Assertion-Based Verification (ABV)

- It has been claimed that assertion based verification:
  
  "is likely to be the next revolution in hardware design verification"

- Basic idea:
  - document designs with formal properties
  - use simulation (dynamic) and model checking (static)

- Problem: too many languages
  - academic logics: LTL, CTL
  - tool-specific industrial versions: Intel, Cadence, Motorola, IBM, Synopsys

- What to do? Solution: a competition!
  - run by Accellera organisation
  - results standardised by IEEE
  - lots of politics
IBM’s *Sugar* and Accellera’s PSL

- *Sugar 1*: property language of IBM RuleBase checker
  - CTL plus *Sugar* Extended Regular Expressions (SEREs)

- Competition finalists: IBM’s *Sugar 2* and Motorola’s *CBV*
  - Intel/Synopsys ForSpec eliminated earlier (apparently industry politics involved)

- *Sugar 2* is based on LTL rather than CTL
  - has CTL constructs: “Optional Branching Extension” (OBE)
  - has clocking constructs for temporal abstraction

- Accellera purged “Sugar” from it property language
  - the word “Sugar” was too associated with IBM
  - language renamed to PSL
  - SEREs now *Sequential Extended Regular Expressions*

- Lobbying to make PSL more like ForSpec (align with SVA)
SEREs: Sequential Extended Regular Expressions

- SEREs are from the industrial PSL (more on PSL later)

- Syntax:

  \[ r ::= p \quad \text{(Atomic formula } p \in AP) \]
  \[ !p \quad \text{(Negated atomic formula } p \in AP) \]
  \[ r_1 \mid r_2 \quad \text{(Disjunction)} \]
  \[ r_1 \& \& r_2 \quad \text{(Conjunction)} \]
  \[ r_1 ; r_2 \quad \text{(Concatenation)} \]
  \[ r_1 : r_2 \quad \text{(Fusion)} \]
  \[ r[\ast] \quad \text{(Repeat)} \]

- Semantics:

  (\( w \) ranges over finite lists of states \( s \); \(|w| \) is length of \( w \);
  \( w_1.w_2 \) is concatenation; \textbf{head} \( w \) is head; \( \langle \rangle \) is empty word)

  \[ \llbracket p \rrbracket(w) = p \in L(\text{head } w) \land |w| = 1 \]
  \[ \llbracket !p \rrbracket(w) = \neg(p \in L(\text{head } w)) \land |w| = 1 \]
  \[ \llbracket r_1 \mid r_2 \rrbracket(w) = \llbracket r_1 \rrbracket(w) \lor \llbracket r_2 \rrbracket(w) \]
  \[ \llbracket r_1 \& \& r_2 \rrbracket(w) = \llbracket r_1 \rrbracket(w) \land \llbracket r_2 \rrbracket(w) \]
  \[ \llbracket r_1 ; r_2 \rrbracket(w) = \exists w_1 \ w_2. \ w = w_1.w_2 \land \llbracket r_1 \rrbracket(w_1) \land \llbracket r_2 \rrbracket(w_2) \]
  \[ \llbracket r_1 : r_2 \rrbracket(w) = \exists w_1 \ s \ w_2. \ w = w_1.s.w_2 \land \llbracket r_1 \rrbracket(w_1.s) \land \llbracket r_2 \rrbracket(s.w_2) \]
  \[ \llbracket r[\ast] \rrbracket(w) = w = \langle \rangle \lor \exists w_1 \ldots w_i. \ w = w_1.\ldots.w_i \land \llbracket r \rrbracket(w_1) \land \ldots \land \llbracket r \rrbracket(w_i) \]
Example SERE

Example

A sequence in which \textit{req} is asserted, followed four cycles later by an assertion of \textit{grant}, followed by a cycle in which \textit{abortin} is not asserted.

Define $p[*3] = p; p; p$

Then the example above can be represented by the SERE:

\[ \text{req; } T[*3]; \text{grant; } !\text{abortin} \]

In PSL this could be written as:

\[ \text{req; } [*3]; \text{grant; } !\text{abortin} \]

where $[*3]$ abbreviates $T[*3]$

more ‘syntactic sugar’ later

e.g. \textit{true}, \textit{false} for $T$, $F$
PSL Foundation Language (FL is LTL + SEREs)

- **Syntax:**
  
  $$f ::= p$$  
  (Atomic formula - \(p \in AP\))
  
  $$!f$$  
  (Negation)
  
  $$f_1 \mathbin{\lor} f_2$$  
  (Disjunction)
  
  $$\text{next } f$$  
  (Successor)
  
  $$\{r\}(f)$$  
  (Suffix implication: \(r\) a SERE)
  
  $$\{r_1\} \rightarrow \{r_2\}$$  
  (Suffix next implication: \(r_1, r_2\) SEREs)
  
  $$[f_1 \mathbin{\text{until}} f_2]$$  
  (Until)

- **Semantics (omits clocking, weak/strong distinction):**

  $$[p]_M(\pi) = p \in L(\pi(0))$$

  $$[!f]_M(\pi) = \neg([f]_M(\pi))$$

  $$[f_1 \mathbin{\lor} f_2]_M(\pi) = [f_1]_M(\pi) \lor [f_2]_M(\pi)$$

  $$[\text{next } f]_M(\pi) = [f]_M(\pi \downarrow 1)$$

  $$\{}r\}(f)]_M(\pi) = \forall \pi' w. (\pi = w.\pi' \land [r]_M(w)) \Rightarrow [f]_M(\pi')$$

  $$\{}r_1\} \rightarrow \{r_2\}_M(\pi) = \forall \pi' w_1 s. (\pi = w_1.s.\pi' \land [r_1]_M(w_1.s))$$

  $$\Rightarrow \exists \pi'' w_2. \pi' = w_2.\pi'' \land [r_2]_M(s.w_2)$$

  $$[[f_1 \mathbin{\text{until}} f_2]]_M(\pi) = \exists i. [f_2]_M(\pi \downarrow i) \land \forall j. j < i \Rightarrow [f_1]_M(\pi \downarrow j)$$

- **There is also an Optional Branching Extension (OBE):**

  - completely standard CTL: \(\text{EX}, E[\rightarrow \rightarrow \mathbf{U} \rightarrow \rightarrow]\), \(\text{EG}\) etc.
Combining SEREs with LTL formulae

- Formula \( \{r\}f \) means LTL formula \( f \) true after SERE \( r \)
- Example

  After a sequence in which \( \text{req} \) is asserted, followed four cycles later by an assertion of \( \text{grant} \), followed by a cycle in which \( \text{abortin} \) is not asserted, we expect to see an assertion of \( \text{ack} \) some time in the future.

  Can represent by

  \[
  \text{always } \{\text{req; [\ast 3]; grant; !abortin}\}(\text{eventually } \text{ack})
  \]

  where \text{eventually} and \text{always} are defined by:

  \[
  \text{eventually } f = \left[\text{true until } f\right] \\
  \text{always } f = !\left(\text{eventually } !f\right)
  \]

- N.B. Ignoring strong/weak distinction
  - strong/weak distinction important for dynamic checking
  - semantics when simulator halts before expected event
  - strictly should write \text{until}!, \text{eventually}!
How can we modify
\[
\text{always \ reqin;ackout;!abortin \ } \rightarrow \ \text{ackin;ackin}
\]
so that the two cycles of \text{ackin} start the cycle after \text{!abortin}?

Two ways of doing this
\[
\begin{align*}
\text{always}\{\text{reqin;ackout;!abortin}\} & \rightarrow \{\text{true;ackin;ackin}\} \\
\text{always}\{\text{reqin;ackout;!abortin}\} & \Rightarrow \{\text{ackin;ackin}\}
\end{align*}
\]

\(\Rightarrow\) is a defined operator
\[
\{r1\}\Rightarrow\{r2\} = \{r1\} \rightarrow \{\text{true};r2\}
\]

Note: \text{true} and \text{T} are synonyms.
Examples of defined notations: consecutive repetition

- Define
  
  \[ r[+] = r; r[*] \]
  
  \[ \text{false}[*] \text{ if } i=0 \]
  
  \[ r[*i] = | \]
  
  \[ | r; ...; r \text{ otherwise (i repetitions)} | \]
  
  \[ r[*i..j] = r[*i] | r[*i+1] | ... | r[*j] \]
  
  \[ [+] = \text{true}[+] \]
  
  \[ [*] = \text{true}[*] \]

- Example

  Whenever we have a sequence of \text{req} followed by \text{ack}, we should see a full transaction starting the following cycle. A full transaction starts with an assertion of the signal \text{start_trans}, followed by one to eight consecutive data transfers, followed by the assertion of signal \text{end_trans}. A data transfer is indicated by the assertion of signal \text{data}

  \[
  \text{always}\{\text{req}; \text{ack}\} |=>\{\text{start_trans}; \text{data}[*1..8]; \text{end_trans}\}
  \]
Fixed number of non-consecutive repetitions

Example

Whenever we have a sequence of \texttt{req} followed by \texttt{ack}, we should see a full transaction starting the following cycle. A full transaction starts with an assertion of the signal \texttt{start\_trans}, followed by eight not necessarily consecutive data transfers, followed by the assertion of signal \texttt{end\_trans}. A data transfer is indicated by the assertion of signal \texttt{data}

Can represent by

\[
\text{always}
\{\texttt{req;ack} \mid\Rightarrow
\{\texttt{start\_trans;}
\{\{!\texttt{data[\ast]\};\texttt{data}}[\ast\textbf{8}];!\texttt{data[\ast]}\}\};
\texttt{end\_trans}\}
\]

Define: \( b[= i] = \{!b[\ast];b\}[\ast i];!b[\ast] \)

Then have a nicer representation

\[
\text{always}\{\texttt{req;ack}\mid\Rightarrow\{\texttt{start\_trans;data[= 8];end\_trans}\}
\]
Variable number of non-consecutive repetitions

- Example

Whenever we have a sequence of \texttt{req} followed by \texttt{ack}, we should see a full transaction starting the following cycle. A full transaction starts with an assertion of the signal \texttt{start\_trans}, followed by one to eight not necessarily consecutive data transfers, followed by the assertion of signal \texttt{end\_trans}. A data transfer is indicated by the assertion of signal \texttt{data}.

- Define

\[
\begin{align*}
b[i..j] &= \{b[i]\} \mid \{b[(i+1)]\} \mid \ldots \mid \{b[j]\}
\end{align*}
\]

- Then

\[
\begin{align*}
\text{always } \{\texttt{req;ack}\} & \implies \\
& \{\texttt{start\_trans;data[= 1..8];end\_trans}\}
\end{align*}
\]

- These examples are meant to illustrate how PSL/Sugar is much more readable than raw CTL or LTL.
Clocking

- Basic idea: \(\texttt{b@clk}\) samples \(b\) on rising edges of \(\texttt{clk}\)

- Can clock SEREs (\(\texttt{r@clk}\)) and formulae (\(\texttt{f@clk}\))

- Can have several clocks

- Official semantics messy due to clocking

- Can ‘translate away’ clocks by pushing \(\texttt{@clk}\) inwards
  - rules given in PSL manual
  - roughly: \(\texttt{b@clk} \leftrightarrow \{!\texttt{clk}[*]\};\texttt{clk} \& b\)
Model checking PSL (outline)

- SEREs checked by generating a finite automaton
  - recognise regular expressions
  - these automata are called “satellites”

- FL checked using standard LTL methods

- OBE checked by standard CTL methods

- Can also check formula for runs of a simulator
  - this is **dynamic verification**
  - semantics handles possibility of finite paths – messy!

- Commercial checkers only handle a subset of PSL
PSL layer structure

- **Boolean layer** has atomic predicates
- **Temporal layer** has LTL (FL) and CTL (OBE) properties
- **Verification layer** has commands for how to use properties
  - e.g. `assert`, `assume`

```
assert always (!en1 & en2))
```

- **Modelling layer**: HDL specification of e.g. inputs, checkers
  - e.g. `augment always(Req -> eventually! Ack)`
  - `add counter to keep track of numbers of Req and Ack`
PSL/Sugar summary

- Combines together LTL and CTL
- Regular expressions – SEREs
- LTL – Foundation Language formulae
- CTL – Optional Branching Extension
- Relatively simple set of primitives + definitional extension
- Boolean, temporal, verification, modelling layers
- Semantics for static and dynamic verification (needs strong/weak distinction)