Under consideration for publication in Formal Aspects of Computing

Validating *Sugar 2.0* semantics using automated reasoning

Michael J. C. Gordon

Abstract.
The Accellera organisation selected *Sugar 2.0*, IBM's formal specification language, as the basis for a standard to "drive assertion-based verification" in the electronics industry. *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 static verification (e.g. model checking) and dynamic verification (e.g. simulation). We motivate and describe a deep semantic embedding of *Sugar 2.0* in the version of higher order logic supported by the HOL 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, Semantics, Formal verification, Model checking, Theorem proving, 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 standardization amongst developers, users and academia,
- Promoting use of the standard, and
- Assuring interoperability for the property specification language amongst 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 2.0*.

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 ITL-like [ITL] constructs called *Sugar Extended Regular Expressions*. *Sugar 2.0* retains CTL constructs in its *Optional Branching Extension* (OBE), but this is de-emphasised in the defining document. 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).

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 by test-bench tools.

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 [GM93].

In Section 2 we discuss a number of motivations for embedding Sugar in HOL. In Section 3, higher order logic and semantic embedding are 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. Progress so far in analysing the semantics using the HOL system is discussed in Section 5. Finally, there is a short section of conclusions and future plans. The initial and corrected complete semantics of *Sugar 2.0* in higher order logic are given in an appendix.

2. Why embed Sugar in HOL?

There are several reasons for embedding *Sugar 2.0* in a machine-processable formal logic.

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 (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.

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 correct semantics. See the end of Section 5.3 for some examples.

2.2. 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 'golden semantics'. One might think of using some XML-based representation of mathematical content. Although there are well-developed syntactic representations like MathML [Mat] there is currently not much support for semantic representations. See the end of Section 5.4 for a bit more 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.

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 *Sugar 2.0* 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.

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 [FoC], but would use compilation by theorem proving to ensure semantic equivalence between the executable checker and the source property.

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 *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.[1]

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 we use 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 or four types: constants, variables, function applications $t_1\ t_2$ (sometimes written $t_1(t_2)$) 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 a definitional mechanism that guarantees no new inconsistency is introduced. Defined constants are, for example, numerals $0, 1, 2$ etc. and strings `"a"`,

[1] `http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/hol/hol98/examples/Sugar2/`

`"b"`, `"ab"` etc. To represent Sugar in HOL we define constants to represent the syntax constructors for the various constructs and then define semantic constants that specify when formulas are true in a model. 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 $\texttt{Pair}\ x\ y$ means $((\texttt{Pair}\ x)\ y)$, so `Pair` is 'curried'). If P is a function that returns a truth-value (i.e. a predicate), then P can be thought of 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.[2] 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 typechecking 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 typechecking, however, $\neg\texttt{T}$ 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 identitity functions with an instance $\lambda x : ty.\ x$ for each type ty. The semantics of *Sugar 2.0* is parametized on the sets of atomic propositions and the set of states. This is represented in HOL by having type variables *aprop* and *state* which can be instantiated to the types modelling atomic propositions and states of particular applications.

The semantics of Sugar formulas are defined with respect to a model `M`, which is a quintuple (S, S_0, R, P, L), where S is a set of states, S_0 is the set of initial states (a subset of S), $R \subseteq S \times S$ is the transition relation, P is a non-empty set of atomic propositions, and `L` is the valuation that maps each state s to the set $L(s)$ of atomic propositions that hold at s.

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 a type *state*, so $S : state \rightarrow bool$. Normally every value of type *state* is a valid state, but allowing the possibility that S correspond to a subset of *state* 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 $(\exists s.\ S_0\ s) \wedge (\forall s.\ S_0\ s \Rightarrow S\ s)$. 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 each state s to a the set of propositions true in s. Thus $L(s) : aprop \rightarrow bool$, and so $L : state \rightarrow (aprop \rightarrow bool)$. 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 $(state, aprop)model$ abbreviate the type of M, then $M : (state, aprop)model$. Here *model* is a binary type constructor.

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

Sugar 2.0 has four kinds of syntactic constructs: Boolean Expressions, Sugar Extended Regular Expressions (SEREs), Foundation Language (FL) formulas and Optional Branching Extension (OBE) formulas. We define four types in the HOL logic to represent each of these kinds of constructs, so that a term with one of these four syntactic types will represent a corresponding construct in the *Sugar 2.0* language.

In the rest of this section we explain how the syntax and a simplified semantics of *Sugar 2.0* is represented in the HOL logic. Because Boolean expressions are very simple, we will explain the syntax and semantics of these in detail. The way the syntax and semantics of the other three kinds of constructs are represented is similar (though the details are very different). In Section 4 we discuss the full unsimplified semantics of *Sugar 2.0*.

3.1. Boolean expressions in Sugar

The syntax of boolean expressions is as follows:

- *Every atomic proposition p 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
```

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 type *(string)bexp*.

The semantics of boolean expressions are specified by defining $l \models b$, where $l \subseteq P$, i.e. l is a set of atomic propositions. The definition is given in the Accellera report by:

- $l \models p \Longleftrightarrow p \in l$
- $l \models (b) \Longleftrightarrow l \models b$
- $l \models \neg b \Longleftrightarrow l \not\models b$
- $l \models b_1 \wedge b_2 \Longleftrightarrow l \models b_1$ and $l \models b_2$

This is represented in HOL by defining a semantic valuation function

$$\texttt{B\_SEM} : (aprop, state)model {\rightarrow} (aprop {\rightarrow} bool) {\rightarrow} (aprop)bexp {\rightarrow} bool$$

such that `B_SEM` (S, S_0, R, P, L) l b is true iff b is built from propositions in P and it is true with respect to the truth assignment l.

Define the function `getP` by `getP`$(S, S_0, R, P, L) \ = \ P$, then the input to HOL to represent the semantics of boolean expressions is:

```
Define
 ‘(B_SEM M l (B_PROP p)      = p ∈ (getP M) ∧ p ∈ l)
  ∧
  (B_SEM M l (B_NOT b)       = ¬(B_SEM M l b))
  ∧
  (B_SEM M l (B_AND(b1,b2)) = B_SEM M l b1 ∧ B_SEM M l b2)‘
```

If we write (M, l $\models$ b) for B_SEM M l b then the semantics above can be written more readably as:

```
((M, l ⊨ p)       = p ∈ P ∧ p ∈ l)
∧
((M, l ⊨ ¬b)      = ¬(M, l ⊨ b))
∧
((M, l ⊨ b1 ∧ b2) = (M, l ⊨ b1) ∧ (M, l ⊨ b2))
```

Note that the symbol $\wedge$ is overloaded: the occurrence in $\mathtt{b}_1 \wedge \mathtt{b}_2$ is part of the boolean expression syntax of Sugar, but the other occurrences are conjunction in higher order logic.

In the rest of this section we describe the syntax and a simplified semantics for the parts of *Sugar 2.0* corresponding to Interval Temporal Logic (ITL), Linear Temporal Logic (LTL) and Computation Tree Logic (CTL). We do not give the input to the HOL system, but use the readable notation similar to that just given for boolean expressions.

3.2. ITL: Sugar Extended Regular Expressions (SEREs)

Interval Temporal Logic (ITL) provides formulas that are true or false of intervals. 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). Each SERE specifies a set of ‘words’, where a word is a finite sequence of sets of atomic propositions. If M is a model (S, S_0, R, P, L), then a word of the model has the form $l_1 l_2 \cdots l_n$, where $l_i \subseteq P$ (for $1 \leq i \leq n$). Such words are represented in HOL using lists, a defined datatype.

Each state $s \in S$ specifies a set $L(s)$ of atomic propositions, so each sequence of states $s_1 s_2 \cdots s_n$ determines a word $L(s_1)L(s_2)\cdots L(s_n)$, which is denoted by $\hat{L}(s_1 s_2 \cdots s_n)$. We use $\hat{L}_M$ for the $\hat{L}$ function of a model M. Sequences of states correspond to the intervals of ITL. Each ITL formula specifies a set of intervals, namely the intervals for which the formula is true. Each *Sugar 2.0* SERE specifies a set of intervals too, namely the intervals that map by $\hat{L}$ to words for which the SERE is true.

If r, $\mathtt{r}_1$, $\mathtt{r}_2$ etc. range over SEREs and b ranges over boolean expressions, then the syntax of *Sugar 2.0* SEREs is given by:

r	::=	b	(Boolean formula)
	|	$\{\mathtt{r}_1\}$ | $\{\mathtt{r}_2\}$	(Disjunction)
	|	$\mathtt{r}_1$; $\mathtt{r}_2$	(Concatenation)
	|	$\mathtt{r}_1$: $\mathtt{r}_2$	(Fusion: ITL’s chop)
	|	$\{\mathtt{r}_1\}$ && $\{\mathtt{r}_2\}$	(Length matching conjunction)
	|	$\{\mathtt{r}_1\}$ & $\{\mathtt{r}_2\}$	(Flexible matching conjunction)
	|	r[$*$]	(Repeat)

The semantics of SEREs is defined by a semantic function S_SEM such that S_SEM M w r if true iff word 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.

```
((M, w ⊨ b) =
```

```
    ∃l. (w = [l]) ∧ (M, l ⊨ b))
  ∧
  ((M, w ⊨ r1;r2) =
    ∃w1 w2. (w = w1w2) ∧ (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 = w1w2) ∧
            (((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)
```

The semantics of SERE's is explained in detail in the *Sugar 2.0* documentation [Sugb].

3.3. LTL: Sugar Foundation Language (FL)

Regular expressions are used to match finite sequence of states. To specify infinite sequences of states, or paths, *Sugar 2.0* provides a kernel of linear temporal logic (LTL) constructs called the Sugar Foundation Language (FL). Each FL formula determines a set of paths.

A path π is represented in HOL as a function from the natural numbers *num* to states *state*, thus π : *num*→*state*. 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 *Sugar 2.0* kernel combines standard LTL notation with a less standard `abort` operation and some constructs using SEREs. For example, the suffix implication operator `{r}(f)` is true if `f` is true whenever `r` has just been matched. More precisely `{r}(f)` is true of a path π if `f` is true of every path π^j where for some $i < j$ the interval $\pi_i \pi_{i+1} \cdots \pi_j$ matches `r` (i.e. $\hat{L}(\pi_i \pi_{i+1} \cdots \pi_j) \models$ `r`). 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. [Suga, Section 4.11]).

`f`	::=	`p`	(Atomic formula)
	|	`¬f`	(Negation)
	|	$f_1 \wedge f_2$	(Conjunction)
	|	`X!f`	(Successor)
	|	$[f_1$ `U` $f_2]$	(Until)
	|	`{r}(f)`	(Suffix implication)
	|	$\{r_1\}$ `|->` $\{r_2\}$`!`	(Strong suffix implication)
	|	$\{r_1\}$ `|->` $\{r_2\}$	(Weak suffix implication)
	|	`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.

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 $\models$ 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 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^{(\mathtt{0,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^{(\mathtt{0,j})}))\ \models\ \mathtt{r1})\\
\qquad\quad \Rightarrow\ \exists\mathtt{k}.\ \mathtt{j}\ \leq\ \mathtt{k}\ \wedge\ (\mathtt{M},\ (\hat{\mathtt{L}}_{\mathtt{M}}\ (\pi^{(\mathtt{j,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^{(\mathtt{0,j})}))\ \models\ \mathtt{r1})\\
\qquad\quad \Rightarrow\ (\exists\mathtt{k}.\ \mathtt{j}\ \leq\ \mathtt{k}\ \wedge\ (\mathtt{M},\ (\hat{\mathtt{L}}_{\mathtt{M}}\ (\pi^{(\mathtt{j,k})}))\ \models\ \mathtt{r2}))\\
\qquad\qquad \vee\\
\qquad\qquad \forall\mathtt{k}.\ \mathtt{j}\ \leq\ \mathtt{k}\ \Rightarrow\ \exists\mathtt{w}.\ (\mathtt{M},\ (\hat{\mathtt{L}}_{\mathtt{M}}\ (\pi^{(\mathtt{j,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{'}.\ (\mathtt{M},\ \pi^{\mathtt{j}}\ \models\ \mathtt{b})\ \wedge\ (\mathtt{M},\ \pi^{(\mathtt{0,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)

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 *Sugar 2.0* contains constructs from CTL called the Optional Branching Extension (OBE). The syntax of the *Sugar 2.0* OBE is completely standard.

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

```
((M, s ⊨ b) = (M, L_M(s) ⊨ b))
∧
((M, s ⊨ ¬f) = ¬(M, s ⊨ f))
∧
((M, s ⊨ f1 ∧ f2) = (M, s ⊨ f1) ∧ (M, s ⊨ f2))
∧
((M, s ⊨ EX f) =
  ∃π. Path M π ∧ (π_0 = s) ∧ (M, π_1 ⊨ f))
∧
((M, s ⊨ [f1 U f2]) =
  ∃π. Path M π ∧ (π_0 = s) ∧
      (M, π_k ⊨ f2) ∧ ∀j. j < k ⇒ (M, π_j ⊨ f1))
∧
((M, s ⊨ EG f) =
  ∃π. Path M π ∧ (π_0 = s) ∧ ∀j. (M, π_j ⊨ f))
```

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. 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 [Suga] we have defined types *(aprop)bexp*, *(aprop)sere*, *(aprop)fl* and *(aprop)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.

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 (M, w $\overset{c}{\models}$ r) instead of the unclocked (M, w $\models$ 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

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

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

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

`b@c`	$\longrightarrow$	`{¬c[*]; c∧b}`
`{r1;r2}@c`	$\longrightarrow$	`{r1@c};{r2@c}`
`{r1:r2}@c`	$\longrightarrow$	`{r1@c}:{r2@c}`
`{{r1}|{r2}}@c`	$\longrightarrow$	`{r1@c}|{r2@c}`
`{{r1}&&{r2}}@c`	$\longrightarrow$	`{r1@c}&&{r2@c}`
`{{r1}&{r2}}@c`	$\longrightarrow$	`{r1@c}&{r2@c}`
`r[*]@c`	$\longrightarrow$	`{r@c}[*]`
`r@c1@c`	$\longrightarrow$	`r1@c1`

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 `{¬c1[*]; c1∧b}@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:[3]

```
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 [Suga, Appendix B.1]. We are currently in the process of trying to validate the clocking rewrites, see Section 5.3.
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

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

$(\mathtt{M},\ \pi\ \overset{\mathtt{c}}{\models}\ \mathtt{f})$
and the strong clocking by defining
$(\mathtt{M},\ \pi\ \overset{\mathtt{c!}}{\models}\ \mathtt{f})$
The complete semantics is given in the appendix, but here is the semantics of boolean expressions `b`:

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

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

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

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[4] 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` $\pi\ n$ 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 "$\forall i \in \mathtt{pl}\ \pi.\ \cdots\ i\ \cdots$" and "$\exists i \in \mathtt{pl}\ \pi.\ \cdots\ i\ \cdots$" 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.

$$\begin{array}{l}((\mathtt{M},\ \pi\ \models\ \mathtt{b})\ =\ (\mathtt{M},\ \mathtt{L}_{\mathtt{M}}(\pi_0)\ \models\ \mathtt{b}))\\ \wedge\\ ((\mathtt{M},\ \pi\ \models\ \neg\mathtt{f})\ =\ \neg(\mathtt{M},\ \pi\ \models\ \mathtt{f}))\\ \wedge\\ ((\mathtt{M},\ \pi\ \models\ \mathtt{f1} \wedge \mathtt{f2})\ =\ (\mathtt{M},\ \pi\ \models\ \mathtt{f1})\ \wedge\ (\mathtt{M},\ \pi\ \models\ \mathtt{f2}))\\ \wedge\\ ((\mathtt{M},\ \pi\ \models\ \mathtt{X!\,f})\ =\ \mathtt{pl}\ \pi\ 1\ \wedge\ (\mathtt{M},\ \pi^1\ \models\ \mathtt{f}))\\ \wedge\\ ((\mathtt{M},\ \pi\ \models\ \mathtt{[f1\ U\ f2]})\ =\\ \quad \exists \mathtt{k} \in \mathtt{pl}\,\pi.\\ \quad\ (\mathtt{M},\ \pi^{\mathtt{k}}\ \models\ \mathtt{f2})\ \wedge\ \forall \mathtt{j} \in \mathtt{pl}\,\pi.\ \mathtt{j}\ <\ \mathtt{k}\ \Rightarrow\ (\mathtt{M},\ \pi^{\mathtt{j}}\ \models\ \mathtt{f1}))\end{array}$$

[4] 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.

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

This semantics has evolved from an existing unpublished semantics[5] 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:

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

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`.

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

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

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

$$
\begin{array}{l}
\vdash\ \mathtt{FirstRise\ M}\,\pi\,\mathtt{T\ i}\ =\ (\mathtt{i} = 0) \\
\vdash\ \mathtt{i} \leq \mathtt{j}\ \Rightarrow\ (\mathtt{NextRise\ M}\,\pi\,\mathtt{T\ (i,j)}\ =\ (\mathtt{i} = \mathtt{j}))
\end{array}
$$

[5] Personal communication from Cindy Eisner and Dana Fisman of IBM.

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 $\mathtt{T}$ 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 $\mathtt{T}$, 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:

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

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

Instantiating `c` to $\mathtt{T}$ and using the corollary about `FirstRise` yields

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

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

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

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

so that we get

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

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

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`, $\{\mathtt{r}\}(\mathtt{f})$, `{r1}|->{r2}` and `f abort b`. With these changes, we used structural induction to prove:[6]

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

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.

[6] 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 $\stackrel{\texttt{c}}{\models}$ and, for formulas, $\stackrel{\texttt{c!}}{\models}$;
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[7]

$$\texttt{SClockImp : clock} \rightarrow \texttt{sere} \rightarrow \texttt{sere}$$
$$\texttt{FClockImp : clock} \rightarrow \texttt{fl} \rightarrow \texttt{fl}$$

We can then attempt to prove that

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

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

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

are harder, and we have not yet finished proving these.[8] To see the complexity involved consider the rewrite for weakly clocked conjunctions [Suga, page 67]:

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

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:

$$\texttt{f1} \vee \texttt{f2} = \neg(\neg\texttt{f1} \wedge \neg\texttt{f2})$$
$$\texttt{G f} = \neg\texttt{F}(\neg\texttt{f})$$

and `F` is defined by

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

Let us define

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

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

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

[7] We are glossing over details here, like what the type `clock` exactly is.
[8] As 5 September, 2002, we have completed the cases for all FL formulas except `[f1 U f2]` and `X!f`, and discovered one error in the unclocked semantics.

For formula conjunction we want to prove:

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

where the first $\wedge$ is in higher order logic and the one in $\mathtt{f1}\ \wedge\ \mathtt{f2}$ is part of the Sugar formula syntax. Using the lemmas below about $\vee$ and the unclocked semantics of the defined operators `W`, `G` and `F`, it is not too hard to prove the desired result about conjunctions.

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

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

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

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

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

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

Besides helping with the proof of the correctness of the rewrites for conjunctions, 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 $j \geq \mathtt{length}\ \pi$.[9] 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

---

[9] 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*.

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 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, but current infrastructure is not quite ready today (2002). 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.

6. Conclusions and future work

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 work described here would not have been possible without the help of the *Sugar 2.0* team of Cindy Eisner and Dana Fisman of IBM. 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.

A preliminary version of this paper 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.

References

[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.

[FoC] See web page: `http://www.haifa.il.ibm.com/projects/verification/focs`.

[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.

[ITL] See web page: `http://www.cms.dmu.ac.uk/~cau/itlhomepage/`.

[Mat] See web page: `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] See web page: `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.

[Suga] See web page: `www.haifa.il.ibm.com/projects/verification/sugar/Sugar\_2.0\_Accellera.%ps`.

[Sugb] See web page: `www.haifa.il.ibm.com/projects/verification/sugar/literature.html`.

APPENDIX: Current HOL semantics

The following four sub-sections are the manually typeset HOL semantics of *Sugar 2.0*. This semantics is our current working version and incorporates currections to the bugs we have found.

7.1. Boolean expressions

The semantics below is identical to that given earlier in Section 3.

$$
\begin{array}{ll}
((\mathtt{M},\ \mathtt{l} \models \mathtt{p}) & = p \in \mathtt{P}_M \wedge p \in \mathtt{l}) \\
\wedge & \\
((\mathtt{M},\ \mathtt{l} \models \mathrm{T}) & = \mathtt{T}) \\
\wedge & \\
((\mathtt{M},\ \mathtt{l} \models \neg \mathtt{b}) & = \neg(\mathtt{M},\ \mathtt{l} \models \mathtt{b})) \\
\wedge & \\
((\mathtt{M},\ \mathtt{l} \models \mathtt{b1} \wedge \mathtt{b2}) & = (\mathtt{M},\ \mathtt{l} \models \mathtt{b1}) \wedge (\mathtt{M},\ \mathtt{l} \models \mathtt{b2}))
\end{array}
$$

7.2. Sugar Extended Regular Expressions

The semantics of SEREs expressions is given by defining a semantic function `S_SEM` such that `S_SEM M w c r` if true iff `w` is in the language of the extended regular expression `r` clocked with `c`.
We write $(\mathtt{M},\ \mathtt{w} \stackrel{\mathtt{c}}{\models} \mathtt{r})$ for `S_SEM M w c 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 $P(\mathtt{w})$ holds for every `w` in `wlist`.

$$
\begin{array}{l}
((\mathtt{M},\ \mathtt{w} \stackrel{\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})) \\
\wedge \\
((\mathtt{M},\ \mathtt{w} \stackrel{\mathtt{c}}{\models} \mathtt{r1;r2}) = \\
\quad \exists \mathtt{w1}\ \mathtt{w2}.\ (\mathtt{w} = \mathtt{w1}\,\mathtt{w2}) \wedge (\mathtt{M},\ \mathtt{w1} \stackrel{\mathtt{c}}{\models} \mathtt{r1}) \wedge (\mathtt{M},\ \mathtt{w2} \stackrel{\mathtt{c}}{\models} \mathtt{r2})) \\
\wedge \\
((\mathtt{M},\ \mathtt{w} \stackrel{\mathtt{c}}{\models} \mathtt{r1:r2}) = \\
\quad \exists \mathtt{w1}\ \mathtt{w2}\ \mathtt{l}.\ (\mathtt{w} = \mathtt{w1}\,[\mathtt{l}]\,\mathtt{w2}) \wedge \\
\qquad\qquad (\mathtt{M},\ (\mathtt{w1}\,[\mathtt{l}]) \stackrel{\mathtt{c}}{\models} \mathtt{r1}) \wedge (\mathtt{M},\ ([\mathtt{l}]\,\mathtt{w2}) \stackrel{\mathtt{c}}{\models} \mathtt{r2})) \\
\wedge \\
((\mathtt{M},\ \mathtt{w} \stackrel{\mathtt{c}}{\models} \{\mathtt{r1}\}|\{\mathtt{r2}\}) = \\
\quad (\mathtt{M},\ \mathtt{w} \stackrel{\mathtt{c}}{\models} \mathtt{r1}) \vee (\mathtt{M},\ \mathtt{w} \stackrel{\mathtt{c}}{\models} \mathtt{r2})) \\
\wedge \\
((\mathtt{M},\ \mathtt{w} \stackrel{\mathtt{c}}{\models} \{\mathtt{r1}\}\&\&\{\mathtt{r2}\}) = \\
\quad (\mathtt{M},\ \mathtt{w} \stackrel{\mathtt{c}}{\models} \mathtt{r1}) \wedge (\mathtt{M},\ \mathtt{w} \stackrel{\mathtt{c}}{\models} \mathtt{r2})) \\
\wedge \\
((\mathtt{M},\ \mathtt{w} \stackrel{\mathtt{c}}{\models} \{\mathtt{r1}\}\&\{\mathtt{r2}\}) = \\
\quad \exists \mathtt{w1}\ \mathtt{w2}.\ (\mathtt{w} = \mathtt{w1}\,\mathtt{w2}) \wedge \\
\qquad\quad (((\mathtt{M},\ \mathtt{w} \stackrel{\mathtt{c}}{\models} \mathtt{r1}) \wedge (\mathtt{M},\ \mathtt{w1} \stackrel{\mathtt{c}}{\models} \mathtt{r2})) \\
\qquad\quad \vee \\
\qquad\quad ((\mathtt{M},\ \mathtt{w} \stackrel{\mathtt{c}}{\models} \mathtt{r2}) \wedge (\mathtt{M},\ \mathtt{w1} \stackrel{\mathtt{c}}{\models} \mathtt{r1})))) \\
\wedge \\
((\mathtt{M},\ \mathtt{w} \stackrel{\mathtt{c}}{\models} \mathtt{r}[*]) = \\
\quad \exists \mathtt{wlist}.\ (\mathtt{w} = \mathtt{Concat}\ \mathtt{wlist}) \wedge \mathtt{Every}\ (\lambda \mathtt{w}.\ (\mathtt{M},\ \mathtt{w} \stackrel{\mathtt{c}}{\models} \mathtt{r}))\ \mathtt{wlist})
\end{array}
$$

$$
\begin{array}{l}
\wedge \\
((\mathtt{M},\ \mathtt{w}\ \overset{\mathtt{c}}{\models}\ \mathtt{r@c1}) = \\
\quad (\mathtt{M},\ \mathtt{w}\ \overset{\mathtt{c1}}{\models}\ \mathtt{r}))
\end{array}
$$

7.3. Foundation Language

We define a semantic function F_SEM such that F_SEM M π c f means FL formula f is true of path π if the current clock is c. The cases for weak (c) and strong (c!) clocks are considered separately.

We write $(\mathtt{M},\ \pi\ \overset{\mathtt{c}}{\models}\ \mathtt{r})$ for F_SEM M π c f and use the following two definitions:

$$
\mathtt{FirstRise}\ \mathtt{M}\ \pi\ \mathtt{c}\ \mathtt{i} \quad = (\mathtt{M},\ (\hat{\mathtt{L}}_{\mathtt{M}}\ (\pi^{(0,\mathtt{i})}))\ \overset{\mathrm{T}}{\models}\ \neg\mathtt{c}[*];\mathtt{c})
$$

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

The semantic clauses are then:

$$
\begin{array}{l}
((\mathtt{M},\ \pi\ \overset{\mathtt{c!}}{\models}\ \mathtt{b}) = \\
\quad \exists \mathtt{i} \in \mathtt{pl}\,\pi.\ \mathtt{FirstRise}\ \mathtt{M}\ \pi\ \mathtt{c}\ \mathtt{i} \wedge (\mathtt{M},\ \mathtt{L}_{\mathtt{M}}(\pi_{\mathtt{i}})\ \models\ \mathtt{b})) \\
\wedge \\
((\mathtt{M},\ \pi\ \overset{\mathtt{c!}}{\models}\ \neg\mathtt{f}) = \\
\quad \neg(\mathtt{M},\ \pi\ \overset{\mathtt{c}}{\models}\ \mathtt{f})) \\
\wedge \\
((\mathtt{M},\ \pi\ \overset{\mathtt{c!}}{\models}\ \mathtt{f1} \wedge \mathtt{f2}) = \\
\quad \exists \mathtt{i} \in \mathtt{pl}\,\pi.\ \mathtt{FirstRise}\ \mathtt{M}\ \pi\ \mathtt{c}\ \mathtt{i}\ \wedge \\
\qquad\qquad (\mathtt{M},\ \pi^{\mathtt{i}}\ \overset{\mathtt{c!}}{\models}\ \mathtt{f1}) \quad \wedge \\
\qquad\qquad (\mathtt{M},\ \pi^{\mathtt{i}}\ \overset{\mathtt{c!}}{\models}\ \mathtt{f2})) \\
\wedge \\
((\mathtt{M},\ \pi\ \overset{\mathtt{c!}}{\models}\ \mathtt{X!\,f}) = \\
\quad \exists \mathtt{i} \in \mathtt{pl}\,\pi.\ \mathtt{FirstRise}\ \mathtt{M}\ \pi\ \mathtt{c}\ \mathtt{i}\ \wedge \\
\qquad\qquad \exists \mathtt{j} \in \mathtt{pl}\,\pi. \\
\qquad\qquad\quad \mathtt{i} < \mathtt{j} \wedge \mathtt{NextRise}\ \mathtt{M}\ \pi\ \mathtt{c}\ (\mathtt{i+1},\mathtt{j}) \wedge (\mathtt{M},\ \pi^{\mathtt{j}}\ \overset{\mathtt{c!}}{\models}\ \mathtt{f})) \\
\wedge \\
((\mathtt{M},\ \pi\ \overset{\mathtt{c!}}{\models}\ [\mathtt{f1\ U\ f2}]) = \\
\quad \exists \mathtt{i}\ \mathtt{k} \in \mathtt{pl}\,\pi.\ \mathtt{k} \geq \mathtt{i} \qquad\qquad \wedge \\
\qquad\qquad\quad \mathtt{FirstRise}\ \mathtt{M}\ \pi\ \mathtt{c}\ \mathtt{i} \qquad \wedge \\
\qquad\qquad\quad (\mathtt{M},\ \pi^{\mathtt{k}}\ \overset{\mathrm{T}}{\models}\ \mathtt{c}) \qquad \wedge \\
\qquad\qquad\quad (\mathtt{M},\ \pi^{\mathtt{k}}\ \overset{\mathtt{c!}}{\models}\ \mathtt{f2}) \qquad \wedge \\
\qquad\qquad\quad \forall \mathtt{j} \in \mathtt{pl}\,\pi. \\
\qquad\qquad\qquad \mathtt{i} \leq \mathtt{j} \wedge \mathtt{j} < \mathtt{k}\ \wedge \\
\qquad\qquad\qquad (\mathtt{M},\ \pi^{\mathtt{j}}\ \overset{\mathrm{T}}{\models}\ \mathtt{c}) \\
\qquad\qquad\qquad \Rightarrow \\
\qquad\qquad\qquad (\mathtt{M},\ \pi^{\mathtt{j}}\ \overset{\mathtt{c!}}{\models}\ \mathtt{f1})) \\
\wedge \\
((\mathtt{M},\ \pi\ \overset{\mathtt{c!}}{\models}\ \{\mathtt{r}\}(\mathtt{f})) = \\
\quad \exists \mathtt{i} \in \mathtt{pl}\,\pi.\ \mathtt{FirstRise}\ \mathtt{M}\ \pi\ \mathtt{c}\ \mathtt{i}\ \wedge \\
\qquad\qquad \forall \mathtt{j} \in \mathtt{pl}\,\pi.\ \mathtt{i} \leq \mathtt{j} \wedge (\mathtt{M},\ (\hat{\mathtt{L}}_{\mathtt{M}}\ (\pi^{(\mathtt{i},\mathtt{j})}))\ \overset{\mathtt{c}}{\models}\ \mathtt{r}) \\
\qquad\qquad\qquad\qquad \Rightarrow \\
\qquad\qquad\qquad\qquad (\mathtt{M},\ \pi^{\mathtt{j}}\ \overset{\mathtt{c!}}{\models}\ \mathtt{f})) \\
\wedge
\end{array}
$$

```
((M, π ⊨^{c!} {r1}|->{r2}!) =
  ∃i ∈ pl π. FirstRise M π c i ∧
           ∀j ∈ pl π.
                i ≤ j ∧ (M, (L̂_M (π^{(i,j)})) ⊨^{c} r1)
                ⇒
                ∃k ∈ pl π. j ≤ k ∧ (M, (L̂_M (π^{(j,k)})) ⊨^{c} r2))
∧
((M, π ⊨^{c!} {r1}|->{r2}) =
 (M, π ⊨^{c!} {r1}|->{r2}!)
 ∨
 ((M, π ⊨^{c} {r1}|->{r2}) ∧
   ∀j ∈ pl π. ∃k ∈ pl π. j ≤ k ∧ NextRise M π c (j,k)))
∧
((M, π ⊨^{c!} f abort b) =
  ∃i ∈ pl π. FirstRise M π c i ∧
            ((M, π^i ⊨^{c!} f )
             ∨
             ∃j ∈ pl π. i < j ∧
               ∃π'.
                (M, π^j ⊨^{T} c ∧ b) ∧ (M, π^{(i,j-1)} π' ⊨^{c!} f)))
∧
((M, π ⊨^{c!} f@c1) =
  (M, π ⊨^{c1} f))
∧
((M, π ⊨^{c!} f@c1!) =
  (M, π ⊨^{c1!} f))
∧
((M, π ⊨^{c} b) =
  ∀i ∈ pl π. FirstRise M π c i ⇒ (M, L_M(π_i) ⊨ b))
∧
((M, π ⊨^{c} ¬f) =
  ¬(M, π ⊨^{c!} f))
∧
((M, π ⊨^{c} f1 ∧ f2) =
   ∀i ∈ pl π. FirstRise M π c i
             ⇒
             ((M, π^i ⊨^{c} f1) ∧ (M, π^i ⊨^{c} f2)))
∧
((M, π ⊨^{c} X! f) =
  ∀i ∈ pl π. FirstRise M π c i ⇒
           ∃j ∈ pl π.
             i < j ∧ NextRise M π c (i+1,j) ∧ (M, π^j ⊨^{c!} f))
∧
((M, π ⊨^{c} [f1 U f2]) =
  (M, π ⊨^{c!} [f1 U f2])
   ∨
  (∃k ∈ pl π.
    ∀l ∈ pl π.
      l ≥ k ⇒ (M, π^l ⊨^{T} ¬c)
                ∧
```

$$
\begin{array}{l}
\qquad\qquad\qquad\quad \forall \mathtt{j} \in \mathtt{pl}\,\pi. \\
\qquad\qquad\qquad\qquad \mathtt{j} \leq \mathtt{k} \Rightarrow (\mathtt{M},\ \pi^{\mathtt{j}} \overset{\mathrm{T}}{\models} \mathtt{c}) \Rightarrow (\mathtt{M},\ \pi^{\mathtt{j}} \overset{\mathtt{c}}{\models} \mathtt{f1}))) \\
\wedge \\
((\mathtt{M},\ \pi \overset{\mathtt{c}}{\models} \{\mathtt{r}\}(\mathtt{f})) = \\
\quad \forall \mathtt{i} \in \mathtt{pl}\,\pi.\ \mathtt{FirstRise\ M}\ \pi\ \mathtt{c\ i} \Rightarrow \\
\qquad\qquad\quad \forall \mathtt{j} \in \mathtt{pl}\,\pi.\ \mathtt{i} \leq \mathtt{j} \wedge (\mathtt{M},\ (\hat{\mathtt{L}}_{\mathtt{M}}\ (\pi^{(\mathtt{i},\mathtt{j})})) \overset{\mathtt{c}}{\models} \mathtt{r}) \\
\qquad\qquad\qquad\qquad \Rightarrow \\
\qquad\qquad\qquad\qquad (\mathtt{M},\ \pi^{\mathtt{j}} \overset{\mathtt{c}}{\models} \mathtt{f})) \\
\wedge \\
((\mathtt{M},\ \pi \overset{\mathtt{c}}{\models} \{\mathtt{r1}\}\mathtt{|->}\{\mathtt{r2}\}\mathtt{!}) = \\
\quad (\mathtt{M},\ \pi \overset{\mathtt{c!}}{\models} \{\mathtt{r1}\}\mathtt{|->}\{\mathtt{r2}\}\mathtt{!}) \\
\quad \vee \\
\quad ((\mathtt{M},\ \pi \overset{\mathtt{c}}{\models} \{\mathtt{r1}\}\mathtt{|->}\{\mathtt{r2}\}) \\
\quad\ \wedge \\
\quad\ \exists \mathtt{k} \in \mathtt{pl}\,\pi.\ \forall \mathtt{l} \in \mathtt{pl}\,\pi.\ \mathtt{l} \geq \mathtt{k} \Rightarrow (\mathtt{M},\ \pi^{\mathtt{l}} \overset{\mathrm{T}}{\models} \neg\mathtt{c}))) \\
\wedge \\
((\mathtt{M},\ \pi \overset{\mathtt{c}}{\models} \{\mathtt{r1}\}\mathtt{|->}\{\mathtt{r2}\}) = \\
\quad \forall \mathtt{i} \in \mathtt{pl}\,\pi.\ \ \mathtt{FirstRise\ M}\ \pi\ \mathtt{c\ i} \\
\qquad\qquad\quad \Rightarrow \\
\qquad\qquad\quad \forall \mathtt{j} \in \mathtt{pl}\,\pi.\ \mathtt{i} \leq \mathtt{j} \wedge (\mathtt{M},\ (\hat{\mathtt{L}}_{\mathtt{M}}\ (\pi^{(\mathtt{i},\mathtt{j})})) \overset{\mathtt{c}}{\models} \mathtt{r1}) \\
\qquad\qquad\qquad\qquad \Rightarrow \\
\qquad\qquad\qquad\qquad ((\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})})) \overset{\mathtt{c}}{\models} \mathtt{r2})) \\
\qquad\qquad\qquad\qquad\ \vee \\
\qquad\qquad\qquad\qquad\ \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}) \overset{\mathtt{c}}{\models} \mathtt{r2}))) \\
\wedge \\
((\mathtt{M},\ \pi \overset{\mathtt{c}}{\models} \mathtt{f\ abort\ b}) = \\
\quad \forall \mathtt{i} \in \mathtt{pl}\,\pi.\ \mathtt{FirstRise\ M}\ \pi\ \mathtt{c\ i} \\
\qquad\qquad\quad \Rightarrow \\
\qquad\qquad\quad ((\mathtt{M},\ \pi^{\mathtt{i}} \overset{\mathtt{c}}{\models} \mathtt{f}) \\
\qquad\qquad\quad\ \vee \\
\qquad\qquad\quad\ \exists \mathtt{j} \in \mathtt{pl}\,\pi.\ \mathtt{i} < \mathtt{j} \wedge \\
\qquad\qquad\quad\ \ \exists \pi\text{'} \in \mathtt{sim}\ \pi.\ (\mathtt{M},\ \pi^{\mathtt{j}} \overset{\mathrm{T}}{\models} \mathtt{c} \wedge \mathtt{b}) \wedge (\mathtt{M},\ \pi^{(\mathtt{i},\mathtt{j}-1)}\pi' \overset{\mathtt{c}}{\models} \mathtt{f}))) \\
\wedge \\
((\mathtt{M},\ \pi \overset{\mathtt{c}}{\models} \mathtt{f@c1}) = \\
\quad (\mathtt{M},\ \pi \overset{\mathtt{c1}}{\models} \mathtt{f})) \\
\wedge \\
((\mathtt{M},\ \pi \overset{\mathtt{c}}{\models} \mathtt{f@c1!}) = \\
\quad (\mathtt{M},\ \pi \overset{\mathtt{c1!}}{\models} \mathtt{f}))
\end{array}
$$

7.4. Optional Branching Extension

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 $(\mathtt{M},\ \mathtt{s} \models \mathtt{f})$ for `O_SEM M s f`, and then the semantics of the OBE is defined by:

$$
\begin{array}{l}
((\mathtt{M},\ \mathtt{s} \models \mathtt{b}) = (\mathtt{M},\ \mathtt{L}_{\mathtt{M}}(\mathtt{s}) \models \mathtt{b})) \\
\wedge \\
((\mathtt{M},\ \mathtt{s} \models \neg\mathtt{f}) = \neg((\mathtt{M},\ \mathtt{s} \models \mathtt{f}))) \\
\wedge \\
((\mathtt{M},\ \mathtt{s} \models \mathtt{f1} \wedge \mathtt{f2}) = (\mathtt{M},\ \mathtt{s} \models \mathtt{f1}) \wedge (\mathtt{M},\ \mathtt{s} \models \mathtt{f2})) \\
\wedge \\
((\mathtt{M},\ \mathtt{s} \models \mathtt{EX\ f}) = \\
\quad \exists \pi.\ \mathtt{Path\ M}\ \pi \wedge \mathtt{pl}\ \pi\ \mathtt{1} \wedge (\pi_0 = \mathtt{s}) \wedge (\mathtt{M},\ \pi_1 \models \mathtt{f}))
\end{array}
$$

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