Applying Theorem Proving to PSL Formal Semantics

- Assertions capture design intent for documentation and verification

  "assertion-based methodologies offer the same leap in productivity for verification that logic synthesis did for design entry"

  `http://www.eetimes.com/story/OEG20021112S0031`

- Assertions are written in a property language. Competing languages:
  - Property Specification Language (PSL)
    - originating from IBM, donated to Accellera
  - SystemVerilog Assertions (SVA)
    - originating from Intel & Synopsys, donated to Accellera
  - *e* properties
    - the property subset of Verisity's proprietary verification language

Property languages come with formal semantics

- Frequently used acronyms
  (`http://www.deepchip.com/items/0423-14.html`)

```
PSL:  Property Specification Language
OVL:  Open Verification Library (Verilog modules)
OVA:  Open Vera Language
SVA:  System Verilog Assertions
SVL:  System Verilog assertion Library (SVA version of OVL)
```

- Property languages are quite simple and come with formal semantics
  - compare situation with programming languages and HDLs

- PSL had formal semantics from the start, SVA semantics being drafted

From IBM's Sugar to Accellera's PSL

- Sugar 1.0 was CTL plus lots of syntactic sugar (hence name)
- Sugar 2.0 added LTL to Sugar 1.0
  - won Accellera competition to select standard property language
  - development handed over to Accellera committee
- Accellera also standardising SystemVerilog verification language
  - derived from Synopsys Vera language
  - with properties based on Intel's ForSpec property language
- Accellera wants to 'align' SystemVerilog assertions with PSL
  - set up a committee and produced a report

Accellera FVTC Alignment Sub-Committee Final Report

Surrendra Dudani[1] Cindy Eisner[2] Dana Fisman[2,3]
Harry Foster[4] John Havlicek[5] Joseph Lu[6] Erich Marschner [7]
Johan Mårtensson[8] Ambar Sarkar[9] Bassam Tabbara[10]

[1] Synopsys [2] IBM [3] Weizmann Institute of Science
[4] Jasper Design Automation [5] Motorola [6] Sun [7] Cadence
[8] Safelogic [9] Paradigm Works [10] Novas Software

February 11, 2004

1 Introduction

On May 29, 2003, Accellera announced the official approval of Accellera PSL (Property Specification Language) 1.01, developed by the Accellera Formal Verification Technical Committee (FVTC), and of SystemVerilog 3.1, developed by the Accellera SystemVerilog Technical Committee. SystemVerilog contains an assertion capability known as SVA, developed by the SystemVerilog Assertions Committee (SV-AC).

During the development of these two standards, the pending co-existence of two different Accellera-approved assertion languages raised concern about the potential for syntactic and semantic conflict between them, which might cause confusion among users. This led to interest in aligning the two languages, to create a "unified kernel" assertion capability that would be common to both. The word 'kernel' in this phrase reflects the recognition that PSL and SVA each contained, and would continue to contain, their own specific features that were not and would not be present in the other language, but that where the two languages overlapped, they should be aligned.

While some effort was made during the development of SVA to align with PSL (for instance, the Semantics Sub-committee of the SV-AC included the developers of the PSL formal semantics), only partial alignment was achieved, in part because the PSL definition was in its final stages of documentation by the time SVA was developed. As PSL 1.01 and System Verilog 3.1 were going through final review in early 2003, the "unified kernel", or common core of syntax and semantics shared by PSL and SVA, was still spotty and poorly defined. In recognition of this, the FVTC proposed a roadmap for further alignment of the two languages. This roadmap was presented to, and approved by, the Accellera Board of Directors in February 2003.

In May 2003, the Alignment Sub-committee of the FVTC was formed to continue alignment of the two languages, following the roadmap approved by the Accellera Board. The Alignment Sub-committee met weekly from May to October 2003, and sporadically from October 2003 to January 2004. Despite

Academic versus industrial strength property languages

- PSL and SVA designed for both static and dynamic checking
  - dynamic checking by simulation monitors (finite traces)
  - static checking by formal verification (infinite traces)
- Industrially motivated language constructs, e.g. in PSL:
  - Patterns
    - $r \mapsto f$ formula f holds whenever pattern (SERE) r matched
  - Multiple clocking
    - f@c formula f holds whenever clock c ticks
  - Abort construct
    - f abort c abort checking formula f if b becomes true
- Makes semantics several times more complex than textbook semantics
  - but several times small is still smallish – i.e. still tractable

Embedding PSL/Sugar in higher order logic (HOL)

- Whilst teaching a course I read an article of IBM's Sugar language
  - added industrial reality to academic CTL and LTL
  - Sugar team at IBM very helpful
  - I teach CTL/LTL as domain specific subsets of HOL
  - model checking is a decision procedure for a subset of HOL (c.f. PVS)

- After my course finished I did an embedding in HOL system logic
  - straightforward and standard exercise
  - initially just fun and sanity checking my understanding
  - applications came later (still looking)

How PSL/Sugar is embedded in higher order logic (HOL)

- Deep semantic embedding
  - define data-types in HOL to represent expressions, formulas etc
  - define data-types to represent paths (finite and infinite)
  - define constant "$\models$" to represent semantics: $\langle path \rangle \models \langle formula \rangle$
- Higher order logic is very natural for representing the semantics
  - only alternative is formal set theory (but limited tool support)
  - HOL system ASCII notation a bit verbose and ad hoc
- Would be nice if official semantics were machine readable
  - currently LaTeX plus English
  - AI+NLP would be needed to mechanically decode
  - encoding in HOL revealed minor 'looseness' and 'semantic typos'

Standard logical notation and equivalent HOL notation

Standard notation	HOL notation	Description
$true$	`T`	truth
$false$	`F`	falsity
$\neg t$	`~`t	negation
$t_1 \wedge t_2$	t_1 `/\` t_2	conjunction
$t_1 \vee t_2$	t_1 `\/` t_2	disjunction
$t_1 \Rightarrow t_2$	t_1 `==>` t_2	implication
$\forall x.P(x)$	`!`x`.`$P(x)$	universal quantification
$\exists x.P(x)$	`?`x`.`$P(x)$	existential quantification
$p \in s$	p `IN` s	set membership
$[0..n)$	`LESS` n	set of natural numbers less than n
$\forall x \in s.\ P(x)$	`!`x`::s.` $P(x)$	universal quantification restricted to s
$\exists x \in s.\ P(x)$	`?`x`::s.` $P(x)$	existential quantification restricted to s
$\forall x \in [0..n).\ P(x)$	`!`x`::LESS` n`.` $P(x)$	universal quantification restricted to numbers less than n
$\exists x \in [0..n).\ P(x)$	`?`x`::LESS` n`.` $P(x)$	existential quantification restricted to numbers less than n
ϵ	`[]`	empty list
x	`[`x`]`	list with one element (singleton)
$l_1 l_2$	l_1 `<>` l_2	list concatenation (append)
$ty_1 \times ty_2$	ty_1 `#` ty_2	Cartesian product of types ty_1 and ty_2
$ty_1 \longrightarrow ty_2$	ty_1 `-->` ty_2	type of functions from ty_1 to ty_2

PSL semantic notation (B2.1 of the Reference Manual)

We use i, j, and k to denote non-negative integers. We denote the i^{th} letter of v by v^{i-1} (since counting of letters starts at zero). We denote by $v^{i..}$ the suffix of v starting at v^i. That is, for every $i <| v |$, $v^{i..} = v^i v^{i+1} \cdots v^n$ or $v^{i..} = v^i v^{i+1} \cdots$. We denote by $v^{i..j}$ the finite sequence of letters starting from v^i and ending in v^j. That is, for $j \geq i$, $v^{i..j} = v^i v^{i+1} \cdots v^j$ and for $j < i$, $v^{i..j} = \epsilon$. We use ℓ^ω to denote an infinite-length word, each letter of which is ℓ.
We use $\overline{v}$ to denote the word obtained by replacing every $\top$ with a $\bot$ and vice versa. We call $\overline{v}$ the *complement* of v.

PSL Notation	HOL representation	Description
∞	`INFINITY`	infinity
ϵ	`[]`	empty path
$\top^\omega$	`TOP_OMEGA`	infinite repetition of $\top$
$\|v\|$	`LENGTH` v	length of a path
v^i	`ELEM` v i	i^{th} letter of v
$v^{i..}$	`RESTN` v i	suffix of v starting at v^i
$v^{i..j}$	`SEL` v (i, j)	sequence starting at v^i and ending at v^j
$v_1 v_2$	`CAT`(v_1, v_2)	concatenation of finite or infinite sequences v_1 and v_2
$\overline{v}$	`COMPLEMENT` v	complement of v (swap $\top$s and $\bot$s)

LRM semantics, pretty-printed HOL and raw HOL

- Example from Accellera Language Reference Manual (B.2.1.1.2)

> $v \models \varphi$ abort $b \iff$ either $v \models \varphi$ or $\exists j < |v|$ s.t. $v^j \Vdash b$ and $v^{0..j-1}\top^\omega \models \varphi$

- Pretty-printed translation into higher order logic

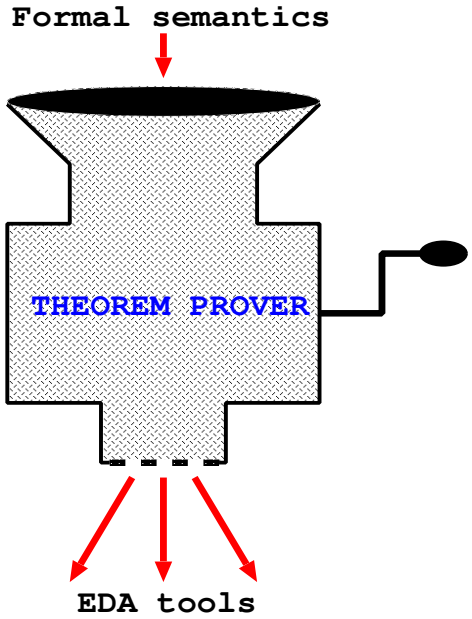
$$v \models f \text{ abort } b \;=\; v \models f \vee \exists j \in [0..|v|).\; v^j \Vdash b \wedge v^{0..j-1}\top^\omega \models f$$

- Raw HOL representation (i.e. not pretty-printed)

```
UF_SEM v (F_ABORT (f,b)) =
 UF_SEM v f
 \/
 ?j :: LESS(LENGTH v).
  B_SEM (ELEM v j) b /\ UF_SEM (CAT(SEL v (0,j-1),TOP_OMEGA)) f
```

Why encode PSL semantics in machine readable logic?

- Analyse the semantics (FAC paper)
  - reveal vagueness, ambiguity, 'semantic typos' etc
  - perform shallow 'sanity checking' proofs
  - verify complex properties
- Implement semantics directed tools (CHARME paper with Hurd & Slind)



- Tool1: evaluate properties on finite paths
- Tool2: compile properties to simulation checkers
- Tool3: semantics driven fully expansive model checker

Issues arising from analysing PSL/Sugar semantics

- Transcription errors when manually translating LaTeX+English to HOL
  - need a 'golden' machine readable format for proofs (XML?)
- Yields interesting data for improving proof support
  - need better tools to combine arithmetical and logical reasoning
- Hard to reuse proof scripts as semantics evolves
  - redid proofs for Sugar 2.0, PSL 1.01 and PSL 1.1

Issues arising from building semantics directed tools

- Fast execution vital
  - Barras `EVAL` tool in HOL good for proof of concept
  - Intel's $reFL^{ect}$ better for actual implementation?
- Research needed to develop efficient executable specifications
  - example: Hurd & Slind execution of regular expression semantics
- Can use theorems to make correct-by-construction optimisations
  - clock elimination rewriting
    $$\vdash \forall r\ v\ c.\ v \models\!\!\!\models^{c} r = v \models \mathcal{R}^{c}(r)\ , \qquad \vdash \forall f\ v\ c.\ v \models\!\!\!\models^{c} f = v \models \mathcal{F}^{c}(f)$$
    (recently completed for 1.1 semantics – messy)
  - cycle-based evaluation (projection, temporal abstraction)
    $$\vdash \forall r\ v\ c.\ v \models\!\!\!\models^{c} r = v|_{c} \models\!\!\!\models r\ , \qquad \vdash \forall f\ v\ c.\ v \models\!\!\!\models^{c} f = v|_{c} \models f$$
    (in progress – complicated for 1.1 semantics – side conditions needed)

Future research

- More analysis of PSL 1.1 semantics
  - especially projection view
- Embed SVA semantics in HOL
  - develop executable translators
- Embed Fisler's Timing Diagram Language in HOL
  - verify translation to PSL and SVA
- Goal: applications for which execution inside a theorem prover is feasible
  - training and property validation tools
  - checker and translator generators

Future research

- More analysis of PSL 1.1 semantics
  - especially projection view
- Embed SVA semantics in HOL
  - develop executable translators
- Embed Fisler's Timing Diagram Language in HOL
  - verify translation to PSL and SVA
- Goal: applications for which execution inside a theorem prover is feasible
  - training and property validation tools
  - checker and translator generators

THE END

Example

- PSL 1.01 Reference Manual Example 2, page 45

```
time   0   1   2   3   4   5   6   7   8   9
-------------------------------------------
clk1   0   1   0   1   0   1   0   1   0   1
a      0   0   0   1   1   1   0   0   0   0
b      0   0   0   0   0   1   0   1   1   0
c      1   0   0   0   0   1   1   0   0   0
clk2   1   0   0   1   0   0   1   0   0   1
```

- Define w to be this path, so w is:
  `{c,clk2}{clk1}{}{clk1,a,clk2}{a}{clk1,a,b,c}{c,clk2}{clk1,b}{b}{clk1,clk2}`
- Can evaluate in SML, or via a command line wrapper
- Example: to evaluate `(c && next!(a until b))@clk1` at all times in w:

```
% pslcheck -all \
    -fl   '(c && next!(a until b))@clk1' \
    -path '{c,clk2}{clk1}{}{clk1,a,clk2}{a}{clk1,a,b,c}{c,clk2}{clk1,b}{b}{clk1,clk2}'
> > true at times 4,5,10
```

Example from http://www.eda.org/vfv/hm/1017.html

```
time   0   1   2   3   4   5   6   7
-------------------------------------
clk1   0   1   0   1   0   1   0   1
a      0   1   1   0   0   0   0   0
b      0   0   0   1   0   0   0   0
c      0   0   0   0   1   0   1   0
clk2   1   0   0   1   0   0   1   0
```

Someone asks Formal Property Language Technical Committee mailing list

```
| p34 line 18 :
|  The multiply-clocked SERE {{a;b}@clk1;c}@clk2 holds tightly
|  from time 0 to time 6. It does not hold tightly
|  over any other interval of the given behaviour.
|
| It seems however to me that it also holds tightly from time
| 1 to time 6, or there is an inconsistency with what is said p33 line 52 :
```

HOL tool can mechanically calculate all intervals where SERE holds tightly:

```
% pslcheck -all \
    -sere '{{a;b}@clk1;c}@clk2' \
    -path '{clk2}{clk1,a}{a}{clk1,b,clk2}{c}{clk1}{c,clk2}{clk1}'
> > true on intervals [0:6]
```