Template-based Circuit Understanding

Adrià Gascón\textsuperscript{1}  Pramod Subramanyan\textsuperscript{2}  Bruno Dutertre\textsuperscript{1}  
Ashish Tiwari\textsuperscript{1}  Dejan Jovanović\textsuperscript{1}  Sharad Malik\textsuperscript{2}

\textsuperscript{1}SRI International  
\textsuperscript{2}Princeton University
Verify/reverse-engineer a digital circuit
⇒
EXTRACT and UNDERSTAND subcomponents
Verify/reverse-engineer a digital circuit
⇒
EXTRACT and UNDERSTAND subcomponents

- FSM extraction [Shi et. al.]
- Functional aggregation and matching [Subramanyan et. al.]
- Word identification and propagation [Li et. al.]
- Identification of repeated structures [Hansen et. al.]
Verify/reverse-engineer a digital circuit

⇒

**EXTRACT** and **UNDERSTAND** subcomponents

- FSM extraction [Shi et. al.]
- Functional aggregation and matching [Subramanyan et. al.]
- Word identification and propagation [Li et. al.]
- Identification of repeated structures [Hansen et. al.]

Most of these techniques do not find the right permutations in word components
Verify/reverse-engineer a digital circuit

⇒

EXTRACT and UNDERSTAND subcomponents
What does it mean to understand a combinational circuit $C$?

- Find an equivalent higher-level definition
  - Flatten verilog netlist $\rightarrow$ High-level Verilog
  - Basic Boolean logic $\rightarrow$
    - Boolean Logic $+$ Words and operations on Words
What does it mean to understand a combinational circuit $C$?

- Find an equivalent higher-level definition
  - Flatten verilog netlist $\rightarrow$ High-level Verilog
  - Basic Boolean logic $\rightarrow$
    
    Boolean Logic + Words and operations on Words

**Goal**

Given purely Boolean Formula $C$, produce “equivalent” Formula $\mathcal{F}$ over the theory of bitvectors.
A Combinational Boolean circuit $C(I, O)$ is
(a) a list of input Boolean variables $I = \langle x_1, \ldots, x_n \rangle$ and
(b) a list $O = \langle f_1, \ldots, f_m \rangle$ of single-output Boolean formulas with inputs $I$.

For $\vec{x} \in \{0, 1\}^n, \vec{y} \in \{0, 1\}^m$, by $C(\vec{x}, \vec{y})$ we denote that $C$ produces output $\vec{y}$ on input $\vec{x}$.
The library approach

Check functional equivalence against a library of known components.

- $C(\langle x_1, \ldots, x_n \rangle, \langle f_1, \ldots, f_m \rangle)$
- $C_{lib}(\langle x_1, \ldots, x_n \rangle, \langle g_1, \ldots, g_m \rangle)$
- **Fixed** permutations $\sigma, \theta$

$$\forall \ i \in \{1, \ldots, m\}, \bar{x} \in \{0, 1\}^m : f_{\theta(i)}(\sigma(\bar{x})) = g_i(\bar{x})$$

Limitation: Permutations $\sigma, \theta$ must be known.
The library approach

Check functional equivalence against a library of known components.

- $C(\langle x_1, \ldots, x_n \rangle, \langle f_1, \ldots, f_m \rangle)$
- $C_{\text{lib}}(\langle x_1, \ldots, x_n \rangle, \langle g_1, \ldots, g_m \rangle)$
- **Fixed** permutations $\sigma, \theta$

\[
\forall \ i \in \{1, \ldots, m\}, \bar{x} \in \{0, 1\}^m : f_{\theta(i)}(\sigma(\bar{x})) = g_i(\bar{x})
\]

**Limitation:** Permutations $\sigma, \theta$ must be known.
Permutation-independent equivalence checking

- $\mathcal{C}(\langle x_1, \ldots, x_n \rangle, \langle f_1, \ldots, f_m \rangle)$
- $\mathcal{C}_{lib}(\langle x_1, \ldots, x_n \rangle, \langle g_1, \ldots, g_m \rangle)$
- To be determined permutations $\sigma$, $\theta$

$\exists \sigma, \theta :$

$\forall i \in \{1, \ldots, m\}, \vec{x} \in \{0, 1\}^m :$

$f_{\theta(i)}(\sigma(\vec{x})) = g_i(\vec{x})$

Limitation: Still too restrictive.

1. $\mathcal{C}$ usually does not have a "standard" functionality.
2. $\mathcal{C}_{lib}$'s functionality must be fully matched.
Permutation-independent equivalence checking

- $\mathcal{C}(\langle x_1, \ldots, x_n \rangle, \langle f_1, \ldots, f_m \rangle)$
- $\mathcal{C}_{lib}(\langle x_1, \ldots, x_n \rangle, \langle g_1, \ldots, g_m \rangle)$
- To be determined permutations $\sigma$, $\theta$

$\exists \sigma, \theta :$

$\forall i \in \{1, \ldots, m\}, \vec{x} \in \{0,1\}^m :$

$f_{\theta(i)}(\sigma(\vec{x})) = g_i(\vec{x})$

Limitation: Still too restrictive.

1. $\mathcal{C}$ usually does not have a “standard” functionality.
2. $\mathcal{C}$’s functionality must be fully matched.
Template-based synthesis

Instead of a reference circuit, our approach requires a template of a specific form.
How do our templates look like?

A template $T$ of a combinational circuit $C(I, O)$ is:

- A subset $O_T \subseteq O$,
- a partition $I = (I_C \cup \bigcup_{i=1}^n (W_i))$, and
- a conjunction of guarded assignments of the form

$$a_i : \psi_i(I_C) \Rightarrow (\theta(O_T) := \phi_i(\sigma(W_{i_1}), \tau(W_{i_2})))$$

where

- $\psi_i$ is a to be determined assignment on $I_C$,
- $\theta, \sigma, \tau$ are to be determined permutations, and
- $\phi_i$ is a binary function over words.
- $i_1, i_2 \in \{1, \ldots, n\}.$
1. Circuit $C(I, O)$
2. Subset $outputs := O$
3. Partition $I := control \cup inputsA \cup inputsB$
4. Template with
   (a) To be determined assignments $v1$, $v2$
   (b) To be determined permutations $p$, $q$

```lisp
(and
  (=>
    (= outputs
        (bv-add
          (permute p inputsA)
          (permute q inputsB))
      (value v1 control))
    (= outputs
        (ite
          (bv-slt
            (permute p inputsA)
            (permute q inputsB))
          (mk-bv 32 1)
          (mk-bv 32 0))
      (value v2 control)))
```

1. Circuit $C(I, O)$
2. Subset outputs := $O$
3. Partition $I := \text{control} \cup \text{inputsA} \cup \text{inputsB}$
4. Template with
   (a) To be determined assignments $v_1, v_2$
   (b) To be determined permutations $p, q$

$$\exists p, q, v_1, v_2 : $$

$$\forall \bar{x} \in \{0, 1\}^n, \bar{y} \in \{0, 1\}^m : $$

$$C(\bar{x}, \bar{y}) \Rightarrow T(p, q, v_1, v_2, \bar{x}, \bar{y})$$
Check validity of Boolean formulas over the theory of bit-vectors with two levels of quantification ($\exists \forall$ QF BV):

$$\exists \vec{x} : C(\vec{x}) \land \forall \vec{y} : A(\vec{x}, \vec{y})$$

1. High-level preprocessing and simplifications [Wintersteiger et. al.]
2. Counterexample-refinement loop, similar to the approach used in 2QBF solvers [Ranjan et. al., Janota et. al.]
3. Functional signatures [Mohnke et. al.]
(1) Miniscoping:

\[ \exists \vec{x} : A \lor B \rightarrow \exists \vec{x} : A \lor \exists \vec{x} : B \]

\[ \forall \vec{x} : A \land B \rightarrow \forall \vec{x} : A \land \forall \vec{x} : B \]

(2) Equality resolution:

\[ \exists \vec{x} : C(\vec{x}) \land \forall \vec{y} : (\bigwedge_{i}(y_i = x_i) \Rightarrow B(\vec{y})) \]

\[ \rightarrow \]

\[ \exists \vec{x} : E(\vec{x}) \land \forall \vec{y} : \bigcup_{i}(\{y_i \rightarrow x_i\})(B(\vec{y})) \]

(3) Distinguishing signatures.
Distinguishing Signatures

An output signature $s_{out}$ is a function $s_{out} : B_n \rightarrow D$ such that, for every function $f$ and permutation $\tau$:

$$s_{out}(f(x_1, \ldots, x_n)) = s_{out}(f(\tau(x_1), \ldots, \tau(x_n)))$$
Distinguishing Signatures

An output signature $s_{out}$ is a function $s_{out} : B_n \to D$ such that, for every function $f$ and permutation $\tau$:

$$s_{out}(f(x_1, \ldots, x_n)) = s_{out}(f(\tau(x_1), \ldots, \tau(x_n)))$$

$$\exists \sigma, \theta :$$

$$\forall i \in \{1, \ldots, m\}, \bar{x} \in \{0, 1\}^m :$$

$$f_{\theta(i)}(\sigma(\bar{x})) = g_i(\bar{x})$$
Distinguishing Signatures

An output signature $s_{out}$ is a function $s_{out} : B_n \rightarrow D$ such that, for every function $f$ and permutation $\tau$:

$$s_{out}(f(x_1, \ldots, x_n)) = s_{out}(f(\tau(x_1), \ldots, \tau(x_n)))$$

$$\exists \sigma, \theta :$$

$$\forall i \in \{1, \ldots, m\}, \vec{x} \in \{0, 1\}^m :$$

$$f_{\theta(i)}(\sigma(\vec{x})) = g_i(\vec{x})$$

$$\exists x, y : s_{out}(f_x) \neq s_{out}(g_y) \Rightarrow \theta(y) \neq x$$
Distinguishing Signatures

An output signature $s_{out}$ is a function $s_{out} : \mathcal{B}_n \rightarrow \mathcal{D}$ such that, for every function $f$ and permutation $\tau$:

$$s_{out}(f(x_1, \ldots, x_n)) = s_{out}(f(\tau(x_1), \ldots, \tau(x_n)))$$

$$\exists \sigma, \theta :$$

$$\forall i \in \{1, \ldots, m\}, \bar{x} \in \{0, 1\}^m :$$

$$f_{\theta(i)}(\sigma(\bar{x})) = g_i(\bar{x}) \land \theta(y) \neq x$$

$$\exists x, y : s_{out}(f_x) \neq s_{out}(g_y) \Rightarrow \theta(y) \neq x$$
We consider one input signature and one output signature.

- Input dependency
- Output dependency

Signatures can be computed *independently* in the circuit and the template.
Experiments

Benchmarks (40 Sat/40 Unsat):

- Reverse engineering benchmarks generated from high-level (behavioral) Verilog using the Synopsys Compiler.
- From ISCAS, an academic processor implementation, and synthetic examples.
- ALUs, multipliers, shifters, counters...

Tools:

- Yices (Yices format)
- Z3 (SMT2 format)
- Bloqger + DepQBF (QDimacs)
- Bloqger + RareQs (QDimacs)
- Bloqger + sKizzo (QDimacs)
- Cir-CEGAR (Mini-SAT) (QDimacs + top literal)

Variants:

- Considered two simple encodings for permutations
- Studied effect of preprocessing, encodings, and signatures
Conclusion and further work

- Yices and Z3 are sensitive to the encoding of permutations
- Preprocessing and signatures are harmless and crucial in many cases
- Benchmarks are available in SMT2, YICES, QBF and (soon) QCIR
- Just putting together two SAT/SMT solvers is not enough
- QDIMACS encoding is not suitable for this kind of synthesis
- Integrate signature computation in the Exist-Forall loop
- Compare to other synthesis algorithms
Questions? Comments? Suggestions?