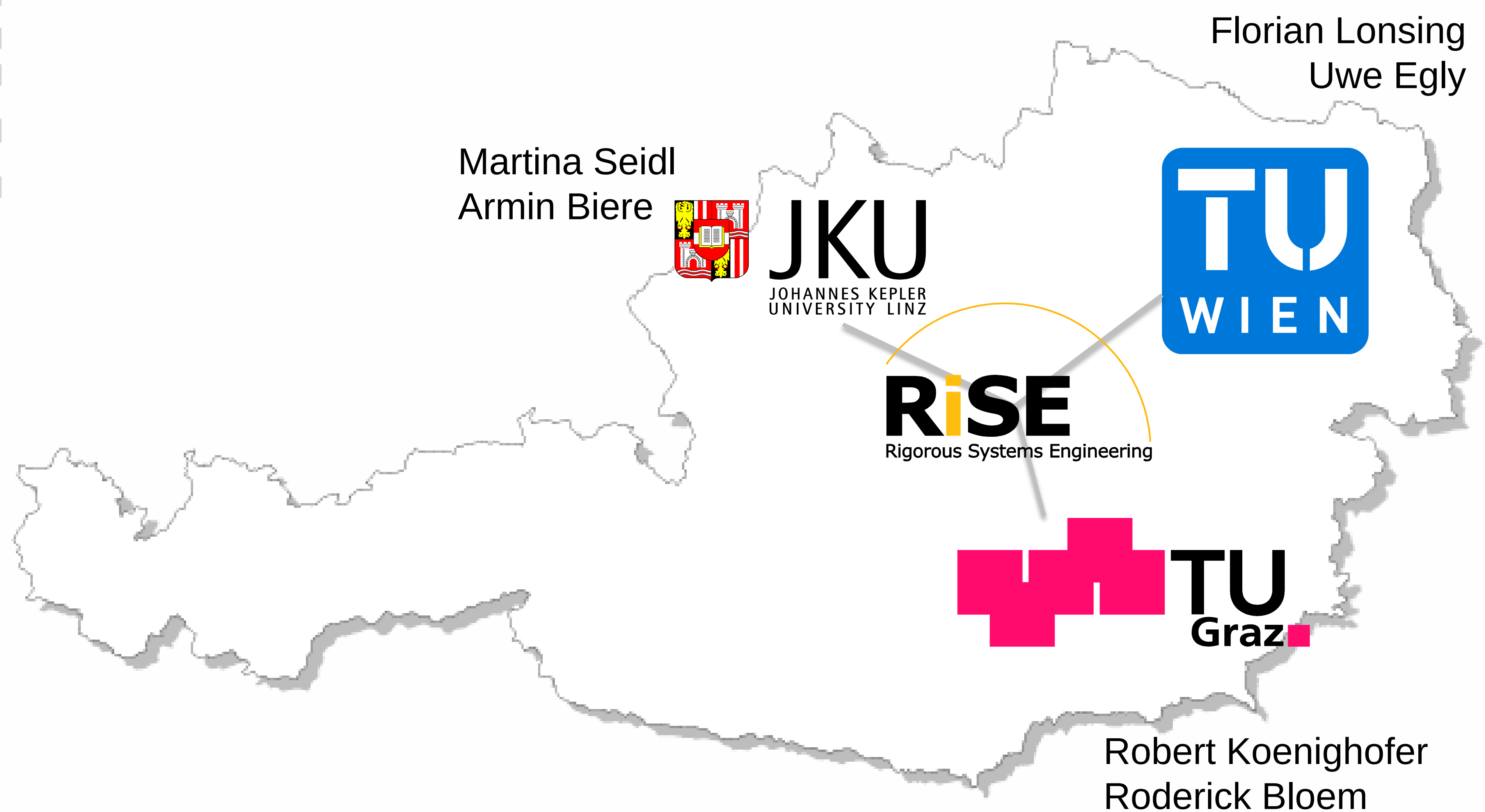
QBF- and SAT-Based Synthesis from Safety Specs

Robert Könighofer <robert.koenighofer@iaik.tugraz.at>
Institute for Applied Information Processing and Communications
Graz University of Technology, Austria

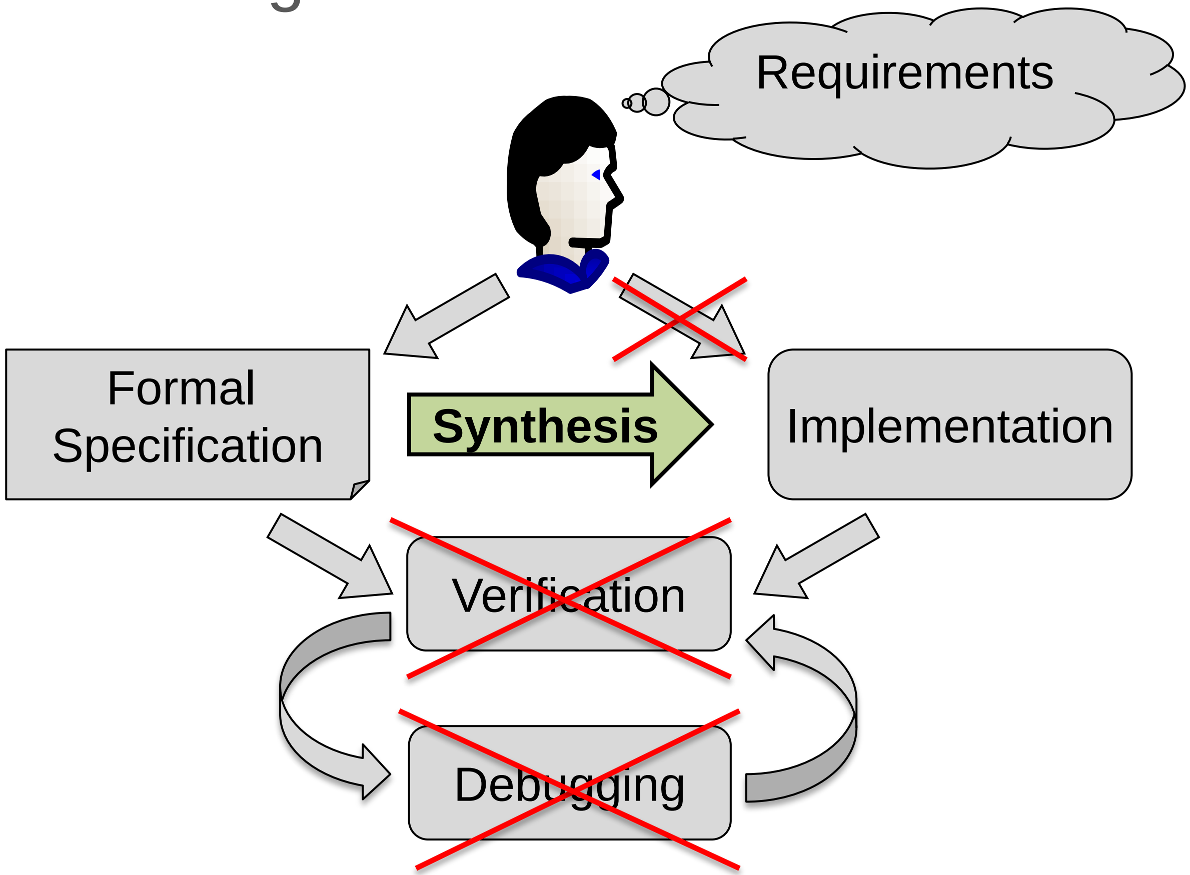
10/17/2013

Rich-Model Toolkit / RiSE Collaboration



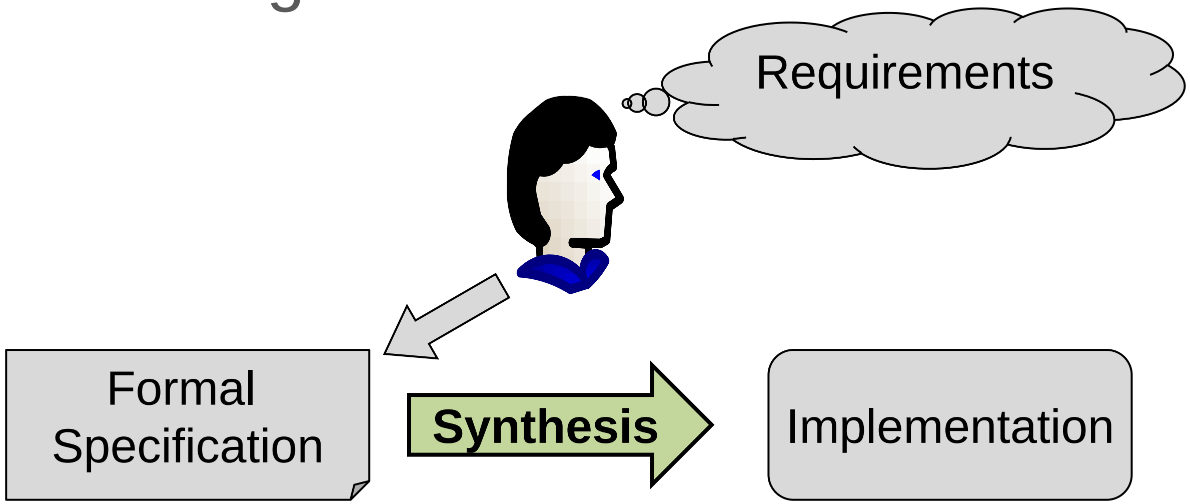
Motivation: Synthesis

- Typical design flow:



Motivation: Synthesis

- Typical design flow:



Motivation: From BDDs to SAT

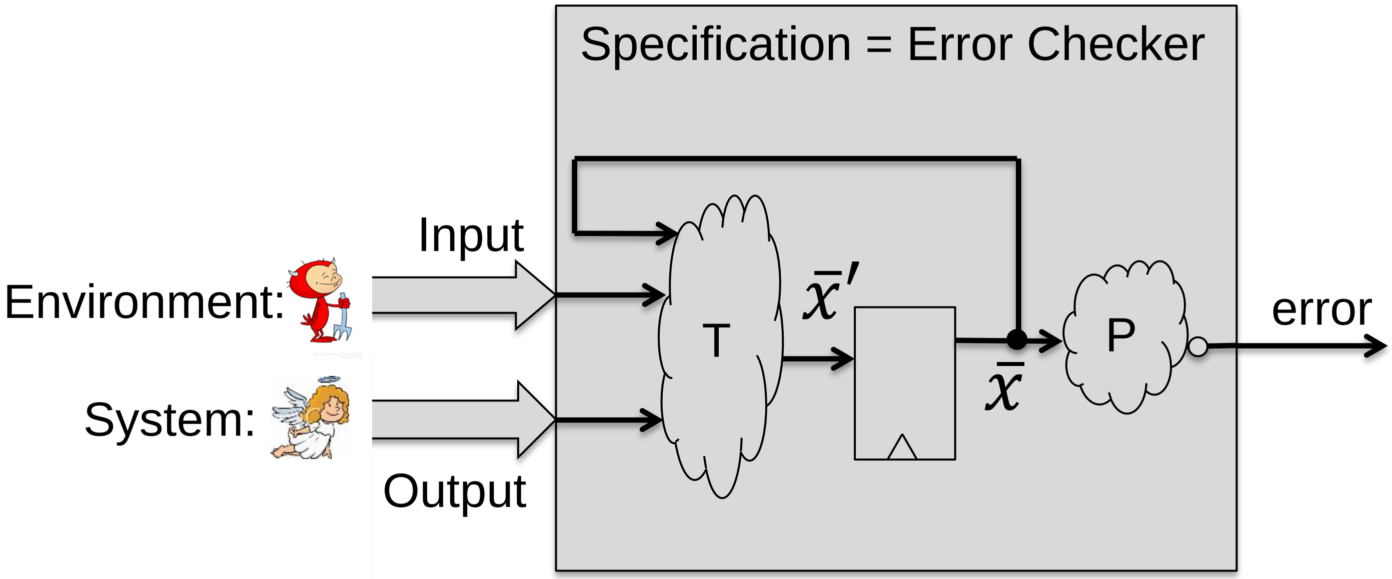
- Challenge: scalability
  - → Symbolic algorithms
  - → Often implemented with BDDs
    - Known scalability issues
- Enormous achievements in decision procedures
  - SAT-solver, QBF-solvers, EPR-solvers, …
  - → Exploit for synthesis

Outline

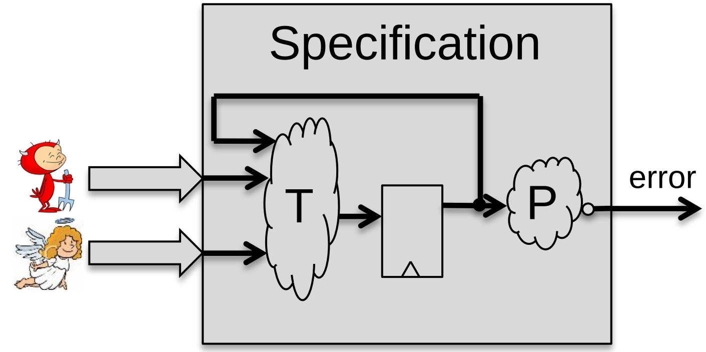
- Problem definition
- Learning-based synthesis method
- Template-based synthesis method
- Extensions
- Experimental results

Problem:
Synthesis from Safety Specifications

- "Something bad must never happen"
- Format:



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

Student

Teacher

Is W^0 correct?

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

$W^1 = update(W^0, s_3)$

Is W^1 correct?

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

$W^2 = update(W^1, s_8)$

Is W^2 correct?

Yes!

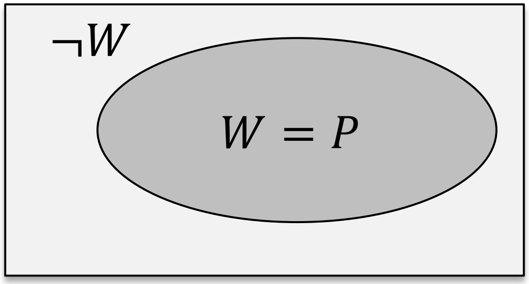
Learning-Based Method

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

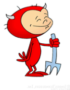
Learning-Based Method

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

$W := P$

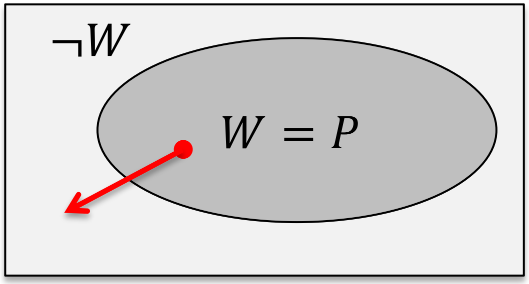


Learning-Based Method

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

$W := P$
while(sat($W \wedge Force^e(\neg W)$)) {
  pick $s \vDash W \wedge Force^e(\neg W)$

}



Learning-Based Method

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

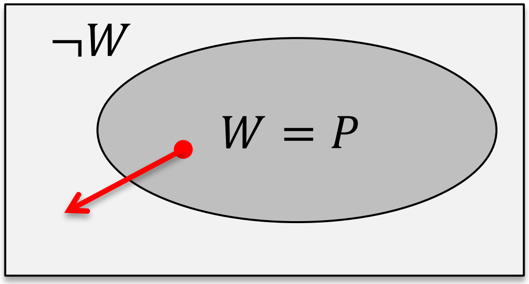
$W := P$

while(sat($W \land Force^{e}(\neg W)$)) {

  pick $s \models W \land Force^{e}(\neg W)$

  $W := W \land \neg s$

}

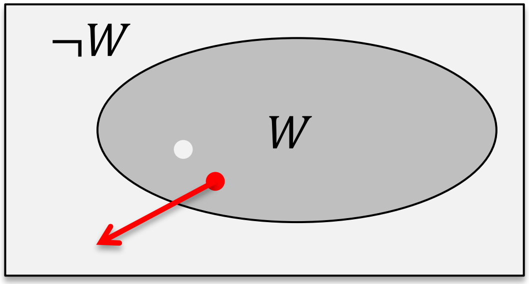


Learning-Based Method

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

$W := P$
while(sat($W \wedge Force^e(\neg W)$)) {
  pick $s \vDash W \wedge Force^e(\neg W)$

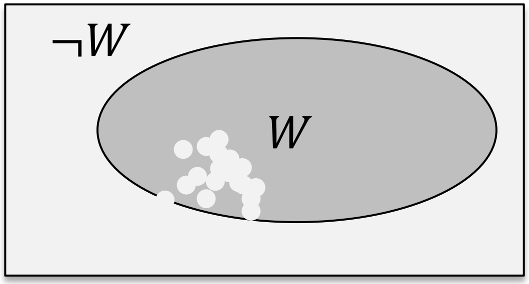
  $W := W \wedge \neg s$
}



Learning-Based Method

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

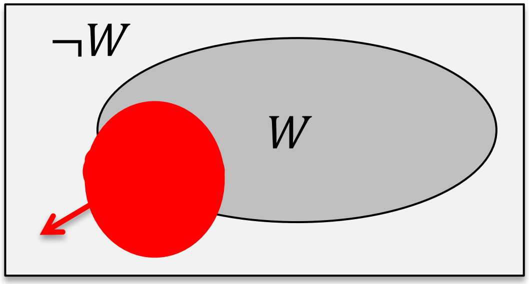
```
W := P
while(sat(W ∧ Force^e(¬W))) {
  pick s ⊨ W ∧ Force^e(¬W)

  W := W ∧ ¬s
}
```



Learning-Based Method

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

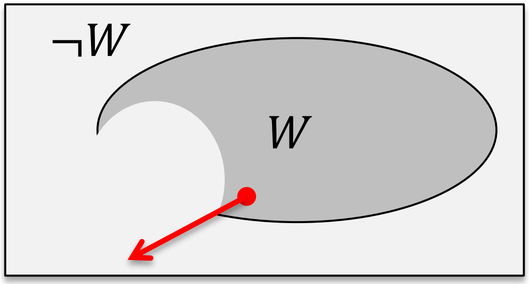
$W := P$
while(sat($W \land Force^e(\neg W)$)) {
  pick $s \vDash W \land Force^e(\neg W)$
  $\mathrm{s} := \mathrm{generalize}(s)$
  $W := W \land \neg s$
}



Learning-Based Method

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

$W := P$
while(sat($W \wedge Force^{e}(\neg W)$)) {
  pick $s \vDash W \wedge Force^{e}(\neg W)$
  s $:=$ generalize(s)
  $W := W \wedge \neg s$
}

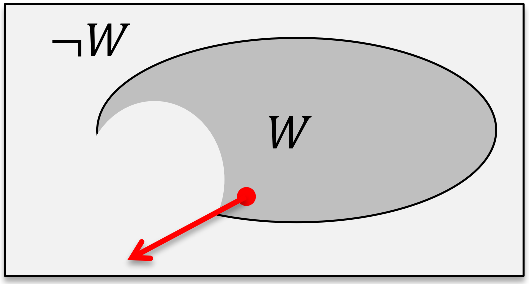


Learning-Based Method

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

$W := P$
while(sat($W \wedge Force^e(\neg W)$)) {
  pick $s \models W \wedge Force^e(\neg W)$
  $s := \text{generalize}(s)$
  $W := W \wedge \neg s$
}

QBF Solver → Satisfying Assignment → $x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge x_4$



Learning-Based Method

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

$W := P$

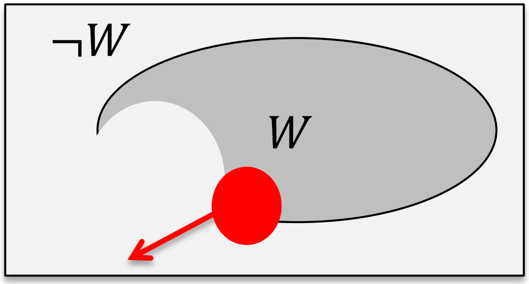
while(sat($W \wedge Force^e(\neg W)$)) {

  pick $s \vDash W \wedge Force^e(\neg W)$

  $s := \text{generalize}(s)$

  $W := W \wedge \neg s$

}



QBF Solver

Satisfying Assignment

$x_1 \wedge \quad x_4$

$\rightarrow Force^e(\neg W)$

Template-Based Synthesis Method

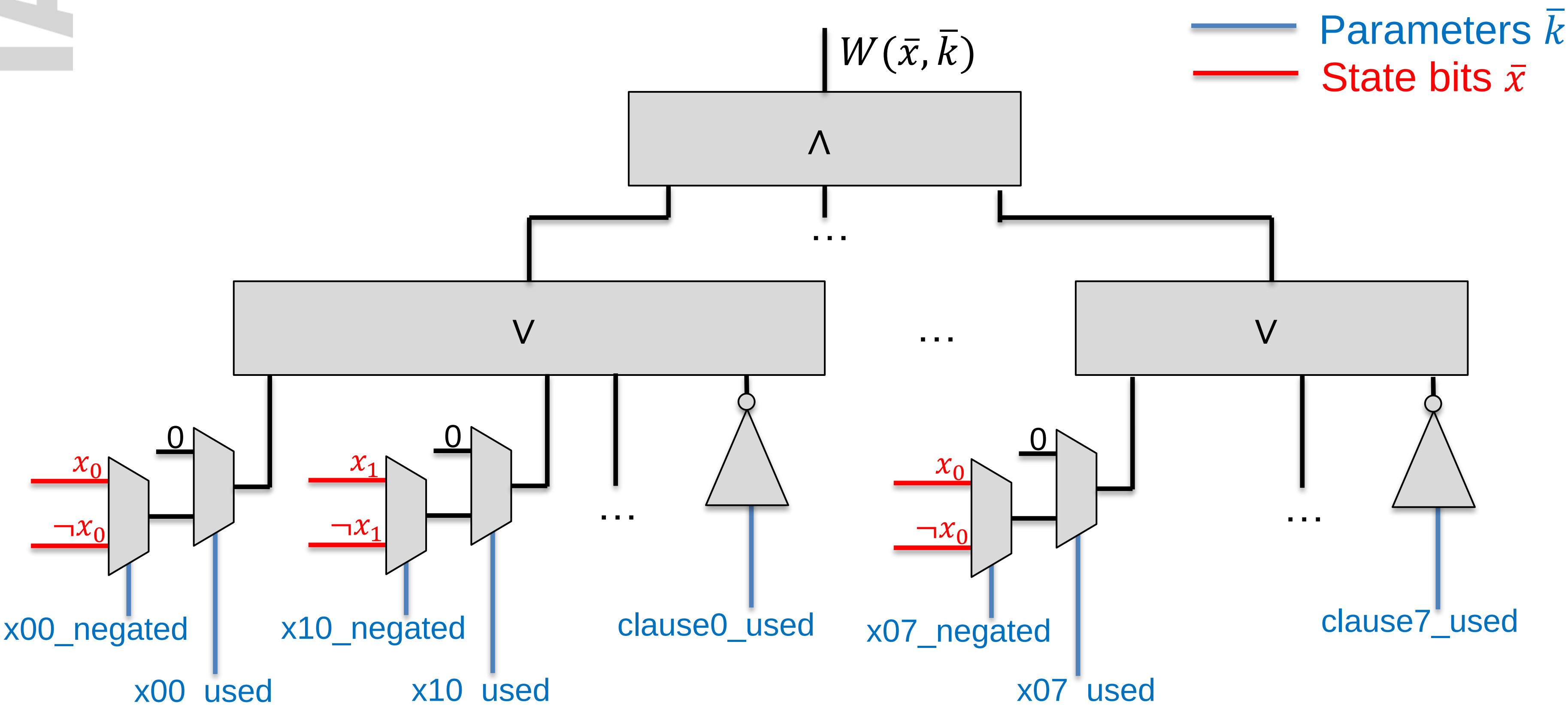
Template-Based Method

- Need to find $W(\bar{x})$ such that:
  - $I(\bar{x}) \rightarrow W(\bar{x})$
  - $W(\bar{x}) \rightarrow P(\bar{x})$
  - $W(\bar{x}) \rightarrow Force^s\big(W(\bar{x})\big)$
- Let $W\big(\bar{x}, \bar{k}\big)$ be a parameterized function
  - Concrete values for $\bar{k}$ → concrete function $W(\bar{x})$
- Solve: $\exists \bar{k}$: $I(\bar{x}) \rightarrow W\big(\bar{x}, \bar{k}\big) \wedge$

$$W\big(\bar{x}, \bar{k}\big) \rightarrow P(\bar{x}) \wedge$$

$$W\big(\bar{x}, \bar{k}\big) \rightarrow Force^s\Big(W\big(\bar{x}, \bar{k}\big)\Big)$$

Template-Based Method:
CNF Template



Extensions

Templates and learning:

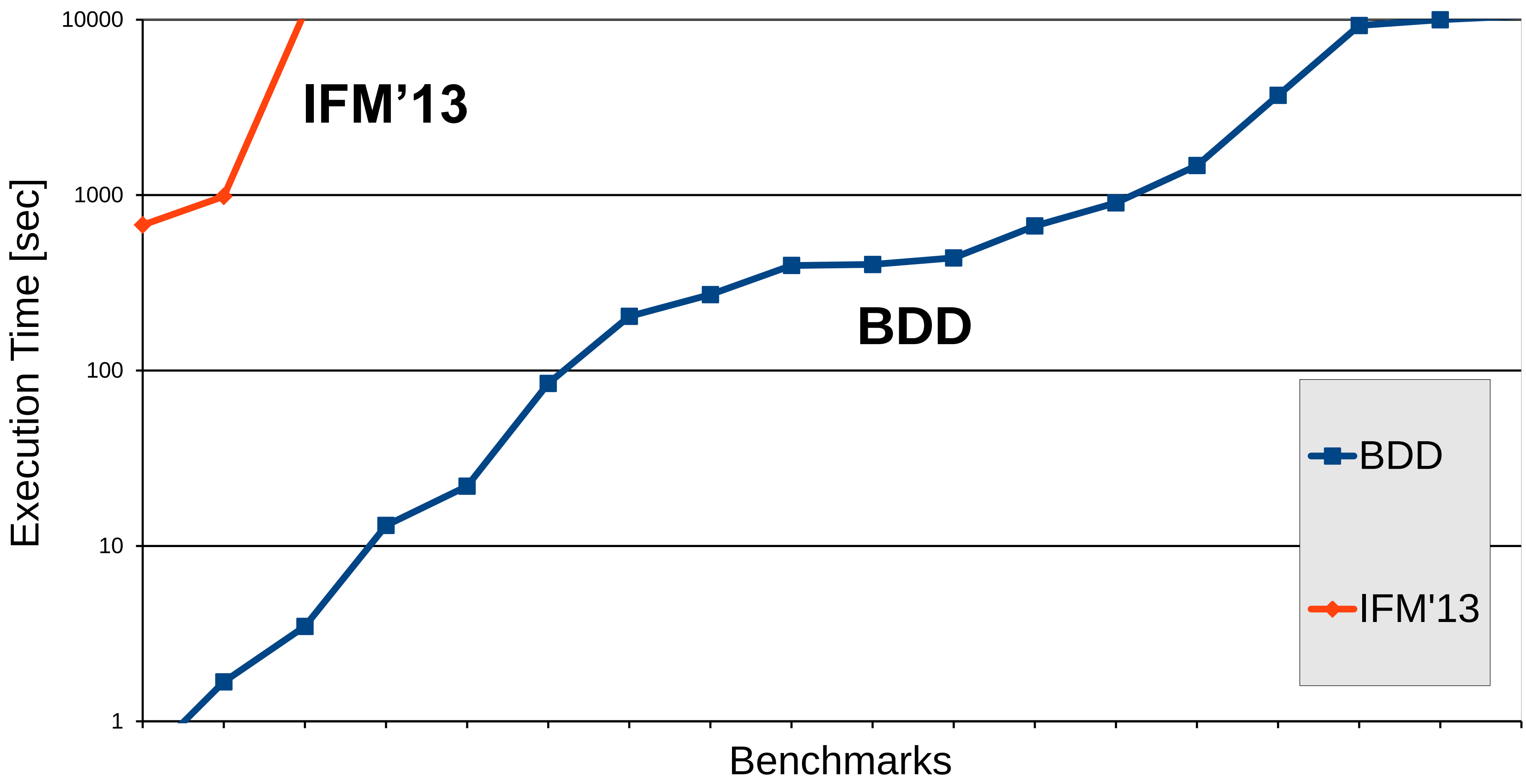
- QBF: Pre-processing
  - Extension of Bloqqer to preserve models

Learning-based method:

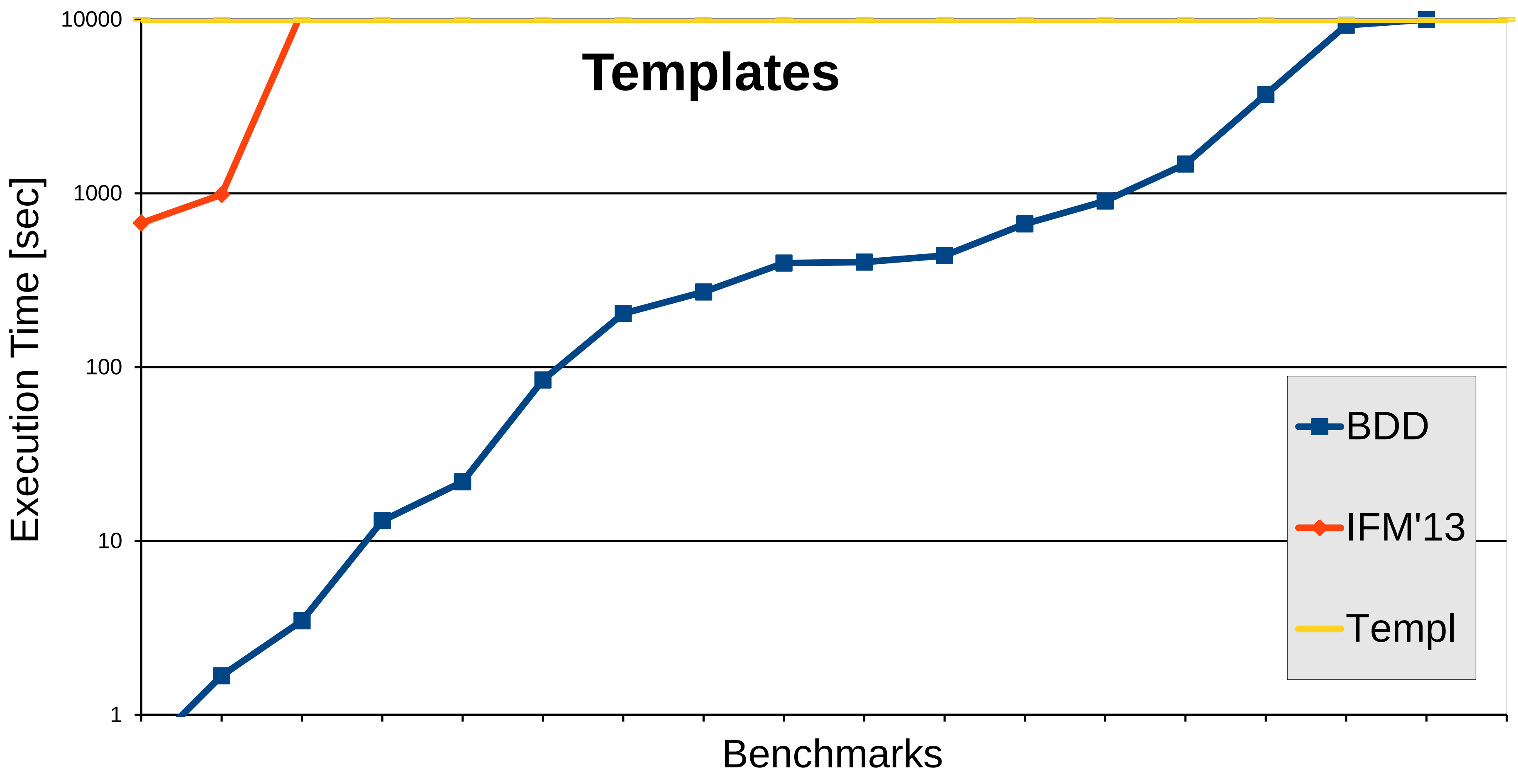
- SAT-based implementation
- Parallelized implementation

Experimental Results

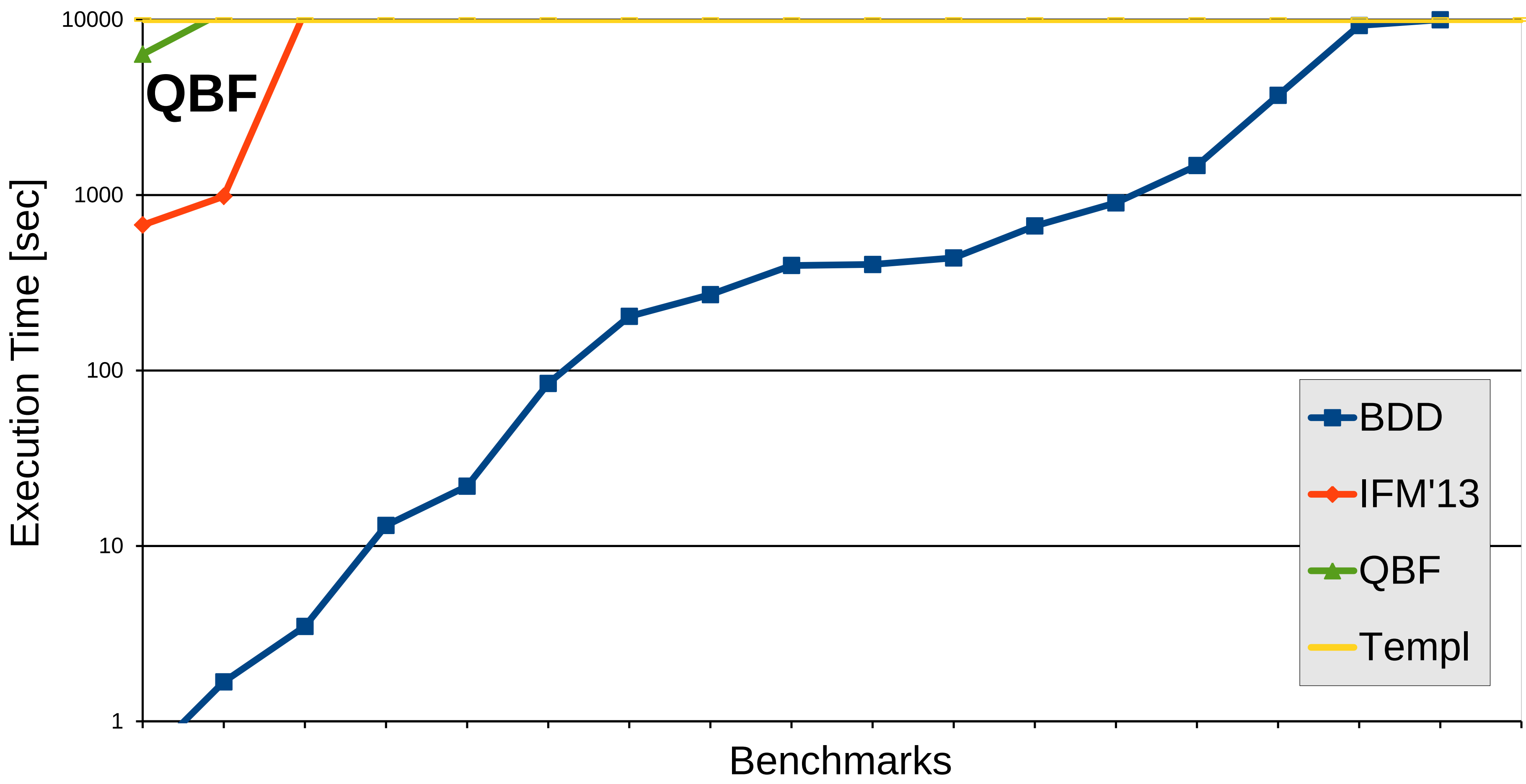
First Experiments:
AMBA Bus Arbiter



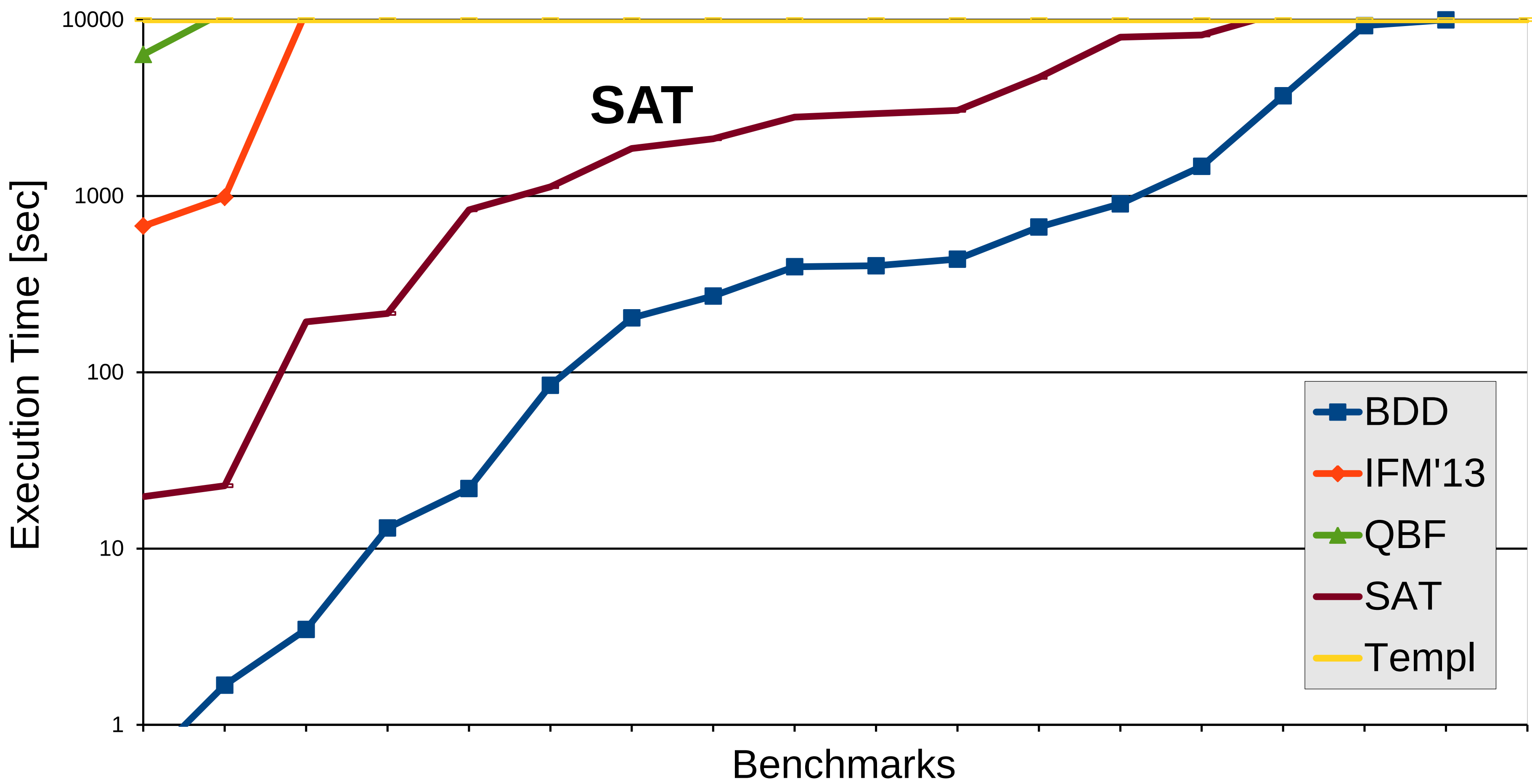
First Experiments:
AMBA Bus Arbiter



First Experiments:
AMBA Bus Arbiter

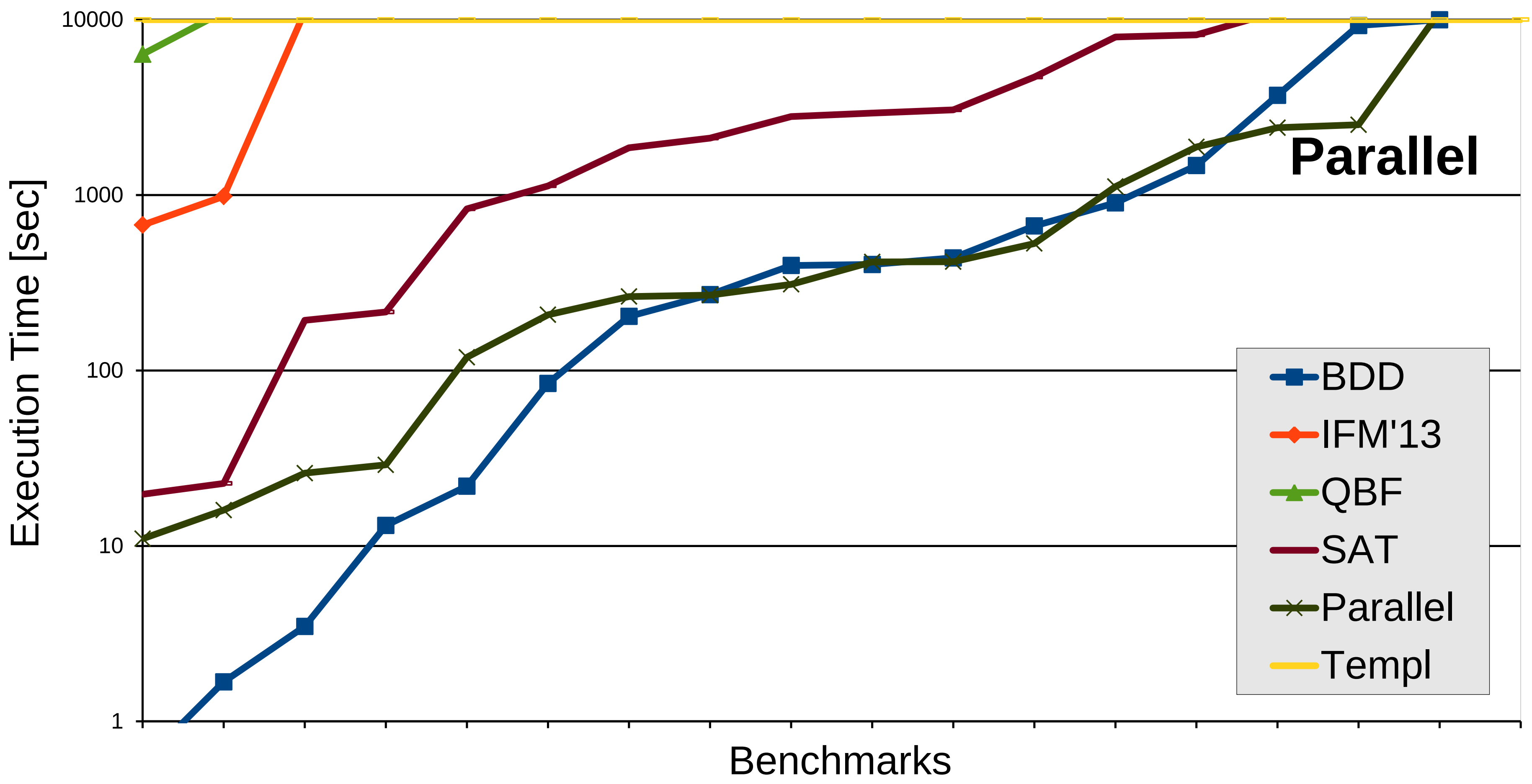


First Experiments:
AMBA Bus Arbiter

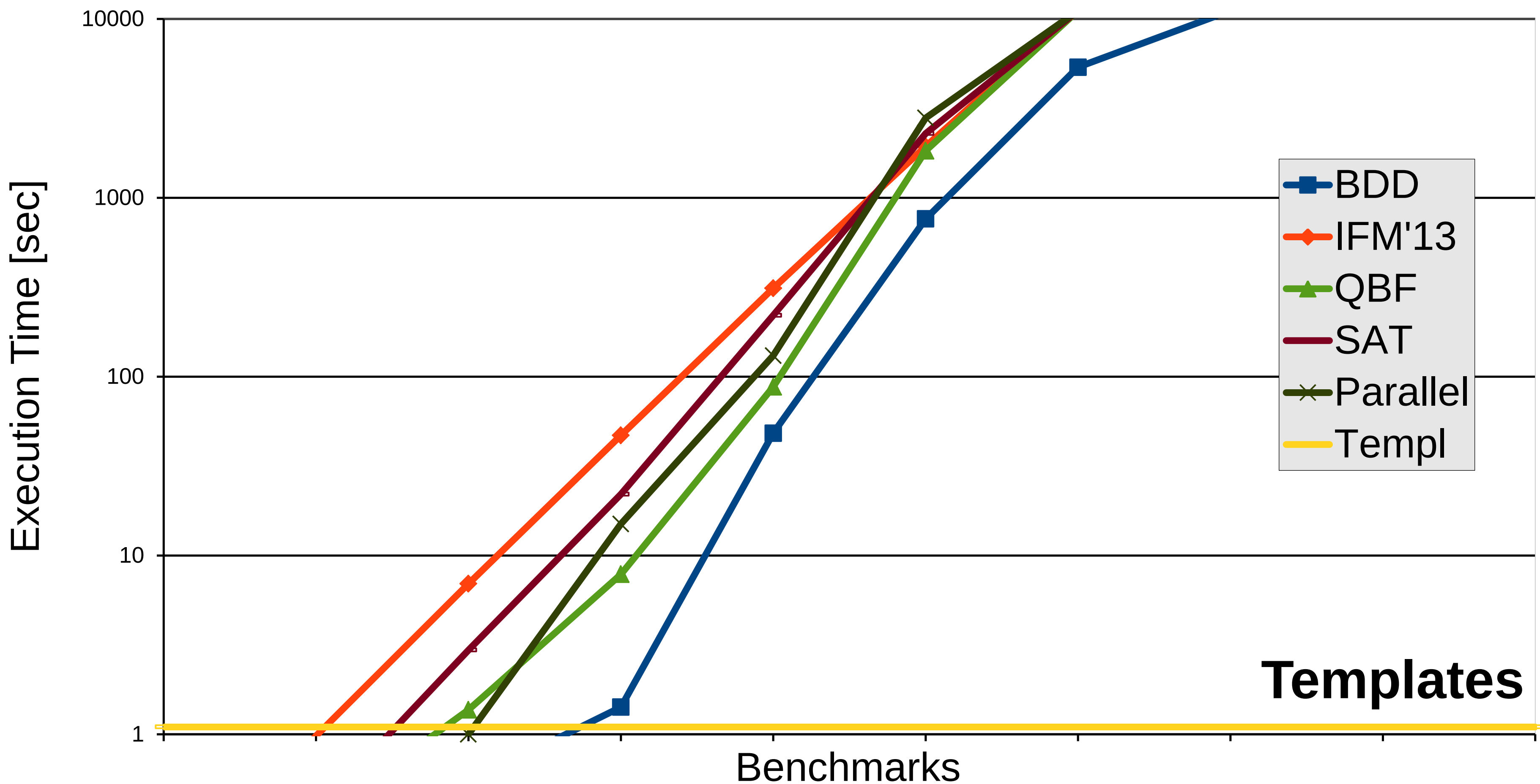


First Experiments:

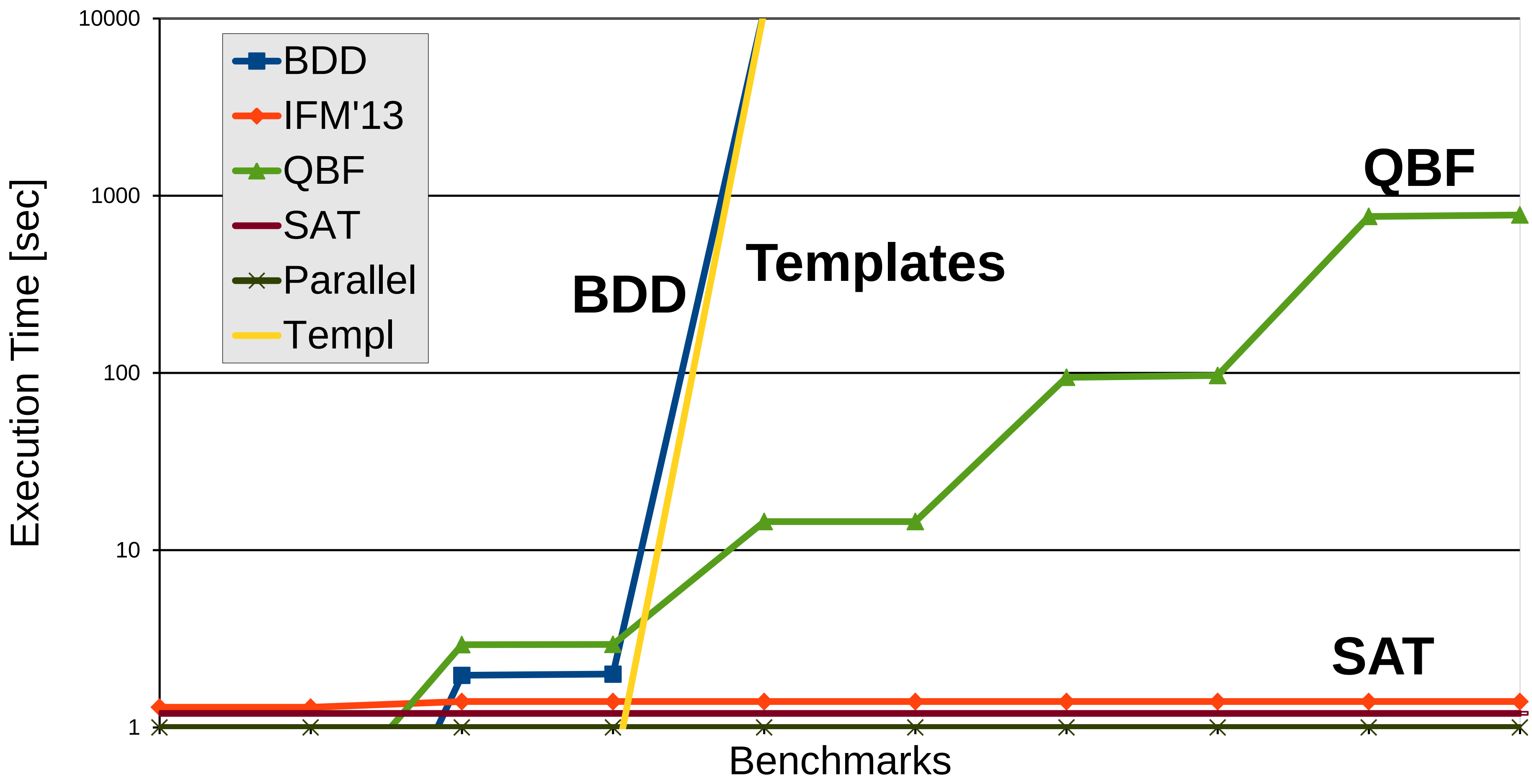
AMBA Bus Arbiter



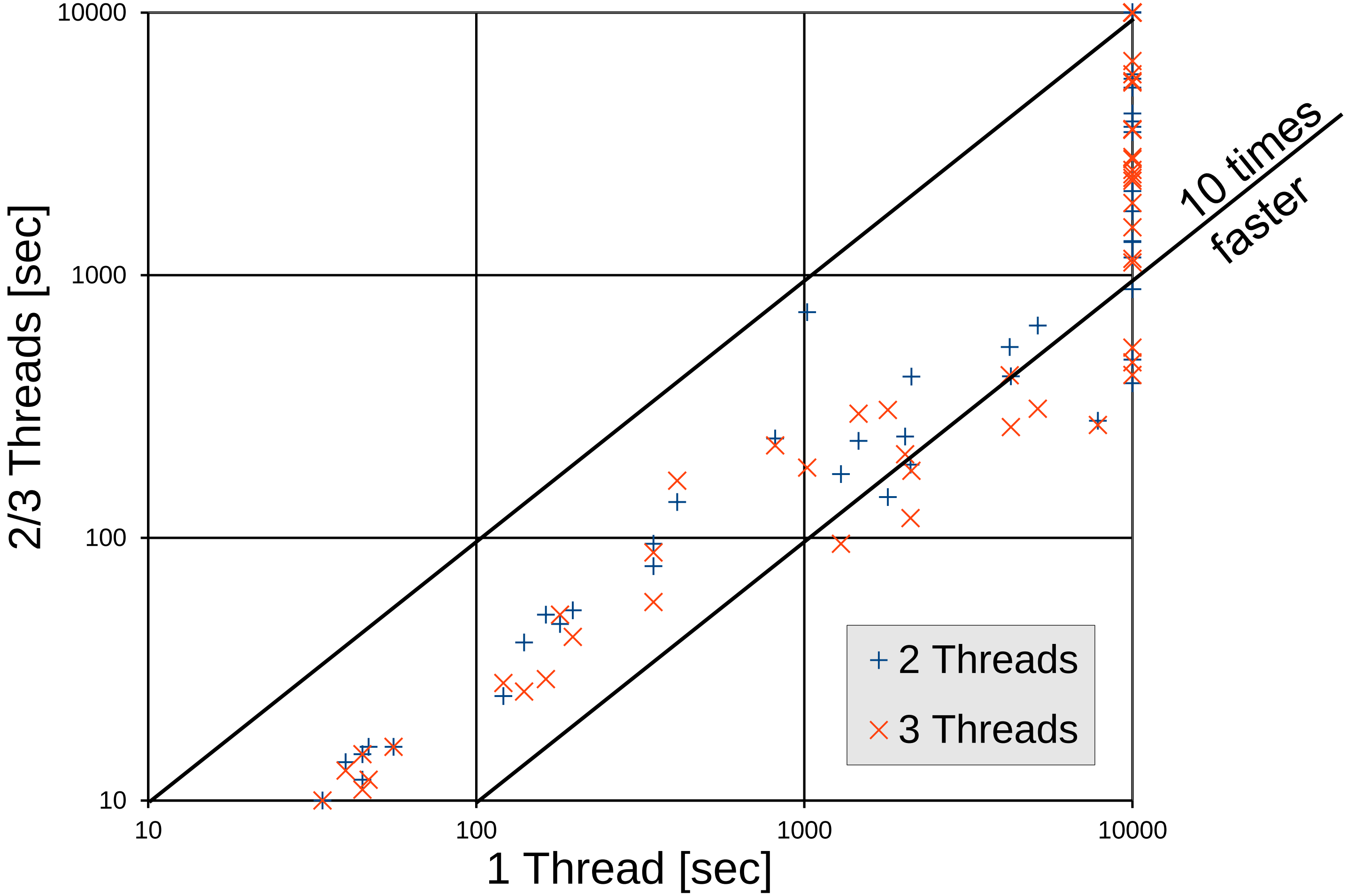
First Experiments:
Combinational Multiplier



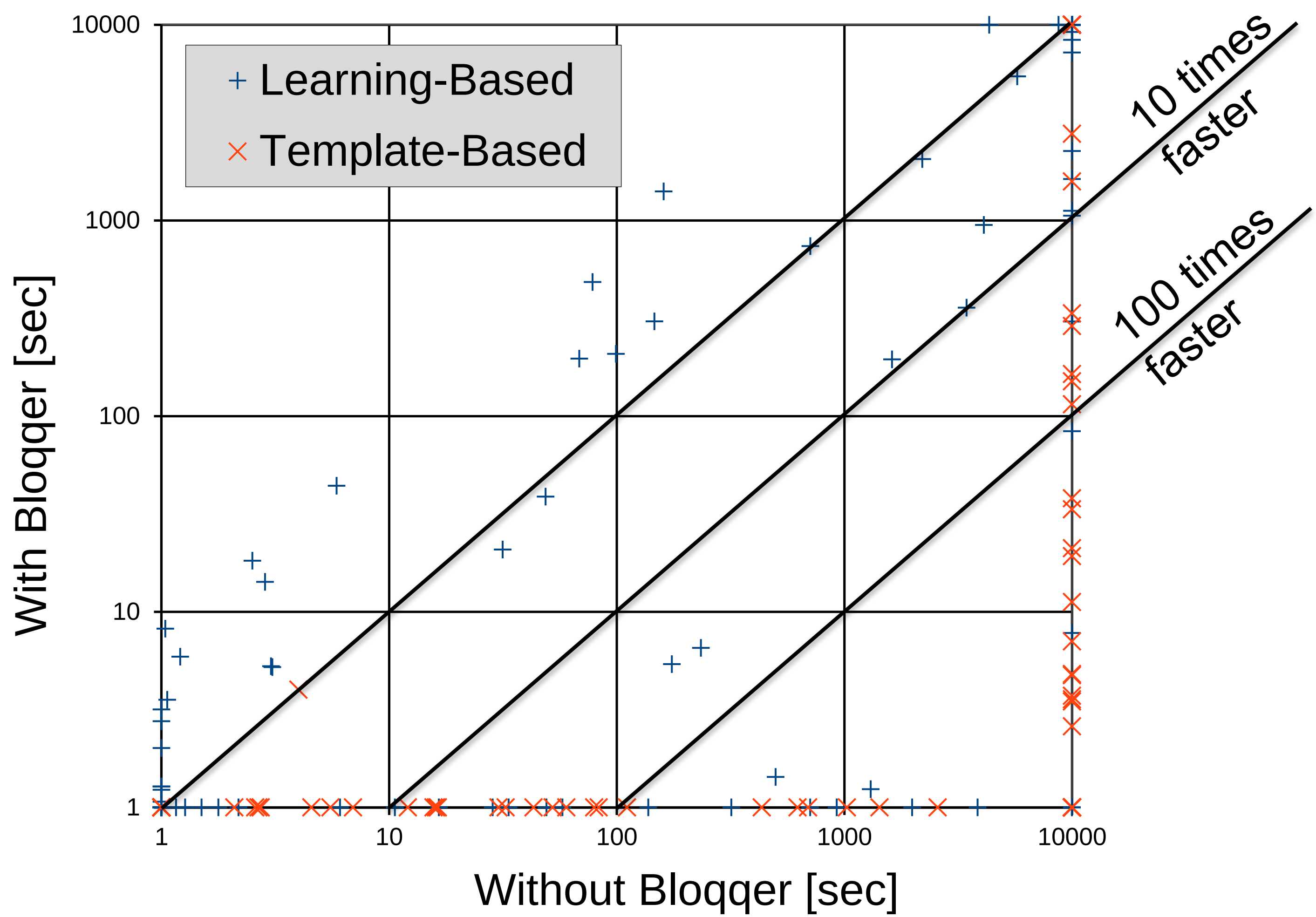
First Experiments:
Barrel Shifter



Parallelization Speedup



QBF Preprocessing Speedup:



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/