QBF- and SAT-Based Synthesis from Safety Specs

Robert Könighofer <robert.koenighofer@iaik.tugraz.at>
Institute for Applied Information Processing and Communications
Graz University of Technology, Austria
10/17/2013
Motivation: Synthesis

- Typical design flow:
Motivation: Synthesis

- Typical design flow:
Motivation: From BDDs to SAT

- Challenge: scalability
  - Symbolic algorithms
  - Often implemented with BDDs
    - Known scalability issues

- Enormous achievements in decision procedures
  - SAT-solver, QBF-solvers, EPR-solvers, ...
  - Exploit for synthesis
Outline

- Problem definition
- Learning-based synthesis method
- Template-based synthesis method
- Extensions
- Experimental results
Problem:
Synthesis from Safety Specifications

- “Something bad must never happen”
- Format:

```
Specification = Error Checker
```

Environment: $\mathcal{X}$
System: $\mathcal{T}$
Input: $\mathcal{X}'$
Output: $\mathcal{X}$
Error: $\mathcal{P}$
Typical Synthesis Flow

1. Compute game graph
2. Compute “Winning Region” W
   - Set of states from which the system can win
     - No matter what the environment does
     - Safety: … stay in safe states
3. Compute a strategy
   - What to do in which situation in order to win
     - Safety: stay in winning region
4. Output strategy
   - E.g., as Verilog circuit
Learning-Based Synthesis Method
Supervised Learning

$W^1 = \text{update}(W^0, s_3)$

Is $W^0$ correct?

No! $s_3 \in W^0$ but it should not be.

Is $W^1$ correct?

No! $s_8 \notin W^1$ but it should be.

Is $W^2$ correct?

Yes!

$W^2 = \text{update}(W^1, s_8)$
Learning-Based Method

- $Force^e(A)$
  - the environment can enforce to reach $A$ in one step
Learning-Based Method

- $Force^e(A)$:
  - the environment can enforce to reach $A$ in one step

\[ W := P \]
Learning-Based Method

- \( Force^e(A) \):
  - the environment can enforce to reach A in one step

\[
\begin{align*}
W & := P \\
\text{while}(\text{sat}(W \land Force^e(\neg W))) \{ \\
& \quad \text{pick } s \models W \land Force^e(\neg W) \\
& \} \\
\end{align*}
\]
Learning-Based Method

- $Force^e(A)$:
  - *the environment* can enforce to reach $A$ in one step

$$W := P$$
$$\text{while} (\text{sat}(W \land Force^e(\neg W))) \{$$
  $$\text{pick } s \models W \land Force^e(\neg W)$$
  $$W := W \land \neg s$$
$$\}$$
Learning-Based Method

- \( \text{Force}^e(A) : \)
  - \textit{the environment can enforce to reach } A \textit{ in one step}

\[
W := P \\
\text{while}(\text{sat}(W \land \text{Force}^e(\neg W))) \{ \\
\quad \text{pick } s \models W \land \text{Force}^e(\neg W) \\
\quad W := W \land \neg s \\
\}
\]
Learning-Based Method

- \( \text{Force}^e(A) \):
  - \textit{the environment can enforce to reach } A \textit{ in one step}

\[
\begin{align*}
W & := P \\
\text{while}(\text{sat}(W \land \text{Force}^e(\neg W))) \{ \\
& \quad \text{pick } s \models W \land \text{Force}^e(\neg W) \\
& \quad W := W \land \neg s \\
\} 
\end{align*}
\]
Learning-Based Method

- **Force\(^e\) (A):**
  - *the environment can enforce to reach A in one step*

\[
\begin{align*}
W & := P \\
\text{while} (\text{sat}(W \land \text{Force}^e(\neg W))) \{ \\
\quad \text{pick } s \models W \land \text{Force}^e(\neg W) \\
\quad s & := \text{generalize}(s) \\
\quad W & := W \land \neg s \\
\} \\
\end{align*}
\]
Learning-Based Method

- $Force^e(A)$:
  - the environment `can enforce to reach A in one step`

\[
\begin{align*}
W & := P \\
\text{while}(\text{sat}(W \land Force^e(\neg W))) \{ \\
& \quad \text{pick } s \models W \land Force^e(\neg W) \\
& \quad s := \text{generalize}(s) \\
& \quad W := W \land \neg s
\}
\end{align*}
\]
Learning-Based Method

- $\text{Force}^e(A)$:
  - the environment can enforce to reach $A$ in one step

\[
W := P
\]
\[
\text{while(sat}(W \land \text{Force}^e(\neg W))) \{
\text{pick } s \models W \land \text{Force}^e(\neg W)
\}
\]
\[
s := \text{generalize}(s)
\]
\[
W := W \land \neg s
\]
Learning-Based Method

- \( \text{Force}^e(A) \):
  - the environment can enforce to reach \( A \) in one step

\[
W := P \\
\text{while} (\text{sat}(W \land \text{Force}^e(\neg W))) \{ \\
pick \ s \models W \land \text{Force}^e(\neg W) \\
s := \text{generalize}(s) \\
W := W \land \neg s \\
\} \\
\neg W \rightarrow \text{Force}^e(\neg W)
\]
Template-Based Synthesis Method
Template-Based Method

- Need to find $W(\bar{x})$ such that:
  - $I(\bar{x}) \rightarrow W(\bar{x})$
  - $W(\bar{x}) \rightarrow P(\bar{x})$
  - $W(\bar{x}) \rightarrow Force^s(W(\bar{x}))$

- Let $W(\bar{x}, \bar{k})$ be a parameterized function
  - Concrete values for $\bar{k} \rightarrow$ concrete function $W(\bar{x})$

- Solve: $\exists \bar{k}: I(\bar{x}) \rightarrow W(\bar{x}, \bar{k}) \land$
  
  $W(\bar{x}, \bar{k}) \rightarrow P(\bar{x}) \land$

  $W(\bar{x}, \bar{k}) \rightarrow Force^s(W(\bar{x}, \bar{k}))$
Template-Based Method: CNF Template

\[ W(\bar{x}, \bar{k}) \]

\[ \bigwedge \]

\[ \bigvee \]

\[ x_0 \]
\[ \neg x_0 \]
\[ x_0_{\text{negated}} \]
\[ x_0_{\text{used}} \]

\[ x_1 \]
\[ \neg x_1 \]
\[ x_1_{\text{negated}} \]
\[ x_1_{\text{used}} \]

\[ \cdots \]

\[ x_{07} \]
\[ \neg x_{07} \]
\[ x_{07}_{\text{negated}} \]
\[ x_{07}_{\text{used}} \]

\[ \cdots \]

\[ \text{Parameters } \bar{k} \]

\[ \text{State bits } \bar{x} \]
Extensions

Templates and learning:
- QBF: Pre-processing
  - Extension of Bloqquer to preserve models

Learning-based method:
- SAT-based implementation
- Parallelized implementation
Experimental Results
First Experiments:
AMBA Bus Arbiter

Execution Time [sec]

BDD
IFM’13

Benchmarks
First Experiments:
AMBA Bus Arbiter

Execution Time [sec]

Benchmarks

BDD
IFM'13
Templ
First Experiments:
AMBA Bus Arbiter

Execution Time [sec] vs. Benchmarks

- BDD
- IFM'13
- QBF
- Templ

QBF
First Experiments:
AMBA Bus Arbiter
First Experiments: AMBA Bus Arbiter

Execution Time [sec]

Benchmarks

- BDD
- IFM'13
- QBF
- SAT
- Parallel
- Templ
First Experiments:
Combinational Multiplier
First Experiments: Barrel Shifter

Execution Time [sec]

- BDD
- IFM'13
- QBF
- SAT
- Parallel
- Templ

Benchmarks

- QBF
- BDD
- Templates
- SAT
Parallelization Speedup

![Graph showing parallelization speedup](image)

- 2 Threads
- 3 Threads

1 Thread [sec] vs 2/3 Threads [sec]

- 10 times faster
QBF Preprocessing Speedup:

- Learning-Based
- Template-Based

10 times faster
100 times faster
Conclusions

- No clear winner
  - Different methods are good at different benchmarks
- SAT-based implementation faster than QBF
  - Room for optimization in QBF
- Parallelization is beneficial
  - Different solvers complement each other
- Tool:
  - Open-source release in progress
    - http://www.iaik.tugraz.at/content/research/design_verification/demiurge/