SAT-Based Methods for Circuit Synthesis*

October 22, 2014

Roderick Bloem
Patrick Klampfl
Robert Könighofer

Uwe Egly
Florian Lonsing

* This work was supported in part by the Austrian Science Fund (FWF) through the national research network RiSE (S11406-N23, S11409-N23) and the project QUAIN (I774-N23), as well as by the European Commission through project STANCE (317753).

www.iaik.tugraz.at
What is Synthesis?

- **Specification:** What?
  - **From:** Graz, Inffeldgasse
  - **To:** Lausanne, 6pm

- **Implementation:** How?
  - Walk to Moserhofgasse
  - Tram 6 to Jakominiplatz
    - Buy tram ticket
  - Tram 3 to train station Graz
  - Buy train ticket
  - Train to Salzburg
  - Train to Zürich
  - Train to Lausanne
  - Walk to Lausanne Fon
  - And so on …
Reactive Synthesis

- Specification:
  - Temporal Logic
    - always($r_1 \rightarrow \text{eventually}(g_1))$
    - always($r_2 \rightarrow \text{eventually}(g_2))$
    - never($g_1 \land g_2$)

- Implementation:
  - Reactive system
Typical Synthesis Flow

- **Specification**
  - always\((i_1 \rightarrow \text{eventually}(o_1))\)

- **Game**

- **Strategy**
  - Focus of this work

- **Environment**
  - Formula: \(S(\bar{i}, \bar{o})\)

- **System**

- **Circuit**
  - IF
    - Inputs \(\bar{i}\)
    - Outputs \(\bar{o}\)
  - THEN
    - | \(i_1\) | \(i_2\) | \(o_1\) | \(o_2\) |
      |-----|-----|-----|-----|
      | 0   | 0   | 1   | -   |
      | 0   | 1   | 1   | 1   |
      | 0   | 1   | 0   | 0   |
  - And so on …
Challenges

- Scalability
  - Symbolic algorithms
  - Traditionally: BDDs
  - This work: SAT/QBF
- Find small circuits
  - Low number of gates
  - Exploit freedom in $S(\bar{t}, \bar{o})$ wisely
- Our work:
  - Comparison of SAT/QBF-based methods
  - Optimizations
Method 1: QBF Certification

Given:
- $\forall i: \exists o: S(i, o)$

Find:
- Skolem function $\bar{o} = f(\bar{i})$

Existing Tool:
- QBFCert [SAT’12]
Method 2: Interpolation [ICCAD’09]

For one output after the other:

- Construct formulas `mustBeTrue(\bar{i})`, `mustBeFalse(\bar{i})`
  - `mustBeTrue(\bar{i}) \land mustBeFalse(\bar{i}) = UNSAT`
- Compute Interpolant `I(\bar{i})`
  - `mustBeTrue(\bar{i}) \rightarrow I(\bar{i}) \rightarrow \neg mustBeFalse(\bar{i})`
Method 3: Computational Learning [FMCAD’12]

For one output after the other:

- Construct formulas $\text{mustBeTrue}(\bar{i}), \text{mustBeFalse}(\bar{i})$
  - $\text{mustBeTrue}(\bar{i}) \land \text{mustBeFalse}(\bar{i}) = \text{UNSAT}$
- “Learn” Interpolant $I(\bar{i})$
  - Counterexample-guided refinement
  - Many options: SAT or QBF, …
Results:

Execution Time

- Cactus Plot

- QBF Learning
- Interpolation
- BDDs
- SAT Learning

Execution Time [sec] vs. Benchmarks
Results:
Circuit Size

- Cactus Plot

- Circuit Size [# Gates]

- BDDs

- Interpolation

- QBF Cert

- QBF Learning

- SAT Learning

Benchmarks

FMCAD 2014
Lausanne, October 22
Conclusions

- SAT-based learning works best
- Execution time and circuit size correlate
- Check out the paper for details
  - Optimizations
  - More results
- Implementation is available:
  - http://www.iaik.tugraz.at/content/research/design_verification/demiurge/
References

