Under consideration for publication in Formal Aspects of Computing

Validating the PSL/Sugar semantics using automated reasoning

Michael J. C. Gordon

Abstract.

The Accellera organisation selected Sugar, IBM's formal specification language, as the basis for a standard to "drive assertion-based verification" in the electronics industry. Sugar combines regular expressions, Linear Temporal Logic (LTL) and Computation Tree Logic (CTL) into a property language intended for both static verification (e.g. model checking) and dynamic verification (e.g. simulation). In 2003 Accellera decided to rename the evolving standard to "Accellera Property Specification Language" (or "PSL" for short).

We motivate and describe a deep semantic embedding of PSL in the version of higher order logic supported by the HOL 4 theorem proving system. The main goal of this paper is to demonstrate that mechanised theorem proving can be a useful aid to the validation of the semantics of an industrial design language.

Keywords: Property language, Sugar, Accellera, PSL, Semantics, Formal verification, Model checking, Theorem proving, higher order logic, HOL

1. Background on the Accellera organisation and the Sugar property language

The Accellera organisation's website contains the following 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 a growing number of syntactically and semantically incompatible formal property languages, Accellera set up a committee:

The Accellera Formal Property Language Technical Committee is chartered with the responsibility of defining a property specification language standard compatible with both the Verilog (IEEE-1364) and VHDL (IEEE-1076) language. This formal language is targeted for both dynamic verification (e.g., simulation) as well as static verification (e.g., model checking). In addition, the Formal Property Language Technical Committee is chartered with:

- Driving standardisation among developers, users and academia,
- Promoting use of the standard, and
- Assuring interoperability for the property specification language among various verification tools within the hardware design flow.

Correspondence and offprint requests to: Mike Gordon, University of Cambridge Computer Laboratory, William Gates Building, JJ Thomson Avenue, Cambridge CB3 0FD, U.K. e-mail: mjcg@cl.cam.ac.uk

This committee conducted a competition to select a property language design to be the basis of the Accellera standard. The finalists of the competition were based on four existing languages:

- 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, the field was narrowed down to Sugar and CBV, and then in April 2002 a vote selected IBM's submission, Sugar. The Accellera Formal Property Language Technical Committee then used Sugar as the starting point for defining a standard language they named PSL (short for Accellera Property Specification Language). This paper describes work that started with Sugar and then evolved to apply to PSL, and so we will sometimes refer to the language as PSL/Sugar.

The version of Sugar submitted by IBM to Accellera (Sugar 2.0) is primarily an LTL-based language that is a successor to the CTL-based Sugar 1 [BBDE$^+$01]. A key idea of both languages is the use of extended regular expression constructs called *Sugar Extended Regular Expressions* or SEREs. Sugar 2.0 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 clocks signals are sampled (different sub-formulas may be evaluated with respect to different clocks). The finite path semantics allows properties to be interpreted on simulation runs by test-bench tools. The addition of clocking and finite path semantics makes the Sugar 2.0 semantics much more complicated than the Sugar 1 semantics. However, for a real 'industry standard' language Sugar is relatively simple.

The aim of this paper is both to document the representation of the semantics of PSL/Sugar in higher order logic and also to describe how theorem proving has been used to validate the semantics. In Section 2, various motivations for this work are discussed. In Section 3, higher order logic and semantic embedding are reviewed. In Section 4, the full semantics PSL in higher order logic is described. In Section 5, progress so far in analysing the semantics using the HOL 4 theorem proving system is discussed. Finally, in Section 6 there is a short section of conclusions and future plans.

The material shown in framed boxes is copied from the LaTeX sources of the the Accellera Property Specification Language Reference Manual [Acc] (henceforth called "LRM" for short). Note that, due to a different style files used for this paper and the LRM, the text in the boxes will be typeset differently from how it is typeset in the LRM.

2. Why embed PSL/Sugar in HOL?

There are various reasons for embedding PSL/Sugar in a machine-processable formal logic. The examples in the following sections are not meant to be exhaustive.

2.1. Debugging 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. 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 PSL/Sugar, such meta-theorems include showing that expected simplifications to the semantics occur if there is no non-trivial clocking, that different representations of the 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. Section 5 describes what we have proved.

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. Such definitions can be validated by proving that they achieve the intended semantics.

2.2. Machine processable semantics

The current PSL/Sugar document is informal mathematics presented as typeset text. Tool developers could benefit from a machine readable 'golden semantics'. One might think of using some XML-based representation of mathematical content. Although there are XML-based presentation oriented representations like *MathML* [Mat] and content oriented representations like *OpenMath* [Ope], there is currently not much support for representing formal semantic documents of the kind needed for PSL/Sugar. See Section 5.1 for some further discussion.

Higher order logic is a widely used formalisation medium (versions of it are used by HOL, Isabelle/HOL, PVS, NuPrl and Coq) and the semantic embedding of model-checkable logics in it is standard [RSS95, SH99, NPW02]. Once one has a representation in higher order logic, then representations in other formats should be straightforward to derive. If one were to select an off-the-shelf logic as a standard for encoding semantics in a machine readable form, then the intersection of the versions of higher order logic supported by the tools mentioned above would be a reasonable choice. The logic used here would be close to this.

2.3. Combining checking with theorem proving

Theorem proving can be used to reason about data-processing over infinite data-types like numbers (e.g. including reals and complex numbers for DSP applications). The combination of PSL/Sugar and higher order logic is quite expressive and provides temporal logic constructs as higher level syntactic sugar for higher order logic, thereby enabling properties to be formulated elegantly.

PSL/Sugar 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 implementing experimental tools that transform properties stated in PSL/Sugar to checking automata. This is inspired by IBM's FoCs project [ABG+00], but uses compilation by theorem proving to ensure semantic equivalence between the executable checker and the source property [GHS03].

2.4. 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 PSL/Sugar is pedagogically valuable. Teaching an industrial property language nicely complements and motivates academic languages like LTL and CTL.

The semantic embedding and validation of Sugar and then PSL using 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 PSL/Sugar has educational potential for training HOL users. In fact, the PSL/Sugar semantics is an example distributed with HOL.

3. Review of higher order logic and semantic embedding

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 in what follows is sufficiently close to standard informal mathematics that it needs no systematic explanation. In this section we briefly outline some features of the version of higher order logic implemented in the HOL 4 system. We refer to this logic as "the HOL logic" or just "HOL".

The HOL logic is built out of *terms* which are of four types: constants, variables, combinations (or function applications) $t_1\ t_2$ and λ-abstractions $\lambda x.\ t$.

The particular set of constants that are available depends on the theory one is working in. The kernel of the HOL logic contains constants T and F representing truth and falsity, respectively. In the HOL system new constants can be defined in terms of existing constants using definitional mechanisms that guarantee no new inconsistencies are introduced. Defined constants include numerals (e.g. $0, 1, 2$), strings (e.g. `"a"`,

`"b"`, `"ab"`) and logical operators (e.g. $\wedge$, $\vee$, $\neg$, $\forall$, $\exists$). The details of HOL's theory of definition are available elsewhere [GM93].

The simple kernel of four kinds of terms can be extended using syntactic sugar to include all the normal notations of predicate calculus. The extension process consists of defining new constants and then adding syntactic sugar to make terms containing these constants look familiar. For example, constants $\forall$, $\exists$ and `Pair` can be defined and then $\forall x.\ \exists y.\ P(x,y)$ is syntactic sugar for $\forall(\lambda x.\ \exists(\lambda y.\ P\ (\texttt{Pair}\ x\ y)))$, (here the function application `Pair` x y means ((`Pair` x) y), so `Pair` is 'curried'). If P is a function that returns a truth-value (i.e. a predicate), then P can be thought of as a set, and we write $x \in P$ to mean $P(x)$ is true. The term $\lambda x.\ \cdots x \cdots$ corresponds to the set abstraction $\{x \mid \cdots x \cdots\}$ and we will write $\forall x \in P.\ Q(x)$ and $\exists x \in P.\ Q(x)$ to mean $\forall x.\ P(x) \Rightarrow Q(x)$ and $\exists x.\ P(x) \wedge Q(x)$, respectively.

Higher order logic is typed to avoid inconsistencies.[1] Types are syntactic constructs that denote sets of values. For example, types *bool* and *num* are atomic types in HOL and denote the sets of booleans and natural numbers, respectively. Complex types can be built using type constructors. For example, if ty_1 and ty_2 are types, then $ty_1 \rightarrow ty_2$ denotes the set of functions with domain ty_1 and range ty_2, and $ty_1 \times ty_2$ denotes the Cartesian product of the sets denoted by ty_1 and ty_2. Type constructors are traditionally applied to their arguments using a postfix notation like $(ty_1, \ldots, ty_n)$*constructor*. The types $ty_1 \rightarrow ty_2$ and $ty_1 \times ty_2$ are just special notations for (ty_1, ty_2)*fun* and (ty_1, ty_2)*prod*, respectively.

If the types for all the variables and constants in a term t are given, then a type-checking algorithm can determine whether t is well-typed – i.e. every function is applied to an argument of the correct type – and compute a type for t. For example, $\neg 3$ is not well-typed (assuming $\neg$ has type $bool \rightarrow bool$ and 3 has type *num*) and would be rejected by type-checking, however, $\neg$`T` is well-typed (assuming `T` has type *bool*) and would be accepted and given type *bool*. Only the well-typed terms are considered meaningful and we write $t : ty$ if term t is well-typed and has type ty. Well-typed terms of type *bool* are the formulas of the HOL logic, thus formulas are a subset of terms: $\forall x.\ \exists y.\ x + 1 < y$ is a term that is a formula, but $x + 1$ is a term (of type *num*) that is not a formula. The HOL logic kernel only has two types and one type constructor: type *bool* of booleans, an infinite type *ind* of 'individuals' and the function type constructor $\rightarrow$. Other types and type constructors can be defined in terms of these [GM93]. For example, the type *num* of numbers is defined as a subset of the primitive type *ind*, and the Cartesian product constructor $\times$ can be defined in terms of $\rightarrow$. Families of terms can be created by using type variables. For example, if variable x is assigned the type α, where α is a type variable, then $\lambda x.\ x$ has type $\alpha \rightarrow \alpha$ and is a family of identity functions with an instance $\lambda x : ty.\ x$ for each type ty.

B.2 of the LRM introduces the semantic notions used to specify the PSL/Sugar semantics as follows:

> The semantics of a Sugar formula are defined with respect to a *model* M. A model is a quintuple (S, S_0, R, P, L), where S is a finite set of states, $S_0 \subseteq S$ is a set of initial states, $R \subseteq S \times S$ is the transition relation, P is a non-empty set of atomic propositions, and L is the valuation, a function $L : S \longrightarrow 2^P$, mapping each state with a set of atomic propositions valid in that state.
> A *path* w is a finite (or infinite) sequence of states $w = (w_0, w_1, w_2, \cdots, w_n)$ (or $w = (w_0, w_1, w_2, \cdots)$). A *computation path* w of a model M is a finite (or infinite) path w such that for every $i < n$, $R(w_i, w_{i+1})$ and for no s, $R(w_n, s)$ (or such that for every i, $R(w_i, w_{i+1})$). Given a finite (or infinite) path w, we define $\hat{L}$, an extension of the valuation function L from states to paths as follows: $\hat{L}(w) = L(w_0)L(w_1)\ldots L(w_n)$ (or $\hat{L}(w) = L(w_0)L(w_1)\ldots$). Thus we have a mapping from states in M to letters of 2^P, and from finite (or infinite) sequences of states in M to finite (or infinite) words over 2^P.
> We will denote a letter from 2^P by ℓ, and a finite or infinite word from 2^P by ω. We denote the length of word ω as $|\omega|$. A finite word $\omega = \ell_0\ell_1\ell_2\cdots\ell_n$ has length $n+1$, while an infinite word has length ∞. We denote by ω^i the suffix of ω starting at ℓ_i. That is, $\omega^i = \ell_i\ell_{i+1}\cdots\ell_n$ (or $\omega^i = \ell_i\ell_{i+1}\cdots$). We denote by $\omega^{i,j}$ the finite sequence of letters starting from ℓ_i and ending in ℓ_j. That is, $\omega^{i,j} = \ell_i\ell_{i+1}\ldots\ell_j$.

In HOL, the parameterisation of the semantics on the sets of atomic propositions and states is represented using type variables *aprop* and *state*, which can be instantiated to types modelling atomic propositions and states of particular applications.

A model is represented in HOL by a term (S, S_0, R, P, L). The set of states S is modelled by a predicate on the type *state*, so $S : state \rightarrow bool$. Often the type variable *state* will be instantiated to a particular type (e.g. a tuple of booleans) such that every value is a valid state, but allowing the possibility that S correspond to a

[1] Russell's paradox can be formulated as: $(\lambda x.\ \neg(x\ x))\ (\lambda x.\ \neg(x\ x)) = \neg((\lambda x.\ \neg(x\ x))\ (\lambda x.\ \neg(x\ x)))$.

subset gives additional flexibility when defining particular models (though we do not exploit this flexibility here). The set of initial states S_0 is also represented by a predicate, so $S_0 : state \rightarrow bool$. The requirement that S_0 be a non-empty subset of S is represented by the formula $(\exists s.\ S_0\ s)\ \wedge\ (\forall s.\ S_0\ s \Rightarrow S\ s)$, which may need to be an assumption when formulating properties (though not in any examples in this paper). The transition relation R is represented by a predicate on pairs. Thus $R(s, s')$ is true if and only if state s' is a possible to successor to state s, so $R : (state \times state) \rightarrow bool$. The set of atomic propositions P is modelled by a predicate on a type *aprop*, so $P : aprop \rightarrow bool$. The valuation L maps a state s to the set of propositions true in s, thus $L : state \rightarrow (aprop \rightarrow bool)$. If M is a model (S, S_0, R, P, L) then:

$$M\ :\ (state \rightarrow bool) \times (state \rightarrow bool) \times ((state \times state) \rightarrow bool) \times (aprop \rightarrow bool) \times (state \rightarrow (aprop \rightarrow bool))$$

Let *model* be a binary type constructor defined so that $(state, aprop)model$ abbreviates the type of M above, then we can write $M : (state, aprop)model$.

To semantically embed PSL/Sugar in HOL we first define types and constants to represent the syntax of Boolean Expressions, Sugar Extended Regular Expressions (SEREs), Foundation Language (FL) formulas and Optional Branching Extension (OBE) formulas. This provides an abstract syntax and gives a representation of PSL/Sugar constructs as terms in the HOL logic. The semantics is then specified by defining functions (i.e constants of functional type) that map each construct to a representation of its meaning expressed in the HOL logic. An embedding in which the syntax and semantics is represented inside a logic is called a deep embedding [BGG+92]. See Section 4.1 for details of how boolean expressions are deeply embedded in HOL.

4. Formal semantics in higher order logic

In this section we give the abstract syntax and semantics of Version 1.01 PSL in higher order logic. In Section 4.1 we briefly show how the syntax and semantics are input to the HOL system, but after that we only give 'pretty printed' semantics using a HOL-to-LaTeX tool implemented by Keith Wansbrough. We do not motivate the particular constructs of Sugar that are found in PSL, nor do we provide a narrative explanation of the semantics in detail, since this is done thoroughly in the LRM.

4.1. Boolean expressions in PSL/Sugar

The syntax of boolean expressions from B.1 of the LRM is as follows:

- Every atomic proposition is a boolean expression.
- If b, b_1, and b_2 are boolean expressions, then so are the following:
  - (b)
  - $\neg b$
  - $b_1 \wedge b_2$

This is represented in HOL by a recursive type definition of a data-type that represents the syntax of boolean expressions. Since atomic propositions are boolean expressions, we parameterise the type of boolean expressions on *aprop*. Thus the type of terms representing boolean expressions is *(aprop)bexp*, where *bexp* is a unary type constructor. The input to the system is:

```
Hol_datatype `bexp = B_PROP   of aprop                    (* atomic proposition      *)
                   | B_NOT    of bexp                     (* negation                *)
                   | B_AND    of bexp * bexp`             (* conjunction             *)
```

this defines a new unary type constructor *bexp* and constants:

```
B_PROP : aprop → (aprop)bexp
B_NOT  : (aprop)bexp → (aprop)bexp
B_AND  : (aprop)bexp × (aprop)bexp → (aprop)bexp
```

We do not provide a separate constructor for "(b)", since putting brackets around an expression does not change its meaning. If atomic propositions are taken to be strings, then the boolean expression $x \wedge \neg y$ would

be represented by the term `B_AND(B_PROP "x", B_NOT(B_PROP "y"))` which has the type `(string)bexp`. The semantics of boolean expressions from B.2.1 of the LRM is:

> We define the semantics of boolean expressions over letters from the alphabet 2^P, thus a letter is a subset of the set of atomic propositions P. The notation $\ell \models b$ means that boolean expression b holds under the truth assignment represented by ℓ. The semantics of boolean expressions are defined as follows, where p denotes an atomic proposition and b, b_1, and b_2 denote boolean expressions.
>
> - $\ell \models p \Longleftrightarrow p \in \ell$
> - $\ell \models (b) \Longleftrightarrow \ell \models b$
> - $\ell \models \neg b \Longleftrightarrow \ell \not\models b$
> - $\ell \models b_1 \wedge b_2 \Longleftrightarrow \ell \models b_1$ and $\ell \models b_2$

This is represented in HOL by defining a semantic function `B_SEM : (aprop→bool)→(aprop)bexp→bool` such that `B_SEM` l b is true iff b is true in state l. The actual input to HOL is

```
Define ‘(B_SEM l (B_PROP(p:’aprop))  =  p IN l)
        /\
        (B_SEM l (B_NOT b)           =  ~(B_SEM l b))
        /\
        (B_SEM l (B_AND(b1,b2))      =  B_SEM l b1  /\  B_SEM l b2)‘
```

however, if we pretty print this using the HOL-to-LaTeX tool, augmented with $l \models b$, $\neg b$, $b_1 \wedge b_2$ for `B_SEM` l b, `B_NOT` b and `B_AND` b_1 b_2, respectively, the definition becomes:

`Define` ‘$(l \models p \;=\; p \in l) \;\wedge\; (l \models \neg b \;=\; \neg l \models b) \;\wedge\; (l \models b_1 \wedge b_2 \;=\; l \models b_1 \;\wedge\; l \models b_2)$‘

Note that after pretty printing the symbols $\neg$ and $\wedge$ become overloaded: the occurrence of $\neg$ in $\neg b$ is part of the boolean expression syntax of PSL/Sugar, but the occurrence in $\neg(l \models b)$ is negation in higher order logic. Similarly $\wedge$ is overloaded: the occurrence in $b_1 \wedge b_2$ is part of the boolean expression syntax, but the other occurrences are conjunction in higher order logic.

4.2. Extended Regular Expressions (SEREs)

Sugar has constructs called *Sugar Extended Regular Expressions* or SEREs. When Accellera renamed Sugar to PSL they kept the acronym SERE, but decreed that it should henceforth abbreviate *Sequential Extended Regular Expression*. The abstract syntax from B.1 of the LRM is:

> - Every boolean expression is a SERE.
> - If r, r_1, and r_2 are SEREs, and clk is a boolean expression, then the following are SEREs:
>   - $\{r\}$
>   - $r_1 \;;\; r_2$
>   - $r_1 \;:\; r_2$
>   - $\{r_1\} \mid \{r_2\}$
>   - $\{r_1\}$ && $\{r_2\}$
>   - $r[*]$
>   - $r@clk$

The semantics from B.2.3.1 of the LRM is shown in the next box:

Clocked SEREs are defined over finite words from the alphabet 2^P and a boolean expression that serves as the clock context. The notation $w \models^c r$, where r is a SERE and c is a boolean expression, means that w is in the language of r in context of clock c. The semantics of clocked SEREs are defined as follows, where b, c, and c_1 denote boolean expressions, r, r_1, and r_2 denote clocked SEREs, and $[i..k)$ denotes the set of integers $\{j \ : \ i \leq j \ \wedge \ j < k\}$.

- $w \models^c b \iff |w| \geq 1$, for every $i \in [0..|w|-1)$, $\ell_i \models \neg c$ and $\ell_{|w|-1} \models c \wedge b$
- $w \models^c \{r\} \iff w \models^c r$
- $w \models^c r_1; r_2 \iff$ there exists w_1 and w_2 such that $w = w_1 w_2$, $w_1 \models^c r_1$, and $w_2 \models^c r_2$
- $w \models^c r_1 : r_2 \iff$ there exists w_1, w_2, and ℓ such that $w = w_1 \ell w_2$, $w_1 \ell \models^c r_1$, and $\ell w_2 \models^c r_2$
- $w \models^c \{r_1\}|\{r_2\} \iff w \models^c r_1$ or $w \models^c r_2$
- $w \models^c \{r_1\} \,\&\&\, \{r_2\} \iff w \models^c r_1$ and $w \models^c r_2$
- $w \models^c r[*] \iff$ either $w = \epsilon$ or there exists $w_1, w_2, \ldots, w_j$ such that $w = w_1 w_2 \ldots w_j$ and for every $i \in [1..j]$, $\ w_i \models^c r$
- $w \models^c r@c_1 \iff$ there exists $i \in [0..|\omega|)$ such that $\omega^{0,i} \models^{\mathrm{T}} \{\neg c_1[*]; c_1\}$ and $\omega^i \models^{c_1} r$

The HOL representation of this semantics of SEREs is defined by a semantic function `S_SEM` such that `S_SEM` w c r is true iff word w is in the language recognised by the extended regular expression r when the clock context (i.e. current clock) is c. The HOL term `S_SEM` w c r is pretty-printed as $w \models^c r$.

In the (pretty printed) HOL logic, a list containing elements $e_0, \ldots, e_n$ is denoted by $[e_0; \ldots; e_n]$. Words are represented as lists. Juxtaposition of words denotes concatenation (e.g. $w1[s]w2$ is the concatenation of $w1$, $[s]$ and $w2$). If *wlist* is a list of lists then Every p *wlist* applies the predicate p to every element of *wlist* and returns the conjunction of the result (e.g. in the semantics below Every $(\lambda w.\ w \models r)$ *wlist* asserts $w \models r$ for every w in *wlist*) and Concat *wlist* denotes the concatenation of the lists in *wlist* (e.g. Concat $[[a; b]; [c]; [d; e; f]] = [a; b; c; d; e; f]$). The notation $|w|$ denotes the length of w (empty words have length 0) and w_i denotes the ith element of w counting from 0, so w_0 is the first element (note that subscripts on symbols not denoting lists are just subscripts). The HOL-toLaTeX translator generates $(m \ .. \ n)$ for the HOL term denoting the half open interval $\{x \mid m \leq x \ \wedge \ x < n\}$, which in the LRM is written $[m \ .. \ n)$. This notation is potentially quite confusing, as it is usual for round brackets to denote an interval open at both ends, but currently the translator cannot automatically generate the LRM notation from the HOL source.

Applying the translator to the HOL semantics of SEREs yields:

$$
\begin{array}{l}
(w \models^c b \ = \ |w| \geq 1 \ \wedge \ (\forall i \in (0 \ .. \ (|w|-1)).\ w_i \models \neg c) \ \wedge \ w_{|w|-1} \models c \wedge b) \\
\wedge \\
(w \models^c r_1; r_2 \ = \ \exists w_1 \ w_2.\ (w = w_1 w_2) \ \wedge \ w_1 \models^c r_1 \ \wedge \ w_2 \models^c r_2) \\
\wedge \\
(w \models^c r_1 : r_2 \ = \ \exists w_1 \ w_2 \ l.\ (w = w_1[l]w_2) \ \wedge \ w_1[l] \models^c r_1 \ \wedge \ [l]w_2 \models^c r_2) \\
\wedge \\
(w \models^c \{r_1\} \mid \{r_2\} \ = \ w \models^c r_1 \ \vee \ w \models^c r_2) \\
\wedge \\
(w \models^c \{r_1\}\&\&\{r_2\} \ = \ w \models^c r_1 \ \wedge \ w \models^c r_2) \\
\wedge \\
(w \models^c r[*] \ = \ \exists wlist.\ (w = \mathsf{Concat}\ wlist) \ \wedge \ \mathsf{Every}\ (\lambda w.\ w \models^c r)\ wlist) \\
\wedge \\
(w \models^c r@c_1 \ = \ \exists i \in (0 \ .. \ |w|).\ w^{0,i} \models \neg c_1[*]; c_1 \ \wedge \ w^i \models^{c_1} r)
\end{array}
$$

This can be compared with the LRM semantics given in the framed box above.

4.3. Foundation Language (FL)

FL combines standard LTL notation with a less standard abort operation and some constructs using SEREs. The abstract syntax from B.1 of the LRM is:

- Every boolean expression is a Sugar FL formula.
- If b and clk are boolean expressions, f, f_1, and f_2 are Sugar FL formulas and r, r_1, and r_2 are SEREs, then the following are Sugar FL formulas:
  - (f)
  - $\neg f$
  - $f_1 \wedge f_2$
  - $X!\ f$
  - $[f_1\ U\ f_2]$
  - $\{r\}(f)$
  - $\{r_1\}\Rightarrow\{r_2\}!$
  - $\{r_1\}\Rightarrow\{r_2\}$
  - f abort b
  - $f@clk$
  - $f@clk!$

The semantics from B.2.3.2 of the LRM is:

The notation $\omega \models^c f$ where f is a formula and c is a boolean expression means that formula f holds along the (finite or infinite) word ω in the context of clock c. The notation $M \models f$ means that $\hat{L}(w) \models^{\mathrm{T}} f$ for every computation path w in M such that $w_0 \in S_0$ (where $\mathrm{T} = p \vee \neg p$ for some $p \in P$). The semantics of a (clocked) Sugar FL formula are defined as follows, where b, c, and c_1 denote boolean expressions, r, r_1, and r_2 denote SEREs, f, f_1, and f_2 denote (clocked) FL formulas, $[i..k)$ denotes the set of integers $\{j\ :\ i \leq j\ \wedge\ j < k\}$, and $(i..k)$ denotes the set of integers $\{j\ :\ i < j\ \wedge\ j < k\}$.

- $\omega \models^c b \iff \ell_0 \models b$
- $\omega \models^c (f) \iff \omega \models^c f$
- $\omega \models^c \neg f \iff \omega \not\models^c f$
- $\omega \models^c f_1 \wedge f_2 \iff \omega \models^c f_1$ and $\omega \models^c f_2$
- $\omega \models^c X!\ f \iff$ there exists $i \in [1..|w|)$ such that $\omega^{1,i} \models^{\mathrm{T}} \{\neg c[*]; c\}$ and $\omega^i \models^c f$
- $\omega \models^c [f_1\ U\ f_2] \iff$ there exists $k \in [0..|\omega|)$ such that $\omega^k \models^{\mathrm{T}} c$, $\omega^k \models^c f_2$, and for every $j \in [0..k)$ for which $\omega^j \models^{\mathrm{T}} c$, $\omega^j \models^c f_1$
- $\omega \models^c \{r\}(f) \iff$ for every $i \in [0..|\omega|)$ such that $\omega^{0,i} \models^c r$, there exists $j \in [i..|\omega|)$ such that $\omega^{i,j} \models^{\mathrm{T}} \{\neg c[*]; c\}$ and $\omega^j \models^c f$
- $\omega \models^c \{r_1\}\Rightarrow\{r_2\}! \iff$ for every $i \in [0..|\omega|)$ such that $\omega^{0,i} \models^c r_1$ there exists $j \in [i..|\omega|)$ such that $\omega^{i,j} \models^c r_2$
- $\omega \models^c \{r_1\}\Rightarrow\{r_2\} \iff$ for every $i \in [0..|\omega|)$ such that $\omega^{0,i} \models^c r_1$, either there exists $j \in [i..|\omega|)$ such that $\omega^{i,j} \models^c r_2$ or for every $j \in [i..|\omega|)$ there exists a finite word ω' such that $\omega^{i,j}\omega' \models^c r_2$
- $\omega \models^c f$ abort $b \iff$ either $\omega \models^c f$ or $\omega \models^c b$ or there exists $i \in [1..|\omega|)$ and word ω' such that $\omega^i \models^{\mathrm{T}} c \wedge b$ and $\omega^{0,i-1}\omega' \models^c f$
- $\omega \models^c f@c_1! \iff$ there exists $i \in [0..|\omega|)$ such that $\omega^{0,i} \models^{\mathrm{T}} \{\neg c_1[*]; c_1\}$ and $\omega^i \models^{c_1} f$

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 PSL/Sugar literature (e.g. [EF02, Section 4.11]). Although weak clocking $f@c$ is listed in the abstract syntax of formulas, it is actually syntactic sugar defined by: $f@c = \neg(\neg f@c!)$.

The HOL semantics is specified by defining a semantic function `F_SEM` such that `F_SEM` $w\ c\ f$ means FL formula f is true of path w with current clock c. The HOL term `F_SEM` $w\ c\ f$ is pretty printed as $w \models^{c} f$. A path w can be either finite or infinite. Finite paths are represented as lists and infinite paths as function from the natural numbers to states. The notation w_i denotes the i-th state in the path (counting from 0, so w_0 is the first state); w^i denotes the 'i-th tail' of w, which is the path obtained by chopping i elements off the front of w; $w^{(i,j)}$ denotes the finite sequence of states from i to j in w, i.e. $w_i w_{i+1} \cdots w_j$. If w is a path, then the length $|w|$ of w is a number if w is finite, and is ∞ if w is infinite (this is represented in the HOL logic using a sum type). If w is finite, then w_i, w^i and $w^{(i,j)}$ are always valid terms, but have unspecified values if i or j are not less than $|w|$ (inspection of the semantics below shows that this does not happen).[2] The juxtaposition $w^{(i,j)}w'$ denotes the path obtained by concatenating the finite sequence $w^{(i,j)}$ on to the front of the path w'. The semantics $w \models^{c} f$ is defined by recursion on f:

$$
\begin{array}{l}
(w \models^{c} b \ = \ |w| > 0 \ \wedge \ w_0 \models b) \\
\wedge \\
(w \models^{c} \neg f \ = \ \neg(w \models^{c} f)) \\
\wedge \\
(w \models^{c} f_1 \wedge f_2 \ = \ w \models^{c} f_1 \ \wedge \ w \models^{c} f_2) \\
\wedge \\
(w \models^{c} X!\, f \ = \ \exists i \in (1 \mathinner{..} |w|).\ w^{1,i} \models^{\mathrm{T}} \neg c[*]; c \ \wedge \ w^i \models^{c} f) \\
\wedge \\
(w \models^{c} [f_1\ U\ f_2] \ = \ \exists k \in (0 \mathinner{..} |w|).\ w^k \models^{\mathrm{T}} c \ \wedge \ w^k \models^{c} f_2 \ \wedge \ \forall j \in (0 \mathinner{..} k).\ w^j \models^{\mathrm{T}} c \Rightarrow w^j \models^{c} f_1) \\
\wedge \\
(w \models^{c} \{r\}(f) \ = \ \forall i \in (0 \mathinner{..} |w|).\ w^{0,i} \models^{c} r \Rightarrow \exists j \in (i \mathinner{..} |w|).\ w^{i,j} \models^{\mathrm{T}} \neg c[*]; c \ \wedge \ w^j \models^{c} f) \\
\wedge \\
(w \models^{c} \{r_1\} \mapsto \{r_2\}! \ = \ \forall i \in (0 \mathinner{..} |w|).\ w^{0,i} \models^{c} r_1 \Rightarrow \exists j \in (i \mathinner{..} |w|).\ w^{i,j} \models^{c} r_2) \\
\wedge \\
(w \models^{c} \{r_1\} \mapsto \{r_2\} \ = \\
\quad \forall i \in (0 \mathinner{..} |w|).\ w^{0,i} \models^{c} r_1 \Rightarrow ((\exists j \in (i \mathinner{..} |w|).\ w^{i,j} \models^{c} r_2) \ \vee \ (\forall j \in (i \mathinner{..} |w|).\ \exists w'.\ w^{i,j} w' \models^{c} r_2))) \\
\wedge \\
(w \models^{c} f \ abort \ b \ = \ w \models^{c} f \ \vee \ w \models^{c} b \ \vee \ \exists i \in (1 \mathinner{..} |w|).\ \exists w'.\ w^i \models^{\mathrm{T}} c \wedge b \ \wedge \ w^{0,i-1} w' \models^{c} f) \\
\wedge \\
(w \models^{c} f@c_1! \ = \ \exists i \in (0 \mathinner{..} |w|).\ w^{0,i} \models^{\mathrm{T}} \neg c_1[*]; c_1 \ \wedge \ w^i \models^{c_1} f)
\end{array}
$$

This semantics is a careful formalisation of the official semantics shown in the box above, with the exception that two minor errors in the LRM have been fixed: in the definition of $w \models^{c} b$ the occurrence of l_0 has been changed to w_0 and $|w| > 0$ has been added. This change ensures that formulas are defined for empty paths (the official semantics is undefined).

4.4. Optional Branching Extension (OBE)

LTL formulas characterise paths. If the transition relation is non-deterministic (i.e. there are states with more than one possible successor) then there are properties that cannot be expressed in LTL, such as "from every state it is possible to get to a state in which property P holds". Such properties can be expressed in Computation Tree Logic (CTL) and PSL/Sugar contains constructs from CTL called the Optional Branching Extension (OBE). The syntax of the OBE in B.1 of the LRM is completely standard.

[2] The logical treatment of 'undefined' terms like $1/0$ or $hd\,[\,]$ has been much discussed. HOL uses a simple approach based on Hilbert's ε-operator that, although appearing rather ad hoc, is guaranteed to be logically sound. 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*.

- Every boolean expression is an OBE formula.
- If f, f_1, and f_2 are OBE formulas, then so are the following:
  - (f)
  - $\neg f$
  - $f_1 \wedge f_2$
  - EXf
  - $E[f_1\ U\ f_2]$
  - EGf

The semantics from B.2.4 of the LRM is standard:

The semantics of OBE formulas are defined over states in the model, rather than finite or infinite words. The notation $M, s \models f$ means that formula f holds in state s of model M. The notation $M \models f$ is equivalent to $\forall s \in S_0 : M, s \models f$. In other words, f is valid for every initial state of M. The semantics of an OBE formula are defined as follows, where b denotes a boolean expression and f, f_1, and f_2 denote OBE formulas.

- $M, s \models b \Longleftrightarrow s \models b$
- $M, s \models (f) \Longleftrightarrow M, s \models f$
- $M, s \models \neg f \Longleftrightarrow M, s \not\models f$
- $M, s \models f_1 \wedge f_2 \Longleftrightarrow M, s \models f_1$ and $M, s \models f_2$
- $M, s \models EX\ f \Longleftrightarrow$ there exists a computation path w of M such that $|w| > 1$, $w_0 = s$, and $M, w_1 \models f$
- $M, s \models E[f_1\ U\ f_2] \Longleftrightarrow$ there exists a computation path w of M such that $w_0 = s$ and there exists $k < |w|$ such that $M, w_k \models f_2$ and for every j such that $j < k$: $M, w_j \models f_1$
- $M, s \models EG\ f \Longleftrightarrow$ there exists a computation path w of M such that $w_0 = s$ and for every j such that $0 \leq j < |w|$: $M, w_j \models f$

In the HOL representation, if M is a quintuple $M = (S, S0, R, P, L)$ representing a Kripke Structure, then the transition relation component of M is $M.R$, the L component is $M.L$ etc. Define:

$\mathsf{Path}\ M\ p\ s\ =\ (p_0 = s)\ \wedge\ (\forall n.\ M.R(p_n, p_{n+1}))$

The semantic function `O_SEM` is defined using $\mathsf{Path}\ M\ p\ s$ so that `O_SEM` $M\ f\ s$ is true iff f is true of M at state s. HOL terms `O_SEM` $M\ f\ s$ are pretty printed as $M, s \models f$. The semantics of the OBE is defined in HOL by:

$$
\begin{array}{l}
(M, s \models b\ =\ M.L\ s \models b) \\
\wedge \\
(M, s \models \neg f\ =\ \neg(M, s \models f)) \\
\wedge \\
(M, s \models f_1 \wedge f_2\ =\ M, s \models f_1\ \wedge\ M, s \models f_2) \\
\wedge \\
(M, s \models EX\ f]\ =\ \exists p.\ \mathsf{Path}\ M\ p\ s\ \wedge\ M, p_1 \models f) \\
\wedge \\
(M, s \models E[f_1\ U\ f_2]\ =\ \exists p.\ \mathsf{Path}\ M\ p\ s\ \wedge\ \exists k \in (0\ ..\ |p|).\ M, p_k \models f_2\ \wedge\ \forall j.\ j < k \Rightarrow M, p_j \models f_1) \\
\wedge \\
(M, s \models EG\ f]\ =\ \exists p.\ \mathsf{Path}\ M\ p\ s\ \wedge\ \forall j \in (0\ ..\ |p|).\ M, p_j \models f)
\end{array}
$$

For infinite paths this is the standard semantics.

5. Analysing the semantics

Section 4 shows both the LRM semantics (in framed boxes) due to Cindy Eisner and Dana Fisman, and the representation in higher order logic. This is the semantics current when this paper was prepared for publication (June, 2003). In Section 5.1 we discuss some issues arising from the representation of PSL/Sugar semantics in the HOL logic. In Section 5.2 we discuss some properties we have proved, and in Section 5.3 we discuss the proofs of these properties using the HOL system.

5.1. Representing the semantics in higher order logic

The original semantics of Sugar submitted to Accellera in 2002 was different in many details from the current PSL semantics. We initially formalised the original semantics and attempted to establish a number of 'sanity checking' properties suggested by Cindy Eisner and Dana Fisman. Initially we were unsuccessful, but following email discussions with the Sugar team, changes were made to the semantics to correct typographical and minor semantic errors and we were then able to prove all the expected results using the HOL system.

Some of the proofs were quite hard work due to the large number of cases needing manual proof guidance, though they were not mathematically deep. After Accellera selected Sugar, its Formal Property Language Technical Committee modified the semantics to meet political and technical goals. This resulted in a significantly simpler formal semantics and many of the proofs done with the original semantics became much shorter. It was relatively easy to modify the HOL proof scripts to generate the theorems for the new semantics and we expect that as the semantics further evolves, which is expected, the effort to check that the properties still hold will be small. Thus although there were several person months of work in creating and validating the HOL semantics, maintaining it as PSL evolves should be less demanding.

The creation of the HOL versions of the Sugar and PSL semantics was done by hand but, despite repeated and painstaking checking, a number of transcription errors were introduced. Such errors would have been minimised if the formal semantics in HOL was created mechanically from the 'golden' version in the Language Reference Manual. Unfortunately the golden version is a mixture of informal English augmented with mathematical notations in LaTeX. We did explore processing this, but concluded that it was impractical.

We experimented with ways of structuring the LaTeX source to represent the 'deep structure' of the semantics rather than its 'surface form', with the goal of defining LaTeX commands (macros) that are semantically meaningful and can be parsed directly into logic with a simple script. 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 in a form that can be both parsed into HOL and typeset into something close to the LRM text. However, the resulting document source would be rather complex and may be hard to maintain, and consequently unacceptable to Accellera.

The long term 'industry standard' solution to this problem is to use XML, but current infrastructure is not quite ready today (2003). One promising possibility is *OpenMath* [Ope]:

> OpenMath is an emerging standard for representing mathematical objects with their semantics, allowing them to be exchanged between computer programs, stored in databases, or published on the worldwide web.

It remains to see whether the *OpenMath* project will eventually deliver concepts and tools to support the representation of the semantics of industrial design languages.

5.2. Properties proved

The semantics of SEREs and FL formulas are defined with respect to a clock. The 'top level' default clock is T, which is always true. Other clocks are specified using the clocking constructs $r@c$ and $f@c!$. SEREs and formulas not containing "@" are called unclocked and the sets of unclocked SEREs and formulas the unclocked subsets.

The LRM supplies separate semantics for the unclocked subsets. As an initial 'sanity check' we verified that the unclocked semantics were consistent with the semantics in Section 4 when restricted to the unclocked subsets. As a second and deeper check, we then verified the correctness of a translation of arbitrary SEREs and formulas to unclocked ones.

5.2.1. Verifying the unclocked semantics for a trivial clock

The unclocked semantics of SEREs is specified in B.2.2.1 of the LRM by:

The semantics of unclocked SEREs are defined over finite words from the alphabet 2^P. We will denote a finite word over 2^P by w. The concatenation of w_1 and w_2 is denoted by $w_1 w_2$. The empty word is denoted by ϵ, so that $w\epsilon = \epsilon w = w$. The notation $w \models r$, where r is a SERE, means that w is in the language of r. The semantics of SEREs are defined as follows, where b denotes a boolean expression, r, r_1, and r_2 denote unclocked SEREs, and $[i..k]$ denotes the set of integers $\{j \ : \ i \leq j \ \wedge \ j \leq k\}$.

- $w \models b \iff |w| = 1$ and $\ell_0 \models b$
- $w \models \{r\} \iff w \models r$
- $w \models r_1; r_2 \iff$ there exist w_1 and w_2 such that $w = w_1 w_2$, $w_1 \models r_1$, and $w_2 \models r_2$
- $w \models r_1 : r_2 \iff$ there exist w_1, w_2, and ℓ such that $w = w_1 \ell w_2$, $w_1 \ell \models r_1$, and $\ell w_2 \models r_2$
- $w \models \{r_1\}|\{r_2\} \iff w \models r_1$ or $w \models r_2$
- $w \models \{r_1\}\,\&\&\,\{r_2\} \iff w \models r_1$ and $w \models r_2$
- $w \models r[*] \iff$ either $w = \epsilon$ or there exist $w_1, w_2, \ldots, w_j$ such that $w = w_1 w_2 \ldots w_j$ and for every $i \in [1..j]$, $w_i \models r$

The unclocked semantics of FL formulas is specified in B.2.2.2 of the LRM by:

The semantics of Sugar FL formulas are defined over finite or infinite words from the alphabet 2^P. The notation $\omega \models f$ means that formula f holds along the (finite or infinite) word ω. The notation $M \models f$ means that $\hat{L}(w) \models f$ for every computation path w in M such that $w_0 \in S_0$. The semantics of an FL formula are defined as follows, where b denotes a boolean expression, r, r_1, and r_2 denote SEREs, f, f_1, and f_2 denote FL formulas, and $[i..k)$ denotes the set of integers $\{j \ : \ i \leq j \ \wedge \ j < k\}$.

- $\omega \models b \iff \ell_0 \models b$
- $\omega \models (f) \iff \omega \models f$
- $\omega \models \neg f \iff \omega \not\models f$
- $\omega \models f_1 \wedge f_2 \iff \omega \models f_1$ and $\omega \models f_2$
- $\omega \models X!\ f \iff |\omega| > 1$ and $\omega^1 \models f$
- $\omega \models [f_1\ U\ f_2] \iff$ there exists $k \in [0..|\omega|)$ such that $\omega^k \models f_2$, and for every $j \in [0..k)$, $\omega^j \models f_1$
- $\omega \models \{r\}(f) \iff$ for every $j \in [0..|\omega|)$ such that $\omega^{0,j} \models r$, $\omega^j \models f$
- $\omega \models \{r_1\} \Rightarrow \{r_2\}! \iff$ for every $j \in [0..|\omega|)$ such that $\omega^{0,j} \models r_1$ there exists $k \in [j..|\omega|)$ such that $\omega^{j,k} \models r_2$
- $\omega \models \{r_1\} \Rightarrow \{r_2\} \iff$ for every $j \in [0..|\omega|)$ such that $\omega^{0,j} \models r_1$ either there exists $k \in [j..|\omega|)$ such that $\omega^{j,k} \models r_2$ or for every $k \in [j..|\omega|)$ there exists a finite word ω' such that $\omega^{j,k}\omega' \models r_2$
- $\omega \models f$ abort $b \iff$ either $\omega \models f$ or $\omega \models b$ or there exists $j \in [1..|\omega|)$ and word ω' such that $\omega^j \models b$ and $\omega^{0,j-1}\omega' \models f$

In HOL, the semantics of SEREs is represented by defining $w \models r$ by recursion on the structure of r with the following equations:

$$
\begin{array}{l}
(w \models b \ = \ (|w| = 1) \ \wedge \ w_0 \models b) \\
\wedge \\
(w \models r_1; r_2 \ = \ \exists w_1 \ w_2.\ (w = w_1 w_2) \ \wedge \ w_1 \models r_1 \ \wedge \ w_2 \models r_2) \\
\wedge \\
(w \models r_1 : r_2 \ = \ \exists w_1 \ w_2 \ l.\ (w = w_1[l]w_2) \ \wedge \ w_1[l] \models r_1 \ \wedge \ [l]w_2 \models r_2) \\
\wedge \\
(w \models \{r_1\} \mid \{r_2\} \ = \ w \models r_1 \ \vee \ w \models r_2) \\
\wedge \\
(w \models \{r_1\}\&\&\{r_2\} \ = \ w \models r_1 \ \wedge \ w \models r_2) \\
\wedge \\
(w \models r[*] \ = \ \exists wlist.\ (w = \mathsf{Concat}\ wlist) \ \wedge \ \mathsf{Every}\ (\lambda w.\ w \models r)\ wlist)
\end{array}
$$

If we define $\mathsf{ClockFree}(r)$ to mean r is unclocked, then we validated the unclocked semantics by proving, using HOL, that:

$$\vdash\ \forall r.\ \mathsf{ClockFree}(r)\ \Rightarrow\ \forall w.\ w \models^{\mathrm{T}} r\ =\ w \models r$$

The unclocked semantics of FL formulas is specified in HOL by defining $w \models f$ by recursion on the structure of formulas f by (warning: remember that the HOL-to-LaTeX translator confusingly generates $(m\ ..\ n)$ to mean $\{x \mid m \leq x\ \wedge\ x < n\}$):

$$
\begin{array}{l}
(w \models b\ =\ |w| > 0\ \wedge\ w_0 \models b) \\
\wedge \\
(w \models \neg f\ =\ \neg(w \models f)) \\
\wedge \\
(w \models f_1 \wedge f_2\ =\ w \models f_1\ \wedge\ w \models f_2) \\
\wedge \\
(w \models X!\, f\ =\ |w| > 1\ \wedge\ w^1 \models f) \\
\wedge \\
(w \models [f_1\ U\ f_2]\ =\ \exists k \in (0\ ..\ |w|).\ w^k \models f_2\ \wedge\ \forall j \in (0\ ..\ k).\ w^j \models f_1) \\
\wedge \\
(w \models \{r\}(f)\ =\ \forall j \in (0\ ..\ |w|).\ w^{0,j} \models r \Rightarrow w^j \models f) \\
\wedge \\
(w \models \{r_1\} \mapsto \{r_2\}!\ =\ \forall j \in (0\ ..\ |w|).\ w^{0,j} \models r_1 \Rightarrow \exists k \in (j\ ..\ |w|).\ w^{j,k} \models r_2) \\
\wedge \\
(w \models \{r_1\} \mapsto \{r_2\}\ = \\
\forall j \in (0\ ..\ |w|).\ w^{0,j} \models r_1 \Rightarrow ((\exists k \in (j\ ..\ |w|).\ w^{j,k} \models r_2)\ \vee\ (\forall k \in (j\ ..\ |w|).\ \exists w'.\ w^{j,k} w' \models r_2))) \\
\wedge \\
(w \models f\ abort\ b\ =\ w \models f\ \vee\ w \models b\ \vee\ \exists j \in (1\ ..\ |w|).\ \exists w'.\ w^j \models b\ \wedge\ w^{0,j-1} w' \models f)
\end{array}
$$

If we define $\mathsf{ClockFree}(f)$ to mean f is unclocked, then we validated the unclocked semantics by proving, using HOL, that:

$$\vdash\ \forall f.\ \mathsf{ClockFree}(f)\ \Rightarrow\ \forall w.\ w \models^{\mathrm{T}} f\ =\ w \models f$$

5.2.2. Verifying the translation to the unclocked subsets

For an arbitrary SERE r and FL formula f, B.5 of the LRM defines rewrites for computing an unclocked SERE $\mathcal{T}^c(r)$ and an unclocked formula $\mathcal{T}^c(f)$.

The rewrite rules for SEREs are:

- $\mathcal{T}^c(b) = \{\neg c[*]; c \wedge b\}$
- $\mathcal{T}^c(r_1\ ;\ r_2) = \mathcal{T}^c(r_1)\ ;\mathcal{T}^c(r_2)$
- $\mathcal{T}^c(r_1 : r_2) = \mathcal{T}^c(r_1) : \mathcal{T}^c(r_2)$
- $\mathcal{T}^c(r_1 \mid r_2) = \mathcal{T}^c(r_1) \mid \mathcal{T}^c(r_2)$
- $\mathcal{T}^c(r_1\ \&\&\ r_2) = \mathcal{T}^c(r_1)\ \&\&\ \mathcal{T}^c(r_2)$
- $\mathcal{T}^c(r[*]) = \{\mathcal{T}^c(r)\}[*]$
- $\mathcal{T}^c(r@c_1) = \{\neg c_1[*];\ \{c_1{:}\mathcal{T}^{c_1}(r)\}\}$

The rewrites for formulas uses the auxiliary definition $f_1 \Rightarrow f_2\ =\ \neg f_1 \vee f_2$ where $f_1 \vee f_2\ =\ \neg(\neg f_1 \wedge \neg f_2)$.

The rewriting rules for PSL/Sugar formulas are:

- $\mathcal{T}^c(b) = b$
- $\mathcal{T}^c(\neg f) = \neg\mathcal{T}^c(f)$
- $\mathcal{T}^c(f_1 \wedge f_2) = (\mathcal{T}^c(f_1) \wedge \mathcal{T}^c(f_2))$
- $\mathcal{T}^c(\mathsf{X!} f) = \mathsf{X!}\ [\neg c\ \mathsf{U}\ (c \wedge \mathcal{T}^c(f))]$
- $\mathcal{T}^c(f_1\ \mathsf{U}\ f_2) = [(c \rightarrow \mathcal{T}^c(f_1))\ \mathsf{U}\ (c \wedge \mathcal{T}^c(f_2))]\ ]$
- $\mathcal{T}^c(\{r\}(f)) = \{\mathcal{T}^c(r)\}([\neg c\ \mathsf{U}\ (c \wedge \mathcal{T}^c(f)\ ])$
- $\mathcal{T}^c(\{r_1\}\Rightarrow\{r_2\}!) = \{\mathcal{T}^c(r_1)\}\Rightarrow\{\mathcal{T}^c(r_2)\}!)$
- $\mathcal{T}^c(\{r_1\}\Rightarrow\{r_2\}) = \{\mathcal{T}^c(r_1)\}\Rightarrow\{\mathcal{T}^c(r_2)\})$
- $\mathcal{T}^c(f\ \text{abort}\ b) = \mathcal{T}^c(f)\ \text{abort}\ (c \wedge b)$
- $\mathcal{T}^c(f@c_1!) = [\neg c_1\ \mathsf{U}\ (c_1 \wedge \mathcal{T}^{c_1}(f))]$

These rewrites are represented in HOL by recursively defining terms `S_CLOCK_COMP` c r (pretty printed as $\mathcal{T}^c(r)$) for SEREs and `F_CLOCK_COMP` c f (pretty printed as $\mathcal{T}^c(f)$) for FL formulas.

The definition for SEREs is the following conjunction of equations that defines `S_CLOCK_COMP` c r by recursion on the structure of r:

$(\mathcal{T}^c(b) = (\neg c[*]; c \wedge b))\ \wedge$
$(\mathcal{T}^c(r_1; r_2) = \mathcal{T}^c(r_1); \mathcal{T}^c(r_2))\ \wedge$
$(\mathcal{T}^c(r_1 : r_2) = \mathcal{T}^c(r_1) : \mathcal{T}^c(r_2))\ \wedge$
$(\mathcal{T}^c(\{r_1\} \mid \{r_2\}) = \{\mathcal{T}^c(r_1)\} \mid \{\mathcal{T}^c(r_2)\})\ \wedge$
$(\mathcal{T}^c(\{r_1\}\&\&\{r_2\}) = \{\mathcal{T}^c(r_1)\}\&\&\{\mathcal{T}^c(r_2)\})\ \wedge$
$(\mathcal{T}^c(r[*]) = \mathcal{T}^c(r)[*])\ \wedge$
$(\mathcal{T}^c(r@c_1) = (\neg c_1[*]; c_1 : \mathcal{T}^{c_1}(r)))$

For formulas, `F_CLOCK_COMP` c f is defined by recursion on the structure of f:

$(\mathcal{T}^c(b) = b)\ \wedge$
$(\mathcal{T}^c(\neg f) = \neg\mathcal{T}^c(f))\ \wedge$
$(\mathcal{T}^c(f_1 \wedge f_2) = \mathcal{T}^c(f_1) \wedge \mathcal{T}^c(f_2))\ \wedge$
$(\mathcal{T}^c(X!\ f) = X!\ ([\neg c\ U\ (c \wedge \mathcal{T}^c(f))]))\ \wedge$
$(\mathcal{T}^c([f_1\ U\ f_2]) = [(c \Rightarrow \mathcal{T}^c(f_1))\ U\ (c \wedge \mathcal{T}^c(f_2))])\ \wedge$
$(\mathcal{T}^c(\{r\}(f)) = \{\mathcal{T}^c(r)\}([\neg c\ U\ (c \wedge \mathcal{T}^c(f))]))\ \wedge$
$(\mathcal{T}^c(\{r_1\} \mapsto \{r_2\}!) = \{\mathcal{T}^c(r_1)\} \mapsto \{\mathcal{T}^c(r_2)\}!)\ \wedge$
$(\mathcal{T}^c(\{r_1\} \mapsto \{r_2\}) = \{\mathcal{T}^c(r_1)\} \mapsto \{\mathcal{T}^c(r_2)\})\ \wedge$
$(\mathcal{T}^c(f\ abort\ b) = \mathcal{T}^c(f)\ abort\ (c \wedge b))\ \wedge$
$(\mathcal{T}^c(f@c_1!) = [\neg c_1\ U\ (c_1 \wedge \mathcal{T}^{c_1}(f))])$

To validate these rewrites and further check the clocked and unclocked semantics we proved

$\vdash\ \forall r\ w\ c.\ w \models\!\!\!\models^{c} r\ =\ w \models \mathcal{T}^c(r)$

and

$\vdash\ \forall f\ w\ c.\ w_0 \models c \Rightarrow (w \models\!\!\!\models^{c} f = w \models \mathcal{T}^c(f))$

and hence, since $\vdash\ \forall w.\ w_0 \models \mathrm{T}$, we have two unconditional equations:

$\vdash\ \forall r\ w.\ w \models\!\!\!\models^{\mathrm{T}} r\ =\ w \models \mathcal{T}^{\mathrm{T}}(r)$
$\vdash\ \forall f\ w.\ w \models\!\!\!\models^{\mathrm{T}} f\ =\ w \models \mathcal{T}^{\mathrm{T}}(f)$

which allow us to compute the 'top level' semantics of any construct by first applying the rewrites and then using the unclocked semantics.

5.3. Remarks on the proofs

The proofs of the properties described in Sections 5.2.1 and 5.2.2 took several months of manual effort (spread out over about a year), they were thus far from automatic. The general structure of the proofs was structural induction over SEREs and/or formulas, followed by a case split on finite and infinite paths, followed by invoking first order provers and arithmetical decision procedures. A substantial effort went into developing numerous lemmas needed to handle the details. The first proofs were about Sugar 2.0 and were significantly more complex than the subsequent proofs of similar properties about PSL. Sugar 2.0 had strong and weak clocking as separate primitive notions, whereas PSL only has strong clocking as primitive, and weak clocking is defined. This meant, for example, that many theorems about Sugar 2.0 required separate mutually inductive proofs for strong and weak clocking. Also, lemmas using the well-ordering of the natural numbers were needed for validating the correctness of the clock removal rewrites for Sugar 2.0, but not for PSL. It is hard to extract general insights from the mass of detail: many of the proofs were complex and tedious to perform, with large numbers of cases, but there was nothing particularly striking about their structure. As we went through the cases many sequence of inference steps were repeated. When we redid the proofs for PSL some of these were packaged into proof scripts (HOL tactics) so that the input to the HOL system became less bulky.

At the start of the proof effort various 'tightenings' of the semantics were developed in collaboration with the Sugar designers. These were mainly to ensure that quantifications over indices were restricted so that the uses of the indices were always meaningful. For example, we added the requirement that all terms $w^{(i,j)}$ occurred in a context where $i \leq j$, so that the arbitrary value of $w^{(i,j)}$ when $i > j$ was never invoked. This process uncovered some minor errors such as "$>$" occurring when there should have been "$\geq$".

After the initial phase, more errors were uncovered as a result of unsolvable subgoals being generated, however no major conceptual problems were uncovered, just 'semantic typos'.

6. Conclusions and future work

It was quite straightforward to use the semi-formal semantics in the PSL/Sugar documentation to create a deep embedding of the whole language kernel. Attempting to prove some simple 'sanity checking' lemmas with a theorem prover quickly revealed bugs in the translated semantics (and in the original). Further probing revealed more bugs. It is hoped that the semantics in the HOL logic that we now have is correct, but one cannot be absolutely sure, and past experience suggests caution!

PSL/Sugar is a language intended for representing the kind of properties currently checked using assertion based verification of hardware (ABV [Coh03]). From a theorem proving perspective it is quite low level, and is inadequate to specify devices using higher level data-types like real numbers (e.g. the specification of floating point algorithms requires proper real numbers [Har00]). In future research we hope to combine specifications using higher level mathematical theories for data operations with PSL for sequential behaviour. We also hope to experiment with augmenting PSL with additional constructs based on Interval Temporal Logic (ITL) [HMM83, ITL]. We have already explored adding ITL constructs, like the chop-operator, to PSL formulas and have shown that if this is done then SEREs can be encoded as formulas whilst preserving the original semantics.

Presumably relatively few people will actually read the formal semantics, so one might ask what is the point of subjecting it to the kind of scrutiny described here. Some discussion of this was given in Section 2. Our view is that a formal semantics is more than just documentation. Just as a formal syntax can be fed into tools to generate useful products, like parsers, so we hope that a formal semantics can be fed into tools to generate useful products, like simulation monitors. Such products will be standards compliant by construction, but more research on logic programming methodology is needed before they can be practical [GHS03].

7. Acknowledgements

The work described here would not have been possible without the help of the Sugar team of Cindy Eisner and Dana Fisman from IBM. They devised the PSL/Sugar language and the semantics that is formalised here in higher order logic. They also supplied the LaTeX sources for the extracts of the LRM shown in

the framed boxes. Both jointly and individually they patiently answered numerous email questions in great detail, supplied valuable comments and corrections to an earlier version of this paper, and suggested lemmas and ways of modifying the HOL semantics to get the proofs described in Section 5 to go through.

The anonymous referees spotted several errors in the submitted version of this paper, and made many suggestions, both detailed and general, for improving the presentation. The submitted paper contained a detailed discussion of some unfinished proofs about Sugar 2.0. These proofs were all subsequently completed, but when the semantics was simplified by Accellera, as Sugar 2.0 evolved into PSL, much of the technical discussion in the submitted paper no longer applied. Rather than discuss the details of proofs for the obsolete Sugar 2.0 semantics, we provide here a higher level perspective on the issues, whilst trying to incorporate the spirit of the referees detailed points.

A preliminary version of this paper based on the original Sugar 2.0 semantics was presented as a work-in-progress contribution at TPHOLs2002 under the title *Using HOL to study Sugar 2.0 semantics* and appeared in the NASA Conference Proceedings CP-2002-21173.

Keith Wansbrough's HOL-to-LaTeX tool for typesetting the ASCII syntax of HOL terms simplified the task of writing this paper and ensures that the various semantics shown are faithful representations of the HOL sources.

References

[ABG+00] Y. Abarbanel, I. Beer, L. Gluhovsky, S. Keidar, and Y. Wolfsthal. FoCs - automatic generation of simulation checkers from formal specifications. In *Proc.* 12^{th} *International Conference on Computer Aided Verification (CAV)*, LNCS 1855. Springer-Verlag, 2000.

[Acc] Accellera Property Specification Language Reference Manual, Version 1.01. At `http://www.eda.org/vfv/docs/psl_lrm-1.01.pdf`.

[BBDE+01] 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.

[BGG+92] 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.

[Coh03] Ben Cohen. *Using PSL/Sugar with Verilog and VHDL: Guide to Property Specification Language for Assertion-Based Verification*. VhdlCohen Publishing, May 2003. ISBN 0-9705394-4-4.

[EF02] C. Eisner and D. Fisman. Sugar 2.0 proposal presented to the Accellera Formal Verification Technical Committee, March 2002. At `http://www.haifa.il.ibm.com/projects/verification/sugar/Sugar_2.0_Accellera.ps`.

[GHS03] Mike Gordon, Joe Hurd, and Konrad Slind. Executing the formal semantics of the Accellera Property Specification Language by mechanised theorem proving. In Daniel Geist and Enrico Tronci, editors, *Proc.* 12^{th} *Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME 2003)*, Lecture Notes in Computer Science. Springer-Verlag, October 2003. 21 - 24 October 2003, University of L'Aquila, Computer Science Department, L'Aquila, Italy.

[GM93] M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: a theorem-proving environment for higher-order logic*. Cambridge University Press, 1993.

[Har00] John Harrison. Formal verification of IA-64 division algorithms. In M. Aagaard and J. Harrison, editors, *Theorem Proving in Higher Order Logics: 13th International Conference, TPHOLs 2000*, volume 1869 of *Lecture Notes in Computer Science*, pages 234–251. Springer-Verlag, 2000.

[HMM83] 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.

[ITL] ITL home page, Antonio Cau, Ben Moszkowski and Hussein Zedan, Software Technology Research Laboratory, De Montford University. At `http://www.cms.dmu.ac.uk/~cau/itlhomepage/`.

[Mat] W3C Math Home. At `http://www.w3.org/Math`.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[Ope] OpenMath website. At `http://www.openmath.org`.

[RSS95] 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.

[SH99] K. Schneider and D. Hoffmann. A HOL Conversion for Translating Linear Time Temporal Logic to omega-Automata. In *Theorem Proving in Higher Order Logics (TPHOLs99)*, number 1690 in Lecture Notes in Computer Science. Springer-Verlag, 1999.