Violation Witnesses and Result Validation for Multi-Threaded Programs

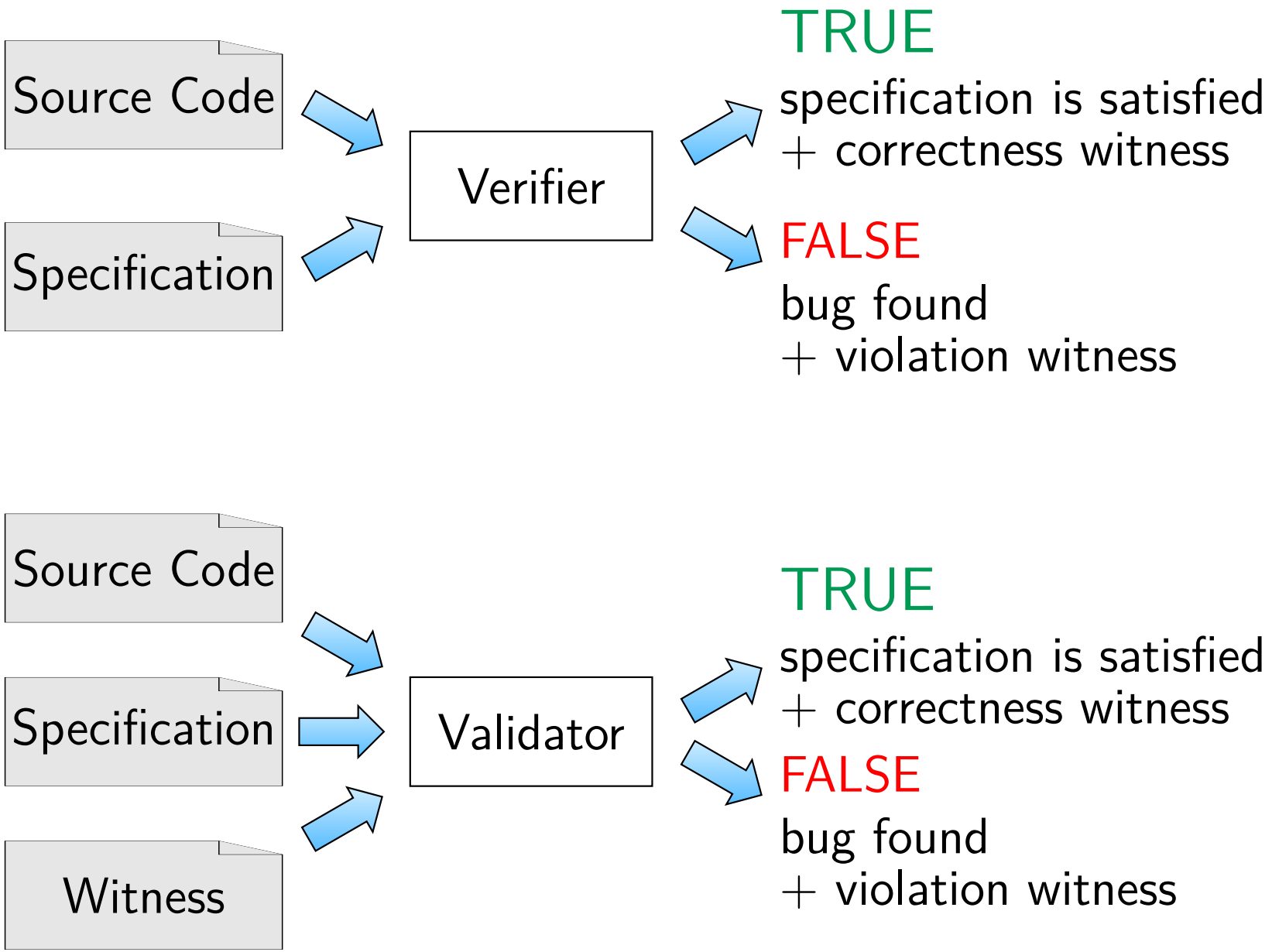
Implementation and Evaluation with CPAchecker

Dirk Beyer and **Karlheinz Friedberger**

LMU Munich, Germany

CPA 2020, online

Software Verification and Validation



Witness Validation

Automaton for guiding the validator

- nodes:
  control states with invariants
- edges:
  transitions with source code information and assumptions
- standardized format: GraphML

```
...
<node id="A19"/>
<node id="A20"/>
<edge source="A19" target="A20">
 <data key="startline">10</data>
 <data key="control">condition-true</data>
 <data key="assumption">k == (0); NUM == (4);</data>
 <data key="assumption.scope">t1</data>
</edge>
...
```

Problems

SV-COMP requires witness validation for more tasks

- including concurrent tasks

1) Witness format not suitable for concurrent programs

- no information about threads available

2) Witness validator for concurrent tasks not available

- only validators for sequential programs

Witness Validation for Concurrent Programs

Correctness Witnesses

- unbounded number of threads
- invariants over different threads

  $\rightarrow$ quantifiers? scope?

Violation Witnesses

- counterexample: fixed number of statements, no loops

  $\rightarrow$ limited thread interleavings
- information about thread interleaving required

Solution

1) Extension of the witness format
   - What is the current thread?
   - Where does a new thread starts?

2) CPAchecker as witness validation for concurrent tasks
   - Based on already existing components
   - Minimal development overhead for CPAchecker

Evaluation
   - Which tools provide sufficient witnesses?
   - How well does CPAchecker perform for validation?

Concurrent Programs with Pthreads

Pthreads and Locks

- *pthread_create*, *pthread_join*, mutex locks
- atomic statements and atomic sequences

What is *important* for a validator?

- guidance through the state space!

  $\rightarrow$ thread interleaving along the counterexample

What is *not important* for a validator?

- already handled by the underlying analysis

  $\rightarrow$ mutex locks, atomic statements

Witnesses for Concurrent Programs

Extension: information about thread interleaving

- What is the current thread?
  $\rightarrow$ $threadId$ for every transition
- Where does a new thread starts?
  $\rightarrow$ $threadCreate$ for introducing a new thread

```
<edge source="A15" target="sink">
 <data key="threadId">0</data>
 <data key="createThread">2</data>
 <data key="startline">26</data>
</edge>

<edge source="A19" target="A20">
 <data key="threadId">1</data>
 <data key="startline">10</data>
</edge>
```

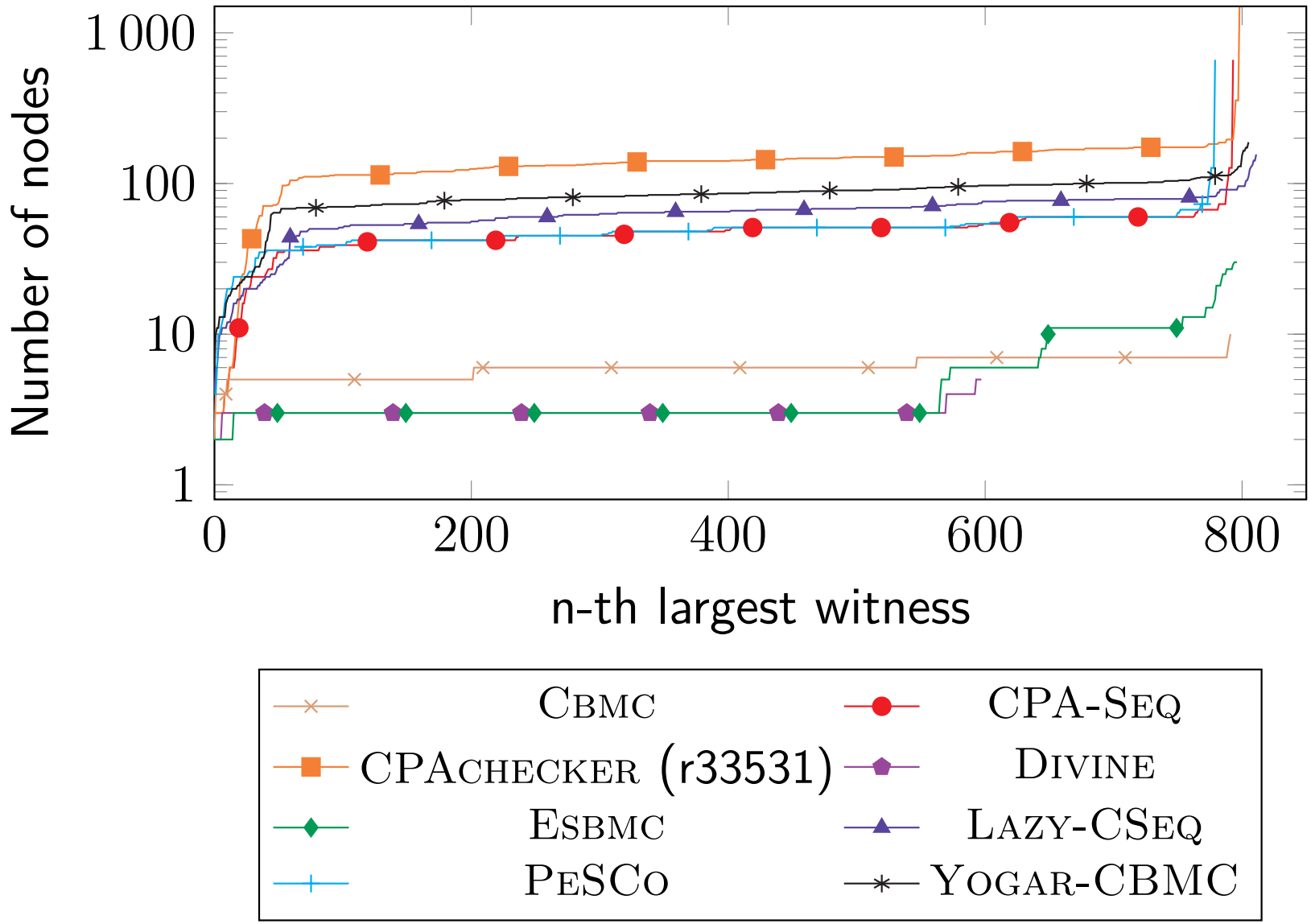
Evaluation

- Tools
  - CPAchecker `r33531`: ThreadingCPA with BDD analysis
  - several participants of SV-COMP'19
- Environment
  - Intel Xeon E3-1230 v5 CPU
  - over 1000 tasks (concurrency set from SV-COMP)
  - Limitations: 15 GB RAM and 15 minutes

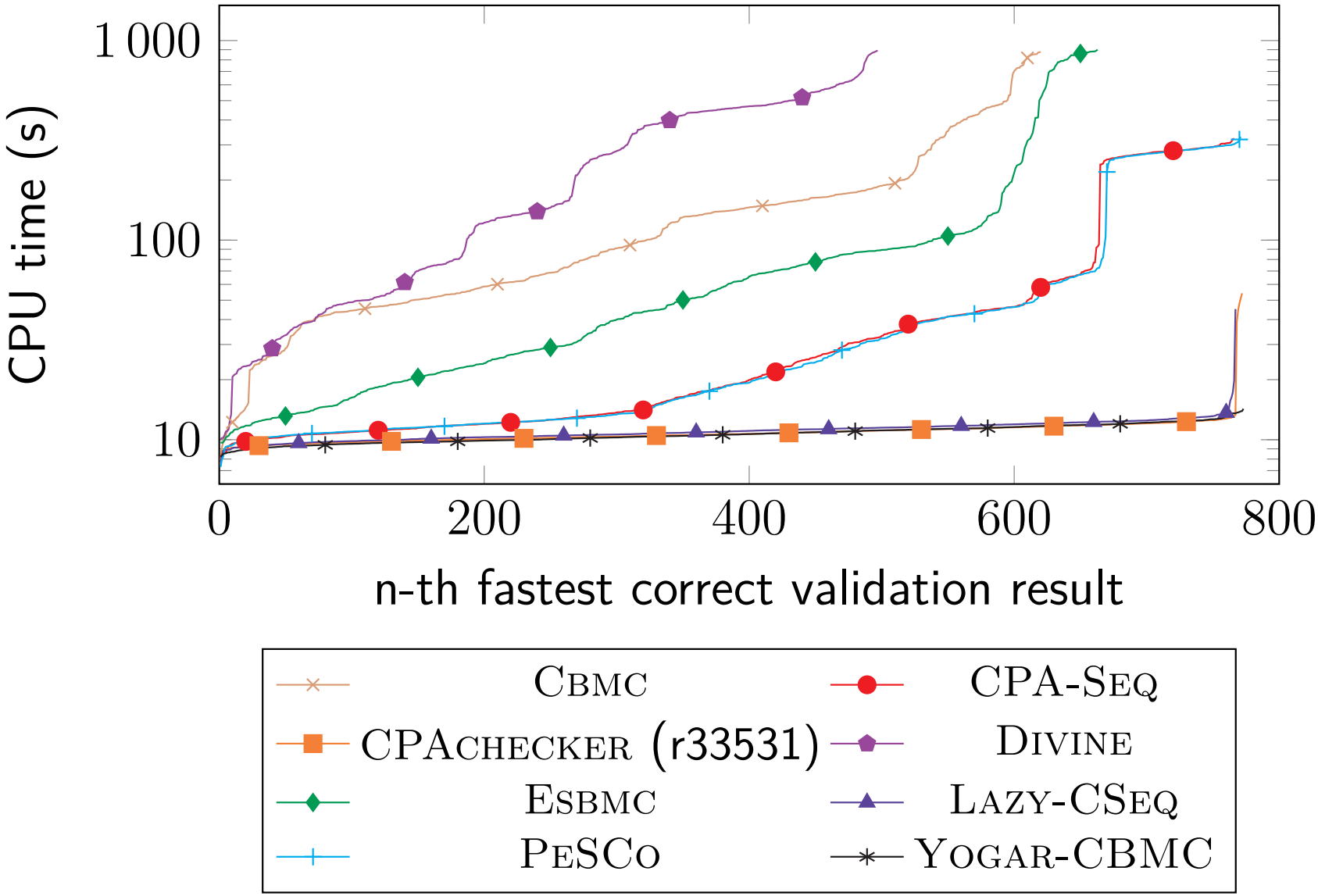
Evaluation: Tools and Features

Verifier	thread id	thread creation	all thread interleavings
Cbmc		✓	
CPA-Seq	✓	✓	
CPAchecker (r33531)	✓	✓	✓
Divine			
Esbmc			
Lazy-CSeq	✓	✓	✓
PeSCo	✓	✓	
Yogar-CBMC	✓	✓	✓

Evaluation: Witnesses from Tools



Evaluation: Validation with CPAchecker



Conclusion

- Witness format is extended with threading information
- CPAchecker sucessfully produces and validates violation witness for concurrent programs
- Some other tools could improve their support for witnesses

Future work

SMT-based analysis for concurrent programs

- improved pointer analysis

Optimization

- shrink witnesses to only relevant information

Encode more properties into witnesses

- deadlocks: should be possible. benchmarks missing
- data races: ?

References

Dirk Beyer and Karlheinz Friedberger:
Violation Witnesses and Result Validation for Multi-Threaded Programs, ISoLA 2020, online