pl}\,\pi.\ (\mathtt{M},\ \pi^{\mathtt{i}}\ \models\ \mathtt{f})$$

$$\vdash\ (\mathtt{M},\ \pi\ \models\ \mathtt{G\ f})\ =\ \forall \mathtt{i} \in \mathtt{pl}\,\pi.\ (\mathtt{M},\ \pi^{\mathtt{i}}\ \models\ \mathtt{f})$$

$$\vdash\ \neg(\mathtt{M},\ \pi\ \models\ \mathtt{G\ f})\ =\ \exists \mathtt{i} \in \mathtt{pl}\,\pi.\ (\mathtt{M},\ \pi^{\mathtt{i}}\ \models\ \neg\mathtt{f})$$

$$\vdash\ \neg(\mathtt{M},\ \pi\ \models\ \mathtt{G\ f})\ =\ \exists \mathtt{i} \in \mathtt{pl}\,\pi.\ (\mathtt{M},\ \pi^{\mathtt{i}}\ \models\ \neg\mathtt{f})\ \wedge\ \forall \mathtt{j} \in \mathtt{pl}\,\pi.\ \mathtt{j} < \mathtt{i}\ \Rightarrow\ (\mathtt{M},\ \pi^{\mathtt{j}}\ \models\ \mathtt{f})$$

$$\vdash\ (\mathtt{M},\ \pi\ \models\ [\mathtt{f1\ W\ f2}])\ =\ (\mathtt{M},\ \pi\ \models\ [\mathtt{f1\ U\ f2}])\ \vee\ (\mathtt{M},\ \pi\ \models\ \mathtt{G\ f1})$$

Using these lemmas it is not too hard to prove the desired result about conjunctions. Besides helping with the proof of this, the lemmas also provide some sanity checking of the definitions.

5.4 Restricting quantifiers

The original semantics specifies that some of the quantifications over integer variables be restricted to range over values the are smaller than the length of the current path π (we represent this using $\mathtt{pl}\,\pi$). Our initial attempts to relate the clocked and unclocked semantics needed additional quantifier restrictions to be added, as discussed at the end of Section 5.2 above. However, during email discussions with the *Sugar 2.0* designers it became clear that in fact all quantifications should be restricted, for otherwise the semantics would rely on the HOL logic's default interpretations of terms like π^j when π is finite and $j \geq \mathtt{length}\ \pi$.[15] With HOL's default interpretation of 'meaningless' terms, it is unclear whether the semantics accurately reflects the designers intentions.

Thus the semantics was modified so that all quantifications are suitably restricted. In addition, and in the same spirit, we added the requirement that all terms $\pi^{(i,j)}$ occurred in a context where $i \leq j$, so that the arbitrary value of $\pi^{(i,j)}$ when $i > j$ was never invoked. Unfortunately these changes broke the proof of:

$$\vdash\ \forall \mathtt{f}\ \pi.\ (\mathtt{M},\ \pi\ \overset{\mathtt{T}}{\models}\ \mathtt{f})\ =\ (\mathtt{M},\ \pi\ \overset{\mathtt{T!}}{\models}\ \mathtt{f})$$

and hence the proof relating the clocked and unclocked semantics. However, it turned out that there was a bug in the semantics: "$\mathtt{l} > \mathtt{k}$" occurred in a couple of places where there should have been "$\mathtt{l} \geq \mathtt{k}$", and when this change was made the proof of the above property, and the equivalence between the unclocked and true-clocked semantics, went through.

However, just as we thought everything was sorted out, the *Sugar 2.0* designers announced they had discovered a bug and pointed out that without their fix we should not have been able to prove what we had. This bug had arisen in the semantics of `X!` formulas when the $\exists$-to-$\forall$ change to the weakly clocked semantics (which we discussed in Section 5.2) was made.

Careful manual analysis showed that an error in the HOL semantics had been introduced when the $\exists$-to-$\forall$ change was made, and this error masked the bug that should have appeared when we tried to do the proof. Thus a bug in the HOL semantics allowed a proof to succeed when it shouldn't have! After removing the transcription error from the HOL semantics the proofs failed, as they should, and after the correct fix, supplied by the Sugar designers, was made to the semantics the proofs went through.

This experience with a transcription error masking a bug has sensitised us to the dangers of manually translating the typeset semantics into HOL. We had carefully and systematically manually checked that the HOL was a correct more than once, but nevertheless the error escaped detection. As a result, we are experimenting with ways of structuring LaTeX source to represent the 'deep structure' of the semantics rather than its 'surface form'. The idea is to define LaTeXcommands (macros) that are semantically meaningful and can be parsed directly into logic with a simple script. The LaTeX definitions of the commands will then

[15] The logical treatment of 'undefined' terms like 1/0 or `hd[]` has been much discussed. HOL uses a simple and consistent approach based on Hilbert's ε-operator. Other approaches include 'free logics' (i.e. logics with non-denoting terms) and three-valued logics in which formulas can evaluate to *true*, *false* and *undefined*.

generate the publication form of the semantics. By giving the commands extra parameters that can be used to hold strings for generating English, but ignored when translating to HOL, it appears possible to use LaTeX to represent the semantics. However, the resulting document source is rather complex and may be hard to maintain. The long term 'industry standard' solution to this problem is to use XML (e.g. MathML), but current infrastructure for MathML is either not quite ready (e.g. Publicon[16]) or not quite polished enough for everyday use (e.g. IBM texexplorer[17], Mozilla[18] and TtM[19])

6 Conclusions

It was quite straightforward to use the informal semantics in the *Sugar 2.0* documentation to create a deep embedding of the whole *Sugar 2.0* kernel. Attempting to prove some simple 'sanity checking' lemmas with a proof assistant quickly revealed bugs in the translated semantics (and possibly in the original). Further probing revealed more bugs.

It is hoped that the semantics in HOL that we now have is correct, but until further properties are proved we cannot be sure, and the experience so far suggests caution!

7 Acknowledgements

The *Sugar 2.0* team of Cindy Eisner and Dana Fisman patiently answered numerous email questions in great detail. They also supplied valuable comments and corrections to an earlier version of this paper, and suggested ways of modifying the HOL semantics to get the proofs described in Section 5 to go through.

References

1. I. Beer, S. Ben-David, C. Eisner, D. Fisman, A. Gringauze, and Y. Rodeh. The temporal logic Sugar. In G. Berry, H. Comon, and A. Finkel, editors, *Proc.* 13^{th} *International Conference on Computer Aided Verification (CAV)*, LNCS 2102. Springer-Verlag, 2001.
2. R. Boulton, A. Gordon, M. Gordon, J. Harrison, J. Herbert, and J. Van Tassel. Experience with embedding hardware description languages in HOL. In V. Stavridou, T. F. Melham, and R. T. Boute, editors, *Theorem Provers in Circuit Design: Proceedings of the IFIP TC10/WG 10.2 International Conference, Nijmegen, June 1992*, IFIP Transactions A-10, pages 129–156. North-Holland, 1992.
3. M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: a theorem-proving environment for higher-order logic*. Cambridge University Press, 1993.
4. J. Halpern, Z. Manna, and B. Moszkowski. A hardware semantics based on temporal intervals. In J. Diaz, editor, *Proceedings of the 10-th International Colloquium on Automata, Languages and Programming*, volume 154 of *LNCS*, pages 278–291. Springer Verlag, 1983.
5. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
6. S. Rajan, N. Shankar, and M.K. Srivas. An integration of model-checking with automated proof checking. In Pierre Wolper, editor, *Computer-Aided Verification, CAV '95*, volume 939 of *Lecture Notes in Computer Science*, pages 84–97, Liege, Belgium, June 1995. Springer-Verlag.
7. K. Slind. Function definition in higher order logic. In J. von Wright, J. Grundy, and J. Harrison, editors, *Theorem Proving in Higher Order Logics: 9th International Conference, Turku, Finland, August 1996: Proceedings*, volume 1125 of *Lecture Notes in Computer Science*, pages 381–397. Springer-Verlag, 1996.
8. `www.haifa.il.ibm.com/projects/verification/sugar/literature.html`.
9. `www.haifa.il.ibm.com/projects/verification/sugar/Sugar_2.0_Accellera.ps`.

[16] http://www.wolfram.com/products/publicon/
[17] http://www-3.ibm.com/software/network/techexplorer/
[18] http://www.mozilla.org/projects/mathml/
[19] http://hutchinson.belmont.ma.us/tth/mml/