Generating Verilog Checkers from PSL Formulas

Joe Hurd
joe.hurd@cl.cam.ac.uk

University of Cambridge
Contents

- Introduction
- Simple Formulas
- Verified Checkers
- An Example Formula
- Conclusion
Introduction: What is PSL?

- “PSL is an intuitive, declarative language for describing behaviour over time.”
- This talk: the Temporal Layer of PSL, essentially LTL with regular expressions:
  - Boolean Expressions
    - Evaluated on a single state.
  - Sequential Extended Regular Expressions (SEREs)
    - Evaluated on a finite sequence of states.
  - Foundation Language Formulas
    - Evaluated on a finite or infinite path of states.
    - This talk: will only consider infinite paths.
Introduction: Verilog Checkers

- Suppose we have a circuit written as a Verilog program,
- and a PSL formula that we would like to hold of every simulation run of the circuit.
  - Think of a simulation run as an infinite path of states.
- We can code up the formula as a Verilog module that monitors the circuit.
  - But how to avoid bugs?
- Using HOL4, we can verify a translation from the PSL formula to a deterministic finite automaton.
  - The DFA is guaranteed to produce an error iff the PSL formula is violated on the simulation path.
  - Thanks to Mike Gordon’s formalization of PSL.
Contents

- Introduction
- Simple Formulas
- Verified Checkers
- An Example Formula
- Conclusion
Safety Violations

- Given a checking automaton for the PSL formula \( f \),
- and an infinite path \( p \),
- when can the automaton report a property violation?

\[
safety\_violation \ p \ f \equiv \exists n. \ \forall q. \ |q| = \infty \ \Rightarrow \ \neg (p^{0,n} q \models f)
\]

\[
\underbrace{p_0 \ \ p_1 \ \ \cdots \ \ p_n}_{\text{bad prefix}} \ \bullet \ \bullet \ \bullet \ \bullet \ \bullet \ \bullet \ \bullet \ \cdots \ \models \ \neg f
\]

- If the bad prefixes form a regular language, then we can detect safety violations with a finite state automaton.
Overall Goal

- This is the overall specification of a checker automaton:

\[
\forall f, p. \\
|p| = \infty \land \text{simple } f \Rightarrow \\
(\text{safety\_violation } p, f) \iff \\
\exists n. \text{amatch } (\text{sere2regexp } (\text{checker } f)) p^{0,n}
\]

- Observe that \text{checker} maps a PSL formula to a SERE.
- Not enough to have an implication, because otherwise a trivial checker $\top$ or $\perp$ would suffice.
- Condition 1: $p$ is an infinite path.
- Condition 2: $f$ is a simple formula.
Strong Operators

- Strong operators can construct liveness properties.
  - Liveness says that a property will eventually happen.
  - A violation is an infinite path where the property never occurs.
- Strong operators can induce subtle safety violations.
- For example, the formulas

\[ \{ \top \} \leftrightarrow \{ \{ P[*] \} : \neg P \}! \]

\[(\text{next } P) \text{ until! } (\neg P)\]

are both safety violations on the path

\[P \ P \ P \ P \ \cdots\]
Consider the formula

\[ \{\top\} \leftrightarrow \{\{P[*]\}\}; \{\neg P \land Q\}\}! \land \{\top\} \leftrightarrow \{\{P[*]\}\}; \{\neg P \land \neg Q\}\}! \]

It’s “pathologically safe” [Kuperferman & Vardi 1999], meaning that there is a path

\[ P \ P \ P \ P \ \cdots \]

with a bad prefix \( \lbrack \) for the property, but there are no bad prefixes for either of the conjuncts.

**Solution:** exclude all strong operators from our simple class.

- **Surprise:** Accellera permit strong suffix implication \( \{\cdot\} \leftrightarrow \{\cdot\}! \) in their simple class of formulas!
Contents

- Introduction
- Simple Formulas
- **Verified Checkers**
- An Example Formula
- Conclusion
Boolean Checkers

- Boolean formulas talk about a single state.
- All boolean formulas are simple:

\[
\vdash \forall f. \text{boolean } f \Rightarrow \text{simple } f
\]

- Define a boolean_checker for boolean formulas:

\[
\vdash \forall f, p. \\
|p| = \infty \land \text{boolean } f \Rightarrow \\
(\text{safety\_violation } p \ f \iff \\
\exists n. \text{amatch (sere2regexp (boolean\_checker } f))) \ p^{0,n}
\]

- Use boolean checkers for boolean formulas:

\[
\vdash \forall f. \text{boolean } f \Rightarrow (\text{checker } f = \text{boolean\_checker } f)
\]
Temporal Checkers: Next

- The next operator ‘postpones’ a formula by one step:

  \[ \vdash w \models \text{next } f \iff |w| > 0 \land w^1 \models f \]

- Next formulas are simple:

  \[ \vdash \forall f. \text{simple } f \Rightarrow \text{simple } (\text{next } f) \]

- Next checkers just prepend the SERE \{\top\}:

  \[ \vdash \text{checker } (\text{next } f) = \{\top\}; \{\text{checker } f\} \]
Temporal Checkers: Until

- The weak until operator is defined thus:

\[ \vdash w \models f \text{ until } g \iff \forall j \in [0..|w|). \, w^j \models f \Rightarrow \exists k \in [0..j + 1). \, w^k \models g \]

- The condition for weak until formulas to be simple:

\[ \vdash \forall f, g. \, \text{simple } f \land \text{boolean } g \Rightarrow \text{simple } (f \text{ until } g) \]

- Weak until checkers are defined as

\[ \vdash \text{checker } (f \text{ until } g) \equiv \{(\text{boolean_checker } g)[*]\}; \{(\text{checker } f) \sqcap \{\text{boolean_checker } g\}\} \]
**Temporal Checkers: Or**

- The $\lor$ temporal operator is defined in the obvious way:

  $$ \vdash w \models f \lor g \iff w \models f \lor w \models g $$

- Our condition for $\lor$ formulas to be simple:

  $$ \vdash \forall f, g. \text{simple } f \land \text{simple } g \Rightarrow \text{simple } (f \lor g) $$

  - Accellera: $\forall f, g. \text{boolean } f \land \text{simple } g \Rightarrow \text{simple } (f \lor g)$
  - We’re more general for both $\lor$ and $\land$.

- $\lor$ checkers are defined as

  $$ \vdash \text{checker } (f \lor g) \equiv \{ \text{checker } f \} \cap \{ \text{checker } g \} $$
Verification

- Most important was the following lemma:

\[ \vdash \forall f, p. \]
\[ \text{simple } f \land |p| = \infty \Rightarrow \]
\[ (p \models \neg f \iff \text{safety\_violation } p f) \]

- For simple formulas, violations are the same as safety violations.

- Necessary to verify until, useful for the other operators.

\[ \vdash \forall f, p. \]
\[ |p| = \infty \land \text{simple } f \Rightarrow \]
\[ (\text{safety\_violation } p f \iff \exists n. \text{amatch } (\text{sere2regexp } (\text{checker } f)) p^{0,n}) \]
Creating Verilog Checkers

- Take the SERE version of the checker, and lazily convert to a nondeterministic finite automaton (NFA).
- Compute the reachable states of the deterministic finite automaton (DFA) via transition theorems:

\[
\vdash \forall s. \\
\text{StoB\_REQ} \notin s \land \text{BtoS\_ACK} \in s \Rightarrow \\
\text{eval\_transitions} R [6] s = [2; 4]
\]

- Finally, print the whole DFA as a Verilog module.
  - An informal step, could introduce bugs :-(
Contents

- Introduction
- Simple Formulas
- Verified Checkers
- **An Example Formula**
- Conclusion
Example: PSL Formula

From page 45 of the Accellera PSL Reference Manual:

\[ c \land \text{next (a until b)} \]

Their example actually uses strong until, we’ll use weak until instead.
Example: SERE

|- checker (...example PSL formula...) =
S_OR
 (S_BOOL (B_NOT (B_PROP c)),
 S_CAT
 (S_BOOL B_TRUE,
  S_CAT
 (S_REPEAT (S_BOOL (B_NOT (B_PROP b))),
   S_OR
    (S_AND
     (S_BOOL (B_NOT (B_PROP a))),
      S_CAT
     (S_BOOL (B_NOT (B_PROP b))),
      S_REPEAT (S_BOOL B_TRUE)));
 S_AND
  (S_CAT
   (S_BOOL (B_NOT (B_PROP a))),
    S_REPEAT (S_BOOL B_TRUE)),
 S_BOOL (B_NOT (B_PROP b)))))))
Example: Verilog Module

```verilog
module Checker (a, b, c);
input a, b, c;
reg [2:0] state;
initial state = 0;
always @(a or b or c)
begin
    case (state)
        0: if (c) state = 5; else state = 1;
        1: begin $display ("Checker: property violated!"); $finish; end
        2: begin $display ("Checker: property violated!"); $finish; end
        3: state = 3;
        4: if (a) if (b) state = 3; else state = 4;
            else if (b) state = 3; else state = 2;
        5: if (a) if (b) state = 3; else state = 4;
            else if (b) state = 3; else state = 2;
        default: begin $display ("Checker: unknown state"); $finish; end
    endcase
end
endmodule
```
Contents

- Introduction
- Simple Formulas
- Verified Checkers
- An Example Formula
- Conclusion
Conclusion

- An interesting exercise that covers a wide range of formulas while staying within PSL.
- Strong operators require more advanced technology.
- Possible practical applications of the Verilog checkers?
  - Will almost certainly require state minimization to be practical. To do!
- **Future Work:** To extend our coverage, must drop SEREs as intermediate language.
  - Would like to implement weak suffix implication $\{ \cdot \} \rightarrow \{ \cdot \}$ which is in the Accellera simple subset.