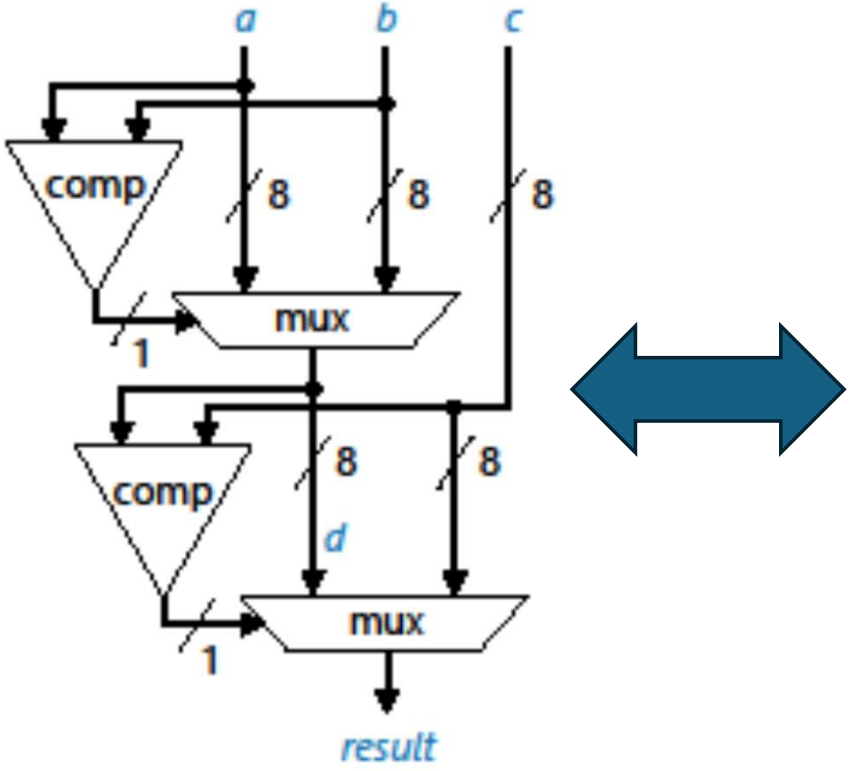
fmcad.25

Unlocking Hardware Verification with Oracle Guided Synthesis

Leiqi Ye, Yixuan Li, Guy Frankel, Jianyi Cheng, Elizabeth Polgreen

THE UNIVERSITY *of* EDINBURGH

Motivating Example – Module of max3



Example: Circuit computing max of three inputs.

```
// Return the max value of three inputs.
module max3(
  input  [7:0] a, input  [7:0] b,
  input  [7:0] c, output [7:0] result
);
  wire   [7:0] d;
  // d = max(a, b)
  assign d = (a>=b) ? a:b;
  // result = max(d, c)
  assign result = (d>=c) ? d:c;
  // assertions
  assert property (a <= result);
  assert property (b <= result);
  assert property (c <= result);
endmodule
```

Motivating Example

- In Formal verification, specification is important.
  - Ensure correctness
  - Reused for regression test
  - Compositional verification
  - Help for unbounded checking
- **60–70% of development time** is spent on **manual specifications**

Background – Verilog & SVA

- Verilog: HDL for digital hardware (structural & behavioral).
- System Verilog Assertions (SVA):
  - Specify invariants that must hold for all inputs.
  - Checked by formal verification tools.
  - Example: assert property (a <= result);

Problem Statement - Informal

- Given a hardware design, to find a specification made up of SVAs
  - Valid for the design
  - Non-trivial
    - Trivial invariants like “assert (true)”

Problem Statement - formal

- Given a Hardware design $H = < V_H, I, S, s_0, T >$
- We try to find a set of assertion A that contains of α
- For each α, it should model the Hardware design
  - In all reachable state S, assertion α should be true
  - $S_\alpha = \{S \mid \alpha(s) = \top\}, \mathrm{S} \subseteq S_\alpha$
  - We say $H \vDash \alpha$

Why Existing Tools Fail

- Existing tools Harm[1] and GoldMine[2]:
  - Use Trace-based, pattern matching
    - Require high quality of traces
    - Require templates or hints.
- Boolean circuits only.

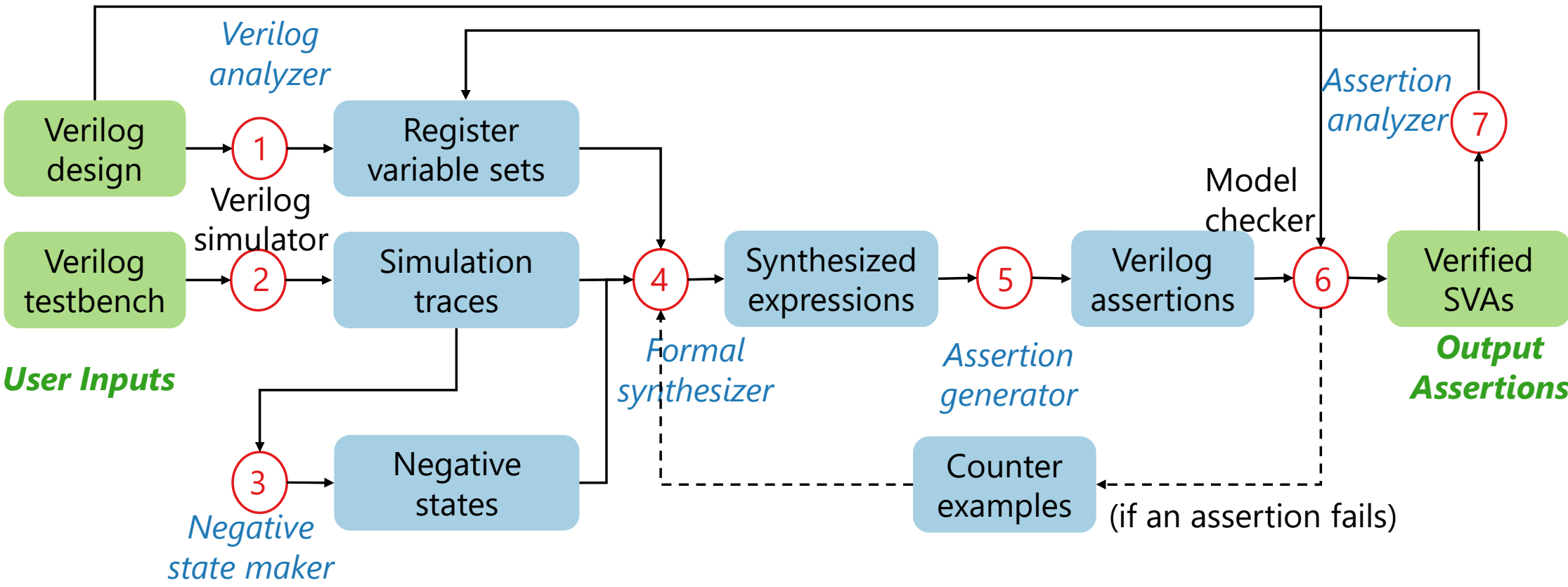
```
assert property (a <= result);
assert property (b <= result);
assert property (c <= result);
```

Our solution: SMART

We present **S**pecification **M**ining of **A**ssertions, Refined via **T**races (SMART): a specification mining method for hardware designs based on oracle-guided inductive synthesis.

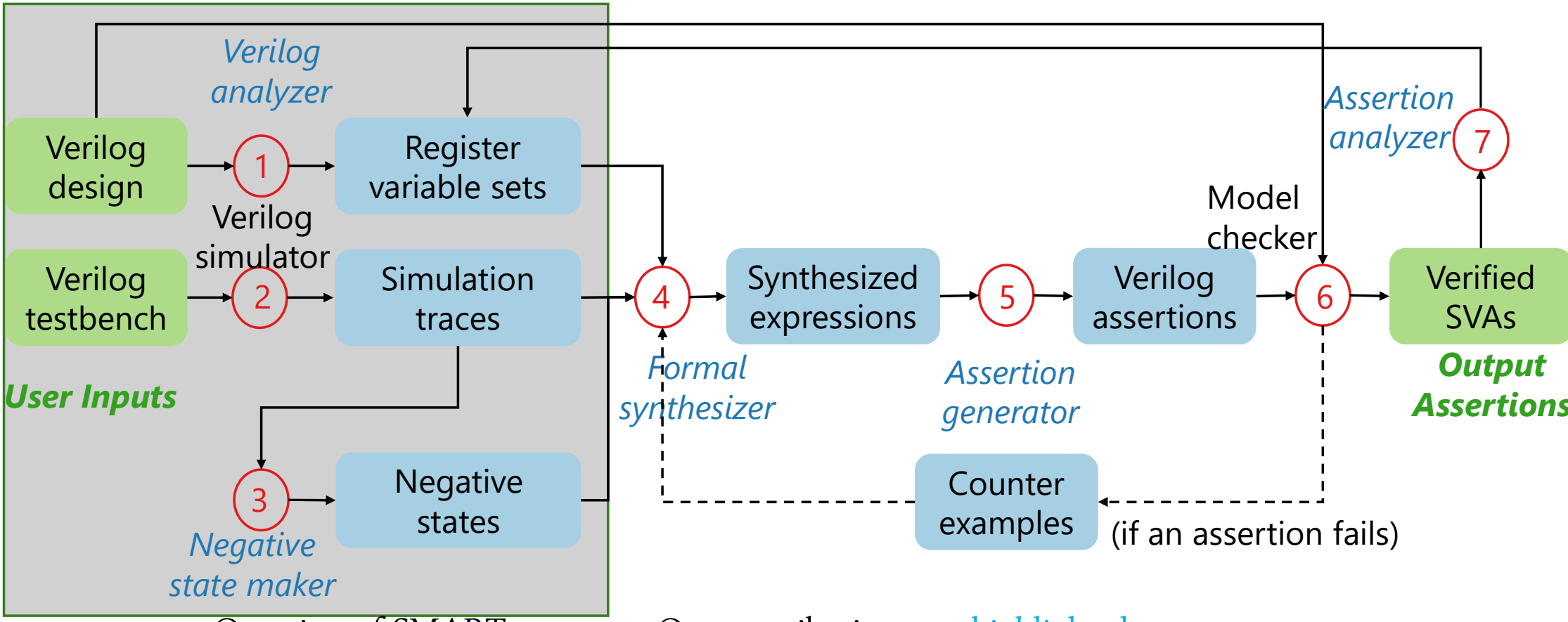
- uses **simulation traces**, **model checking** and **counterexamples to guide synthesis**
- generates formally verified specifications
- differentiate more design changes than SOTA

SMART Structure



Overview of SMART structure. Our contributions are highlighted.

Pre-synthesis



Overview of SMART structure. Our contributions are highlighted.

LY1

Pre-synthesis: step-1 Verilog analyzer

- Identify register variables that change with clock V_π
- Subset variables selection from V_i to reduce the synthesis space for large problem

$$|V_i| = \begin{cases} |V_\Pi| - 1, & \text{if } |V_\Pi| \leq 5 \\ 5, & \text{if } 5 < |V_\Pi| \leq 400, \\ 20, & \text{if } 400 < |V_\Pi|, \end{cases}$$

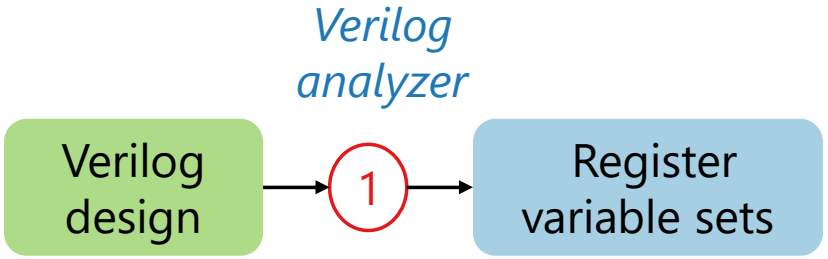
Slide 11

LY1 Spend more content on how varibale set are designed.
Leiqi Ye, 2025-10-02T16:37:50.862

LY1 0 Add more detail.
Leiqi Ye, 2025-10-02T16:38:37.697

LY1 1 Below all the slide
Leiqi Ye, 2025-10-02T16:39:37.010

Pre-synthesis: step-1 Verilog analyzer




```
// Return the max value of three
inputs.
module max3(
  input  [7:0] a, input  [7:0]
b,
  input  [7:0] c, output [7:0]
result
);
  wire   [7:0] d;
  // d = max(a, b)
  assign d = (a>=b) ? a:b;
  // result = max(d, c)
  assign result = (d>=c) ? d:c;

endmodule
```

$\{a, b, d, result\}$

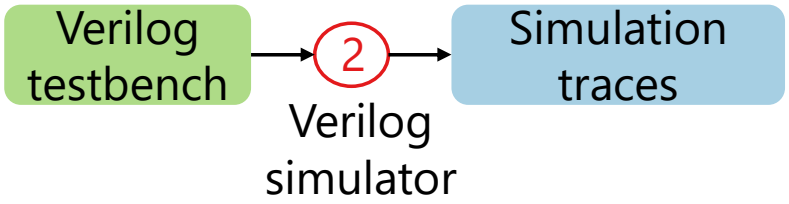
$\{c, b, d, result\}$

$\{c, b, result\}$

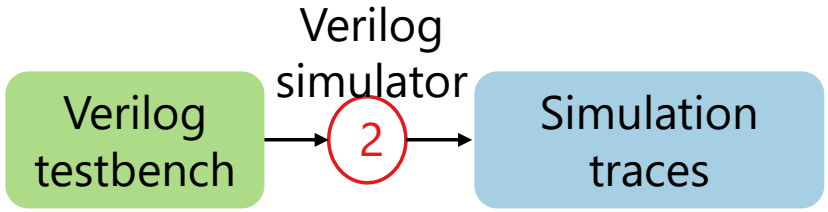
.....

Pre-synthesis: step-2 Verilog Simulation

- Verilog simulator
  - Collect simulation traces
  - Make full random assignment to i/o of the design
  - Positive examples P from runtime traces



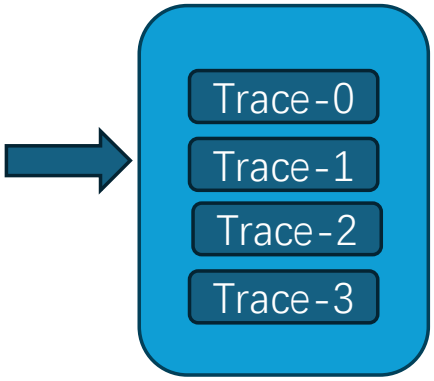
Pre-synthesis: step-2 Verilog Simulation



User Inputs

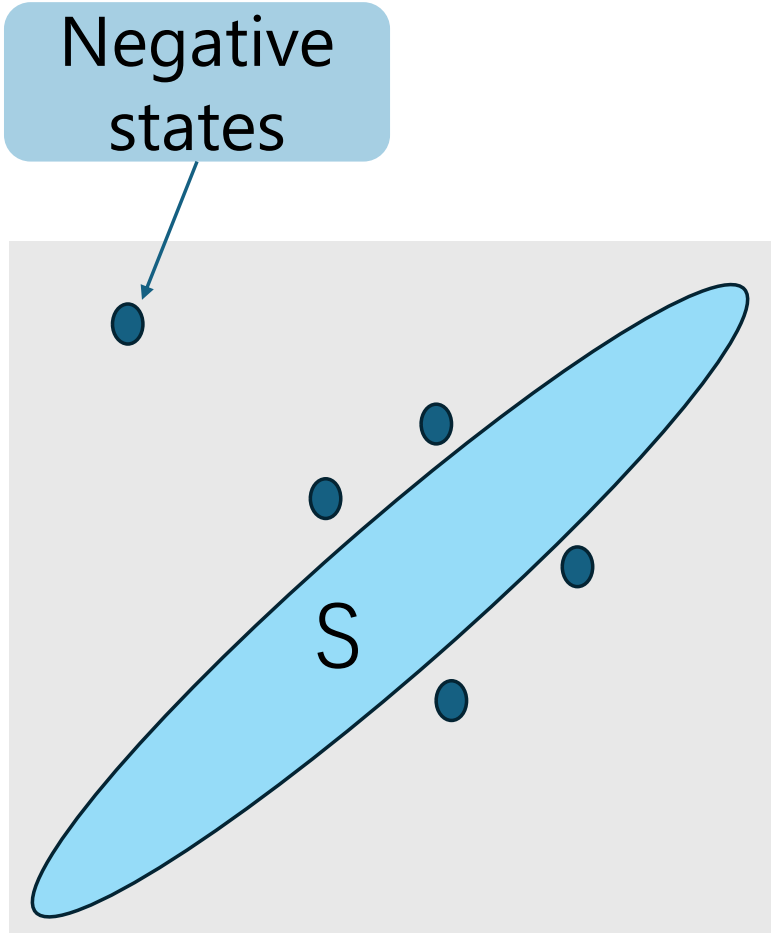
```
// Return the max value of three
inputs.
module max3(
  input  [7:0] a, input  [7:0]
b,
  input  [7:0] c, output [7:0]
result
);
  wire   [7:0] d;
  // d = max(a, b)
  assign d = (a>=b) ? a:b;
  // result = max(d, c)
  assign result = (d>=c) ? d:c;

endmodule
```

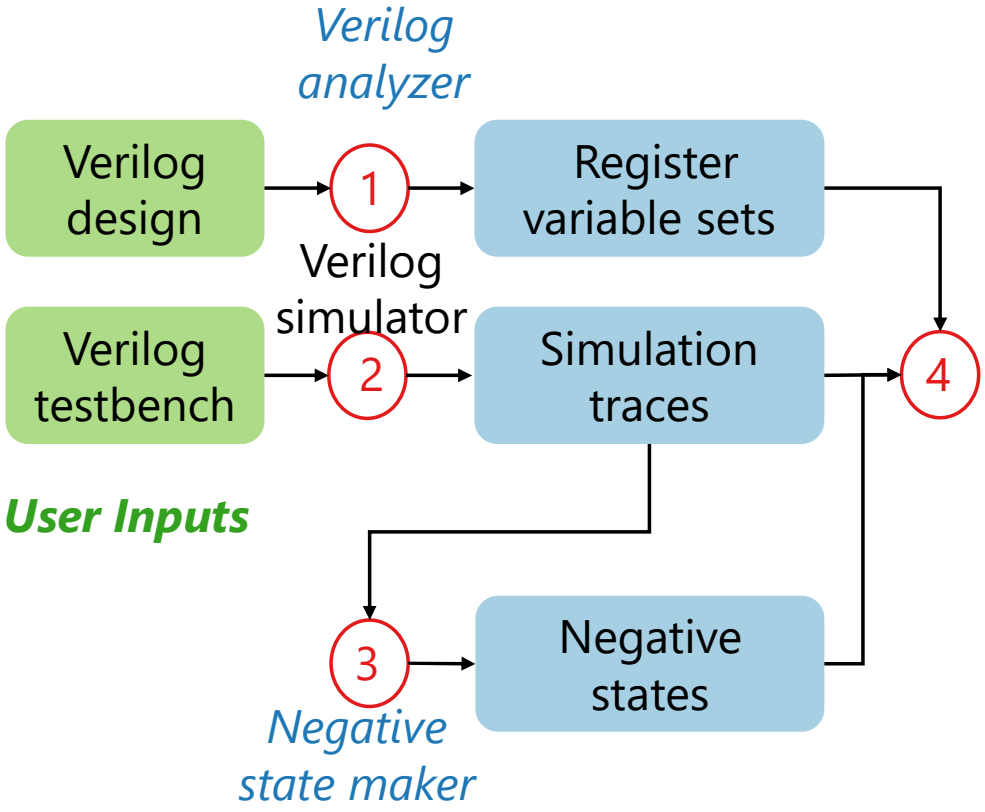


Pre-synthesis: step-3 Negative state maker

- Generate a state outside of model reachable state S
- Generate state ⟶ check reachability by model checker
  - If unreachable ⟶ add to N
  - If reachable ⟶ add to P



Pre-synthesis: step-3 Negative state maker



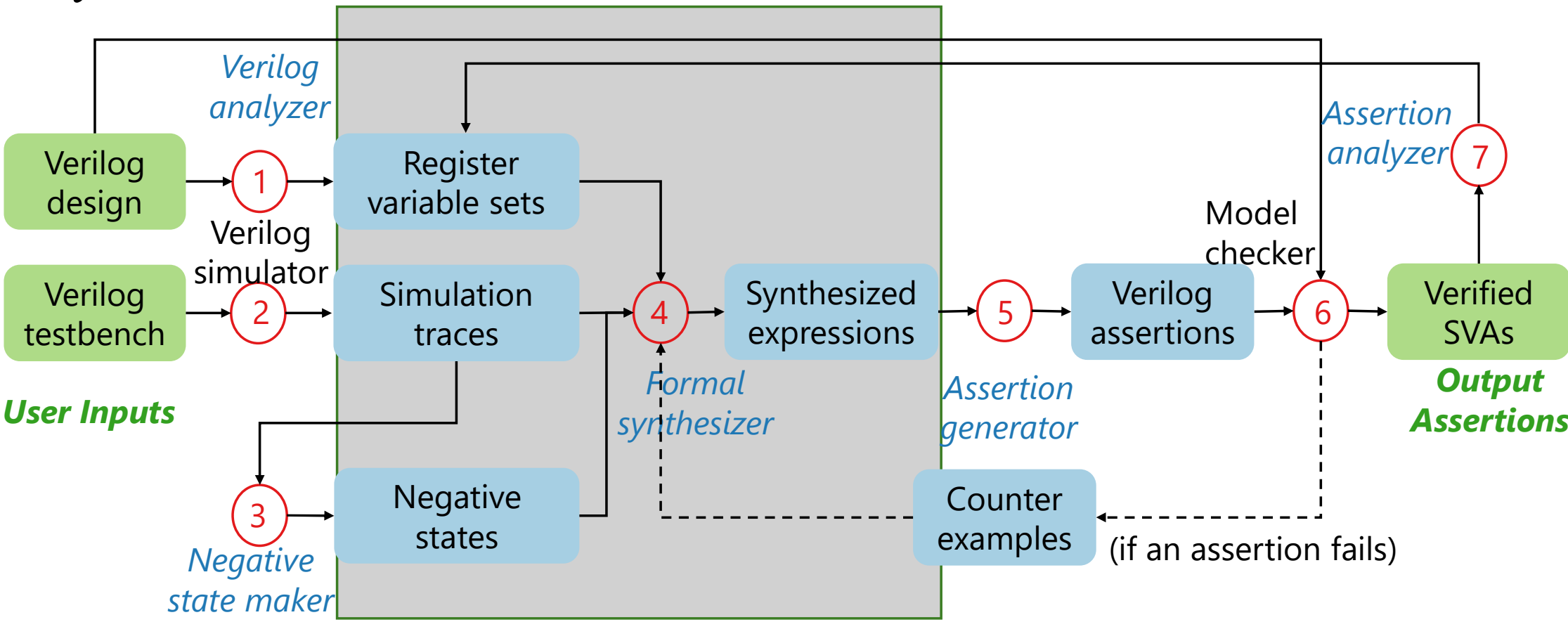
$\{a = 1, b = 1, d = 2, result = 3\} \in S_{reachable}$?

```
// Return the max value of three
inputs.
module max3(
  input  [7:0] a, input  [7:0]
b,
  input  [7:0] c, output [7:0]
result
);
  wire   [7:0] d;
  // d = max(a, b)
  assign d = (a>=b) ? a:b;
  // result = max(d, c)
  assign result = (d>=c) ? d:c;

endmodule
```

$!\{a = 1, b = 1, d = 2, result = 3\}$

Synthesis



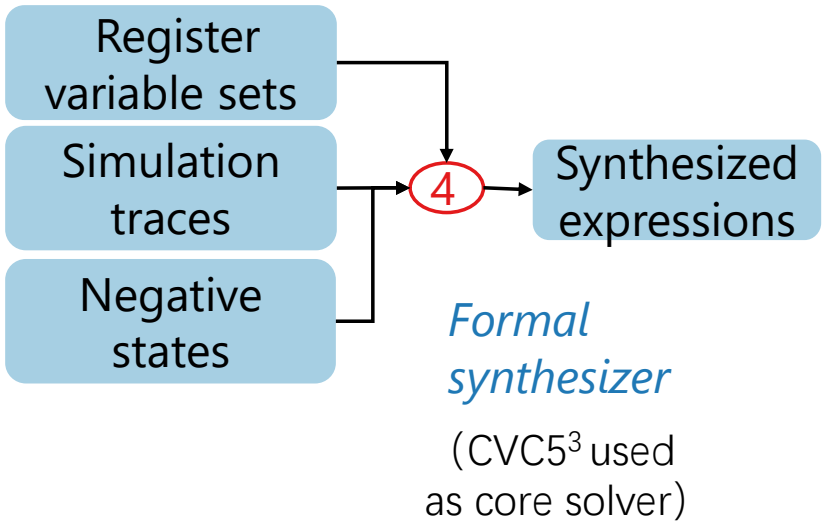
Overview of SMART structure. Our contributions are highlighted.

LY2

Step-4 Synthesis

Solve synthesis conjecture with Synthesis α:

- $\exists \alpha.$
- $\forall s^{+} \in P, \alpha(s^{+}) = \top$
- $\forall s^{-} \in N, \alpha(s^{-}) = \bot$

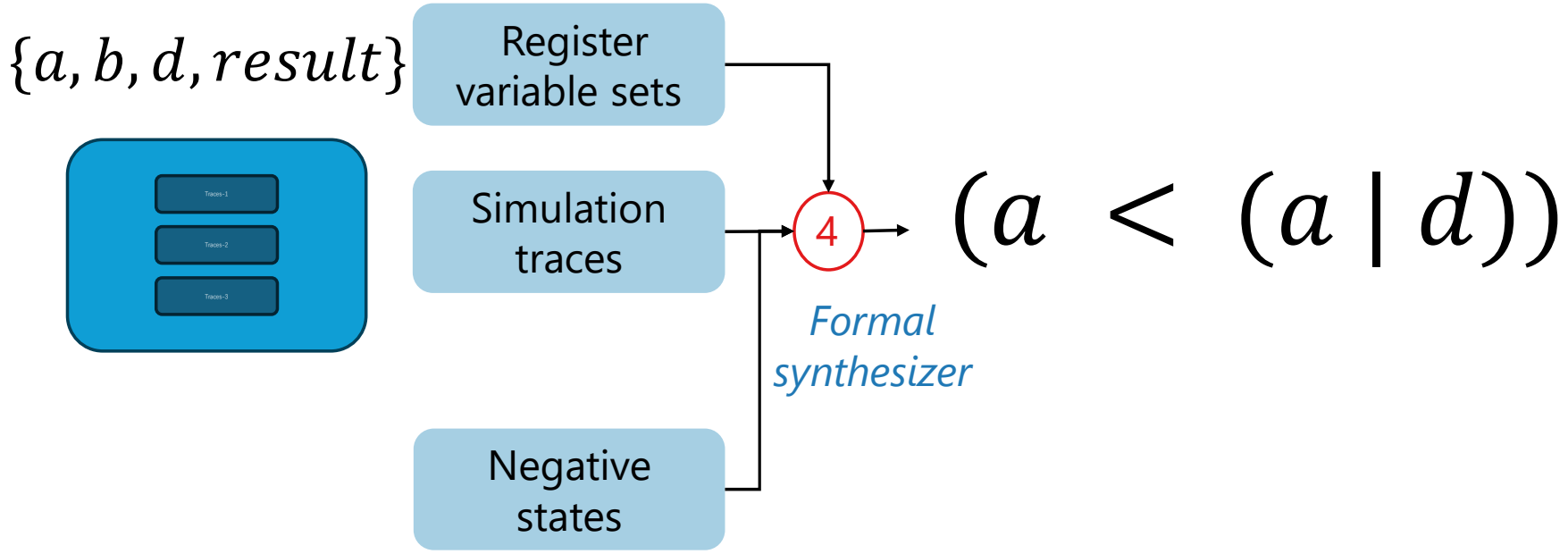


Slide 18

LY1 Correct about the example.
Leiqi Ye, 2025-10-02T16:59:41.625

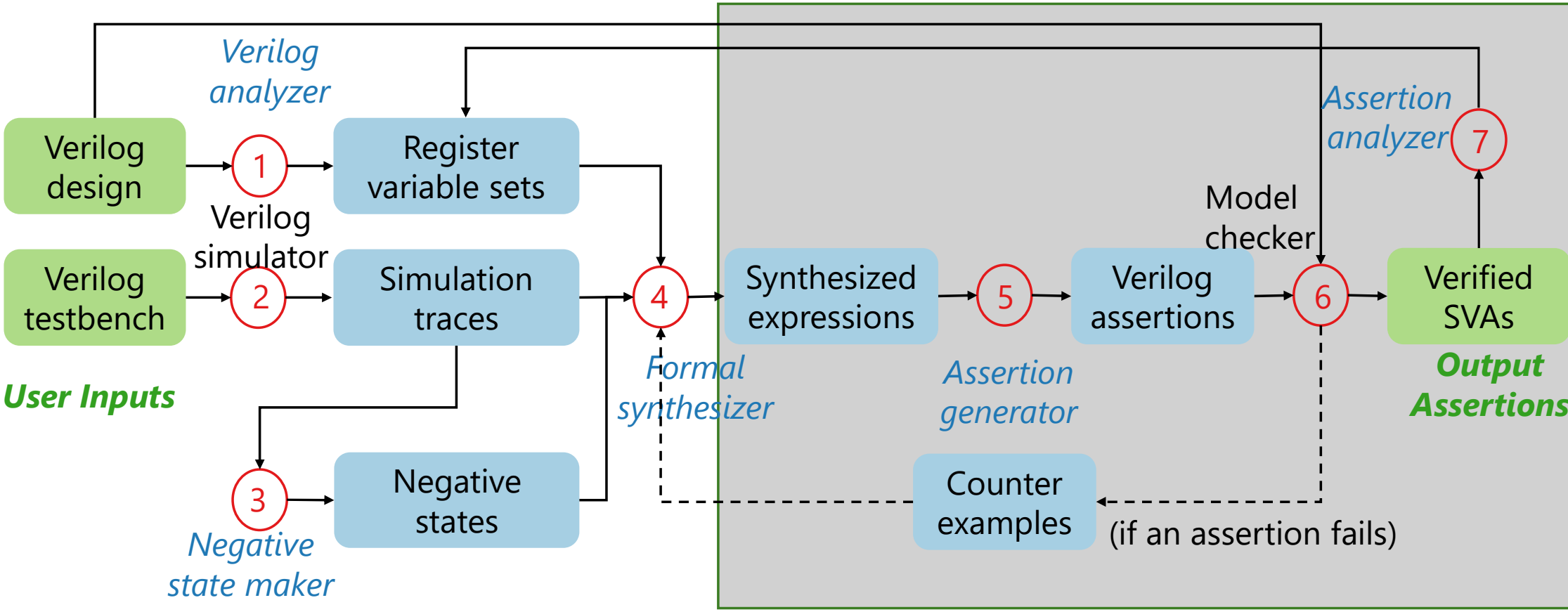
LY2 Say the we use CVC5
Leiqi Ye, 2025-10-03T15:14:29.606

Step-4 Synthesis



$!\{a = 1, b = 1, d = 2, result = 3\}$

Counterexample Refinement

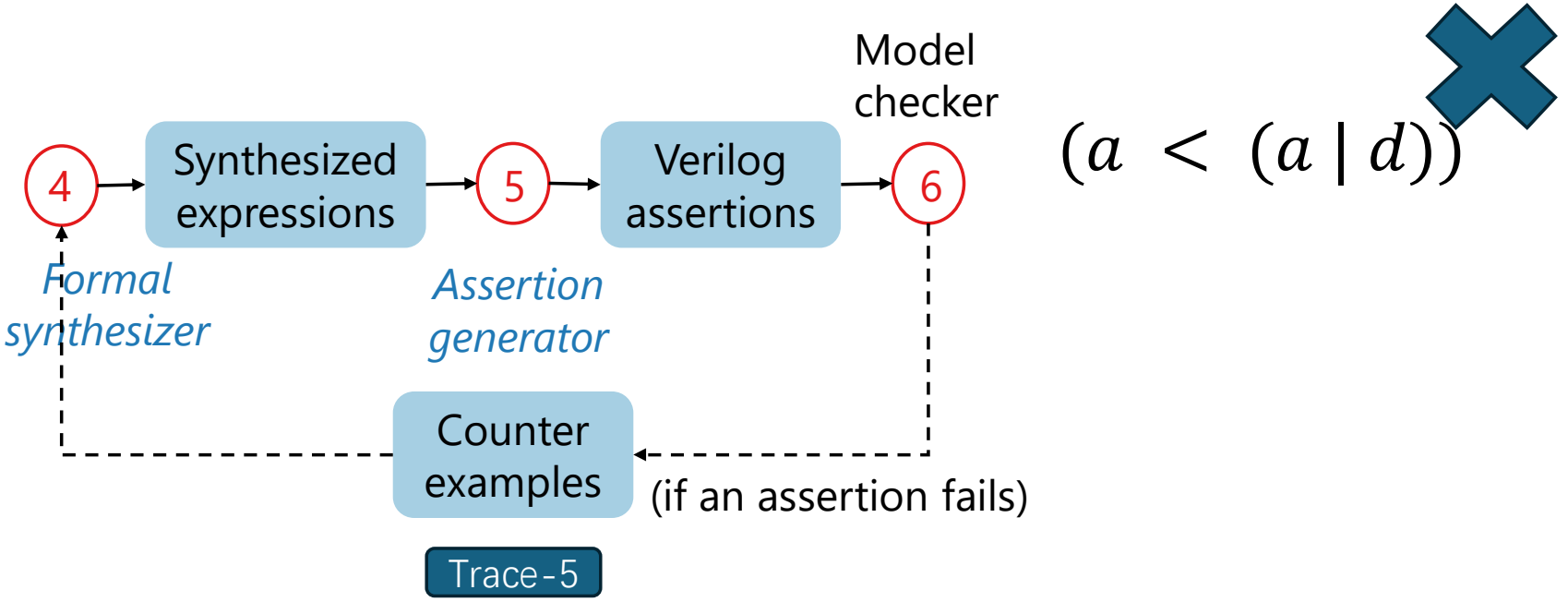


Overview of SMART structure. Our contributions are highlighted.

Step-5&6 Counterexample Refinement

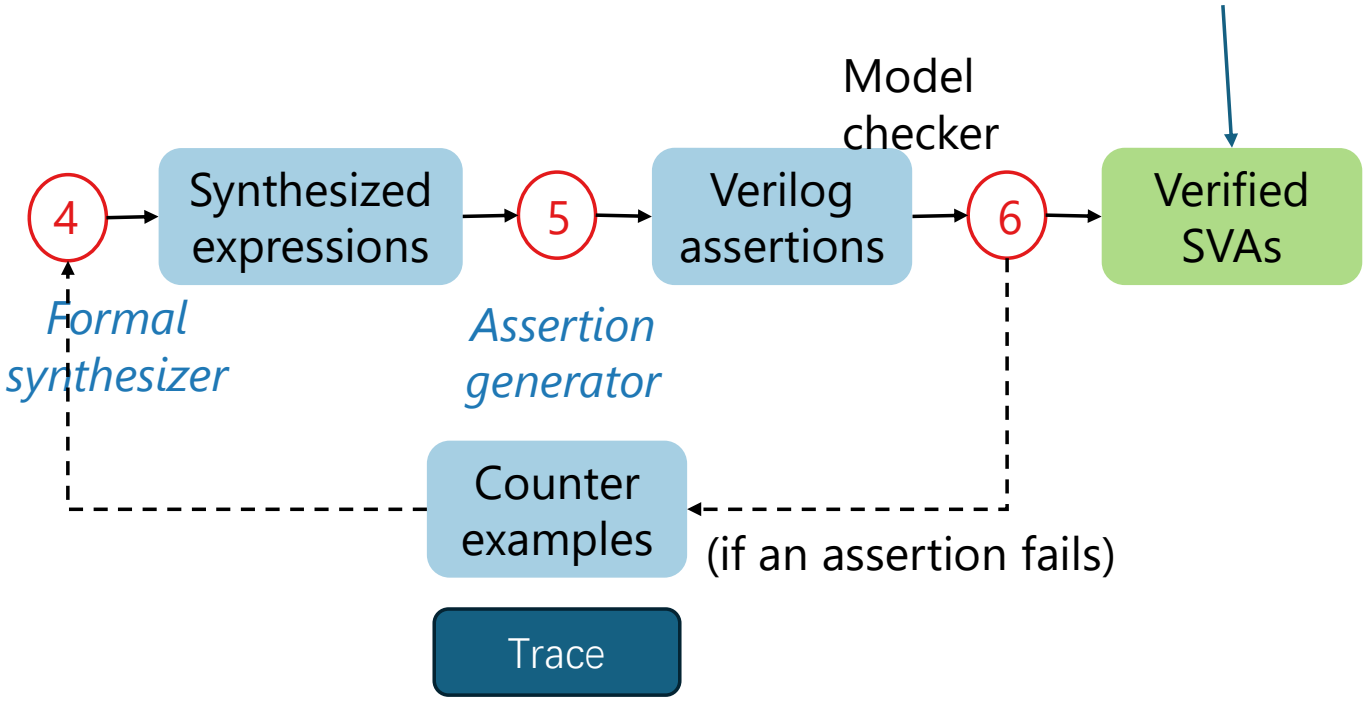
- Candidate assertion α is generated on subset variables
  - We only know $H_i \models \alpha$
- Verify α on Original model H
  - If pass: we know $H \models \alpha$
  - If fail: return counterexample trace
    - Add all states in the counterexample to P.

Step-5&6 Counterexample Refinement



Step-5&6 Counterexample Refinement

$$(result \leq (a \mid d))$$



Step-7 Assertion Analysis

- If assertion valid $\longrightarrow$ add to set A
- Analysis $\alpha \longrightarrow$ change subset variable policy
- Repeat until use up of subset variable sets
- Result: compact, verified, non-redundant assertion set

Step-7 Assertion Analysis

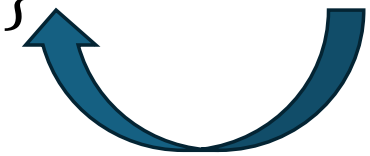
$\{a, b, d, result\}$ ⇒ $result <= (a \mid d)$

$\{a, b, result\}$ ⇒ $a \leq result$

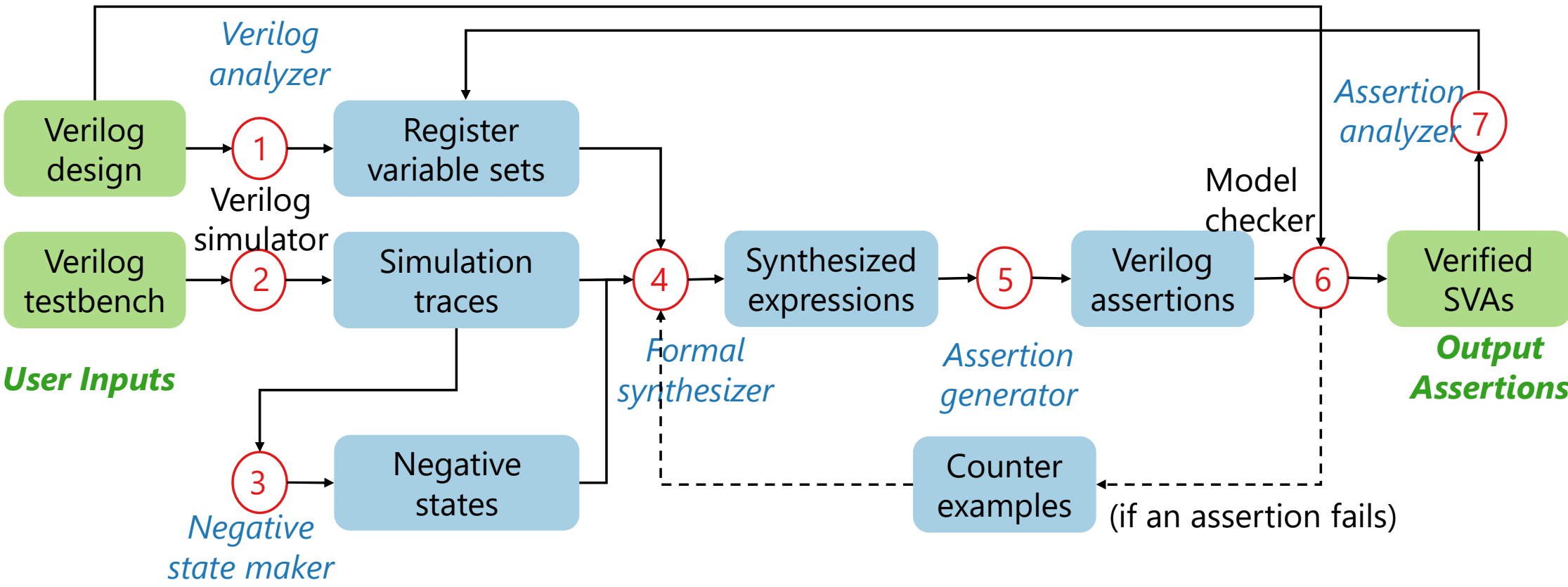
$\{c, b, result\}$

.....

$\{b, result\}$



SMART Structure



Overview of SMART structure. Our contributions are highlighted.

Evaluation Setup

- Benchmarks:
  - ISCAS'85 (combinational circuits)[4].
  - ISCAS'89 (sequential circuits)[3].
  - GoldMine benchmarks (CPU modules)[2].
- Tools compared: SMART, HARM[1], GoldMine[2].
- Metrics:
  - Verification correctness (VC-rate).
  - Mutation detection (MD-rate).

Evaluation – Verification Correctness

- We evaluate all the assertions generated by all comparing tool by model checker.
- The Verification correctness rate is:

$$VC = \frac{|A_{correct}|}{|A|}$$

Evaluation – Mutation Detection

- What is MD?
  - Small random changes in Verilog code
  - Check if assertions can distinguish
    - original vs. mutated design
  - The metrics used by related work Harm[1]

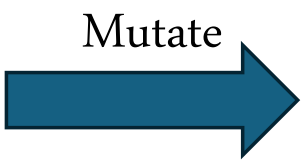
```
// Return the max value of
three inputs.
module max3(
  input  [7:0] a, input
[7:0] b,
  input  [7:0] c, output
[7:0] result
);
  wire   [7:0] d;
  // d = max(a, b)
  assign d = (a>=b) ? a:b;
  // result = max(d, c)
  assign result = (d>=c) ?
d:c;

endmodule
```

Evaluation – Mutation Detection

```
// Return the max value of
three inputs.
module max3(
  input  [7:0] a, input
[7:0] b,
  input  [7:0] c, output
[7:0] result
);
  wire   [7:0] d;
  // d = max(a, b)
  assign d = (a>=b) ? a:b;
  // result = max(d, c)
  assign result = (d>=c) ?
d:c;

endmodule
```




```
// Return the max value of
three inputs.
module max3(
  input  [7:0] a, input
[7:0] b,
  input  [7:0] c, output
[7:0] result
);
  wire   [7:0] d;
  // d = max(a, b)
  assign d = (a<=b) ? a:b;
  // result = max(d, c)
  assign result = (d>=c) ?
d:c;

endmodule
```

Evaluation – Mutation Detection

```
// Return the max value of three
inputs.
module max3(
  input  [7:0] a, input  [7:0] b,
  input  [7:0] c, output [7:0]
result
);
  wire   [7:0] d;
  // d = max(a, b)
  assign d = (a>=b) ? a:b;
  // result = max(d, c)
  assign result = (d>=c) ? d:c;
  // assertions
  assert property (a <= result);
  assert property (b <= result);
  assert property (c <= result);
endmodule
```

```
// Return the max value of three
inputs.
module max3(
  input  [7:0] a, input  [7:0] b,
  input  [7:0] c, output [7:0]
result
);
  wire   [7:0] d;
  // d = max(a, b)
  assign d = (a<=b) ? a:b;
  // result = max(d, c)
  assign result = (d>=c) ? d:c;
  // assertions
  assert property (a <= result);
  assert property (b <= result);
  assert property (c <= result);
endmodule
```

Evaluation – Mutation Detection

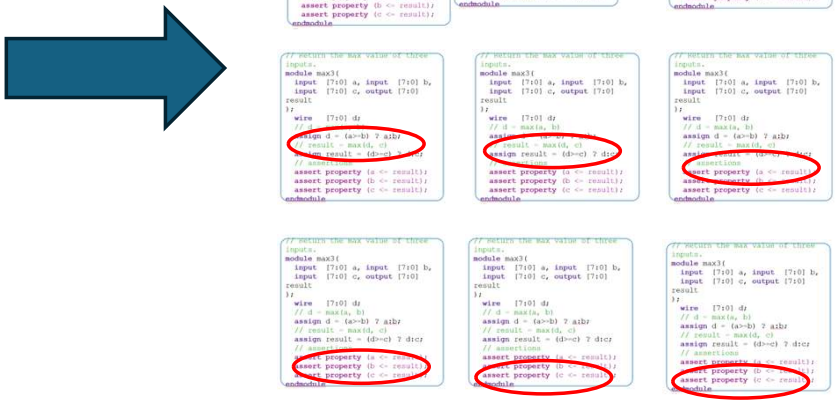
```
// Return the max value of three
inputs.
module max3(
  input  [7:0] a, input  [7:0] b,
  input  [7:0] c, output [7:0]
result
);
  wire   [7:0] d;
  // d = max(a, b)
  assign d = (a>=b) ? a:b;
  // result = max(d, c)
  assign result = (d>=c) ? d:c;
  // assertions
  assert property (a <= result);
  assert property (b <= result);
  assert property (c <= result);
endmodule
```

$$(\wedge \mid \uparrow \mid \downarrow \mid \oplus \mid \odot \mid \neg \mid buf) \xrightarrow{r_{g1}} \vee$$
$$(\vee \mid \uparrow \mid \downarrow \mid \oplus \mid \odot \mid \neg \mid buf) \xrightarrow{r_{g2}} \wedge$$
$$(\vee \mid \wedge \mid \downarrow \mid \oplus \mid \odot \mid \neg \mid buf) \xrightarrow{r_{g3}} \uparrow$$
$$(\vee \mid \wedge \mid \uparrow \mid \oplus \mid \odot \mid \neg \mid buf) \xrightarrow{r_{g4}} \downarrow$$
$$(\vee \mid \wedge \mid \uparrow \mid \downarrow \mid \odot \mid \neg \mid buf) \xrightarrow{r_{g5}} \oplus$$
$$(\vee \mid \wedge \mid \uparrow \mid \downarrow \mid \oplus \mid \neg \mid buf) \xrightarrow{r_{g6}} \odot$$
$$(\vee \mid \wedge \mid \uparrow \mid \downarrow \mid \oplus \mid \odot \mid buf) \xrightarrow{r_{g7}} \neg$$
$$(\vee \mid \wedge \mid \uparrow \mid \downarrow \mid \oplus \mid \odot \mid \neg) \xrightarrow{r_{g8}} buf$$
$$(\circ \mid \hat{} \,) \xrightarrow{r_{b1}} \&$$
$$(\& \mid \hat{} \,) \xrightarrow{r_{b2}} \circ$$
$$(\& \mid \circ) \xrightarrow{r_{b3}} \hat{}$$
$$\text{bitvector literal} \xrightarrow{r_{ran}} \text{random value}$$

$$(- \mid * \mid / \mid \%) \xrightarrow{r_{a1}} +$$
$$(+ \mid * \mid / \mid \%) \xrightarrow{r_{a2}} -$$
$$(+ \mid - \mid / \mid \%) \xrightarrow{r_{a3}} *$$
$$(+ \mid - \mid * \mid \%) \xrightarrow{r_{a4}} /$$
$$(+ \mid - \mid * \mid /) \xrightarrow{r_{a5}} \%$$
$$(\neq \mid > \mid < \mid \geq \mid \leq) \xrightarrow{r_{r1}} \simeq$$
$$(\simeq \mid > \mid < \mid \geq \mid \leq) \xrightarrow{r_{r2}} \neq$$
$$(\simeq \mid \neq \mid < \mid \geq \mid \leq) \xrightarrow{r_{r3}} >$$
$$(\simeq \mid \neq \mid > \mid \geq \mid \leq) \xrightarrow{r_{r4}} <$$
$$(\simeq \mid \neq \mid > \mid < \mid \leq) \xrightarrow{r_{r5}} \geq$$
$$(\simeq \mid \neq \mid > \mid < \mid \geq) \xrightarrow{r_{r6}} \leq$$
$$\text{bool} \xrightarrow{r_{neg}} \neg\text{bool}$$

Evaluation – Mutation Detection

```
// Return the max value of three
inputs.
module max3(
  input  [7:0] a, input  [7:0] b,
  input  [7:0] c, output [7:0]
result
);
  wire   [7:0] d;
  // d = max(a, b)
  assign d = (a>=b) ? a:b;
  // result = max(d, c)
  assign result = (d>=c) ? d:c;
  // assertions
  assert property (a <= result);
  assert property (b <= result);
  assert property (c <= result);
endmodule
```



Evaluation – Mutation Detection

- We evaluate all the assertions on mutation benchmark by model checker.
- The Mutation Detection rate is:

$$\frac{|\{H' \in H \mid H' \neg\models \alpha\}|}{|H|}$$

Comparison with other tools

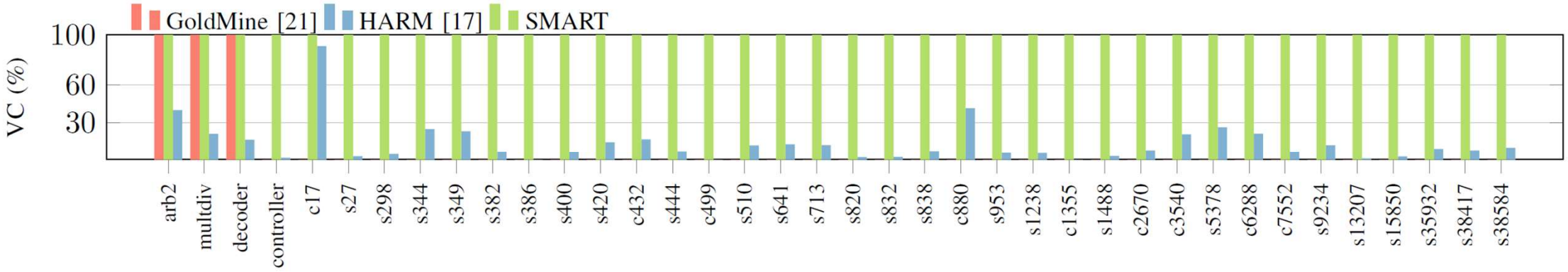


Fig. 3: Verification Correctness (VC) rates of different approaches over various benchmarks.

SMART produces assertions that is all formal verified on the original design.

Comparison with other tools

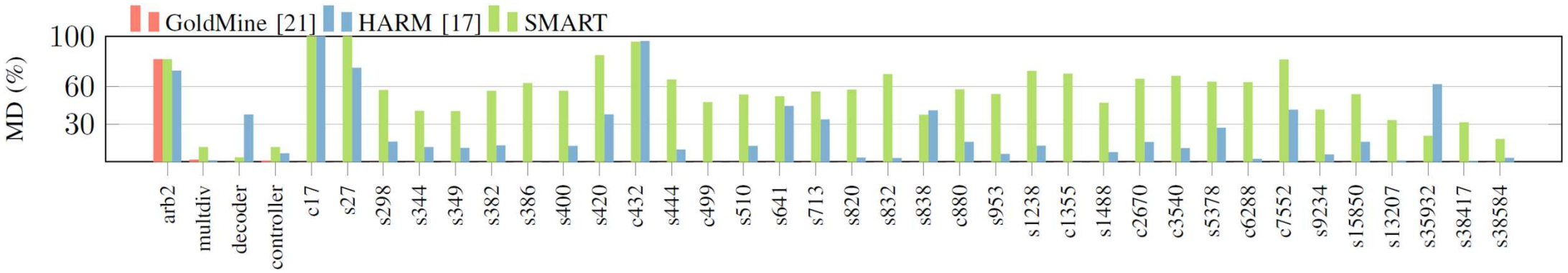


Fig. 4: Mutation detection (MD) rates of different approaches over various benchmarks.

	Structural Verilog (34 benchmarks)				Behavioral Verilog (4 benchmarks)				All (38 benchmarks)			
	#	$\|A_{gen}\|$	VC-rate	MD-rate	#	$\|A_{gen}\|$	VC-rate	MD-rate	#	$\|A_{gen}\|$	VC-rate	MD-rate
GoldMine	0	0	n/a	0.0%	4	38.8	100%	21.16%	4	38.8	100%	4.3%
HARM	34	60676	8.41%	21.39%	4	3655	8.6%	29.6%	38	54674	8.4%	22.25%
SMART	34	1069	100%	55.41%	4	47	100%	25.17%	38	943.5	100%	52.23%

LY1

Impact of Counterexample of refinement

Fig. 5: Impact of Counterexamples on SMART's Mutation Detection Performance.

Slide 37

LY1 Eliazebeth: an you present the average sizes on behavioural and on structural verilog separately? otherwise hard to compare goldmine to others

Leiqi Ye, 2025-09-24T16:34:34.388

LY1

Assertion Set Size & Readability

- SMART: avg. 943 assertions.
- HARM: avg. >50,000 assertions.
- GoldMine: ~39 (but only behavioral Verilog).
- SMART: compact, less redundant.

LY1 Separation material

Leiqi Ye, 2025-09-25T09:35:12.519

Limitations

- Only single-cycle properties.
- Runtime bottleneck: model checking.
- VC and MD-rate is still not enough for evaluating “good specification”

Conclusion

- Presented SMART: Oracle-guided synthesis for hardware verification.
- Generates correct, meaningful SVAs.
- Outperforms state-of-the-art on correctness & detection.
- Future: temporal properties, scalability.
- https://github.com/lichye/smartVerilog