Violation Witnesses and Result Validation for Multi-threaded Programs

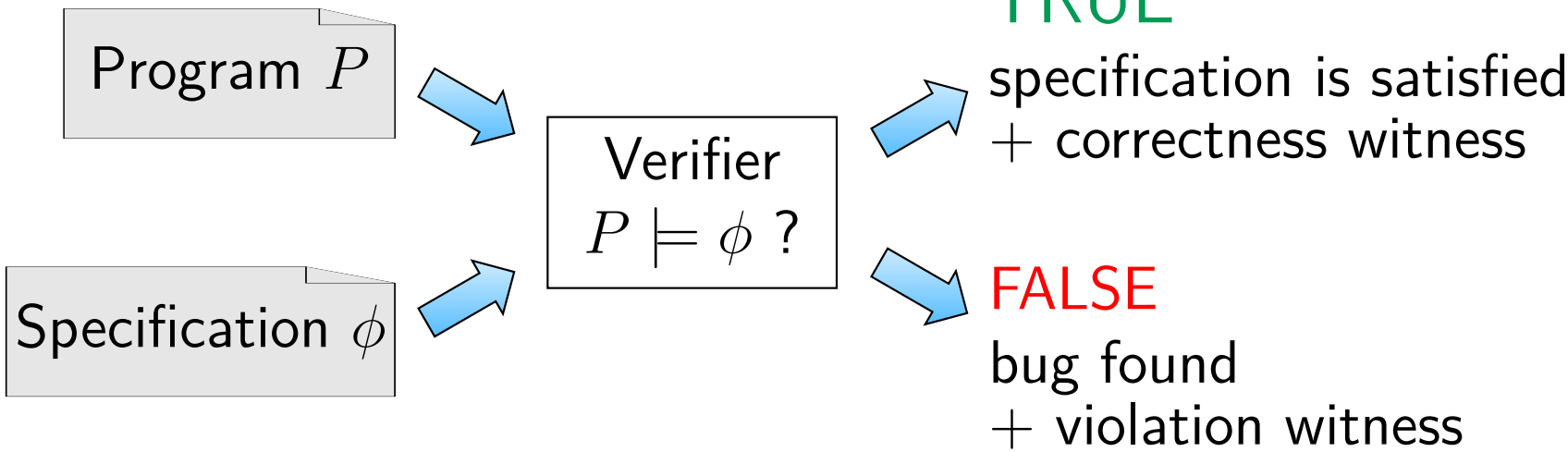
Implementation and Evaluation with CPAchecker

Dirk Beyer and Karlheinz Friedberger

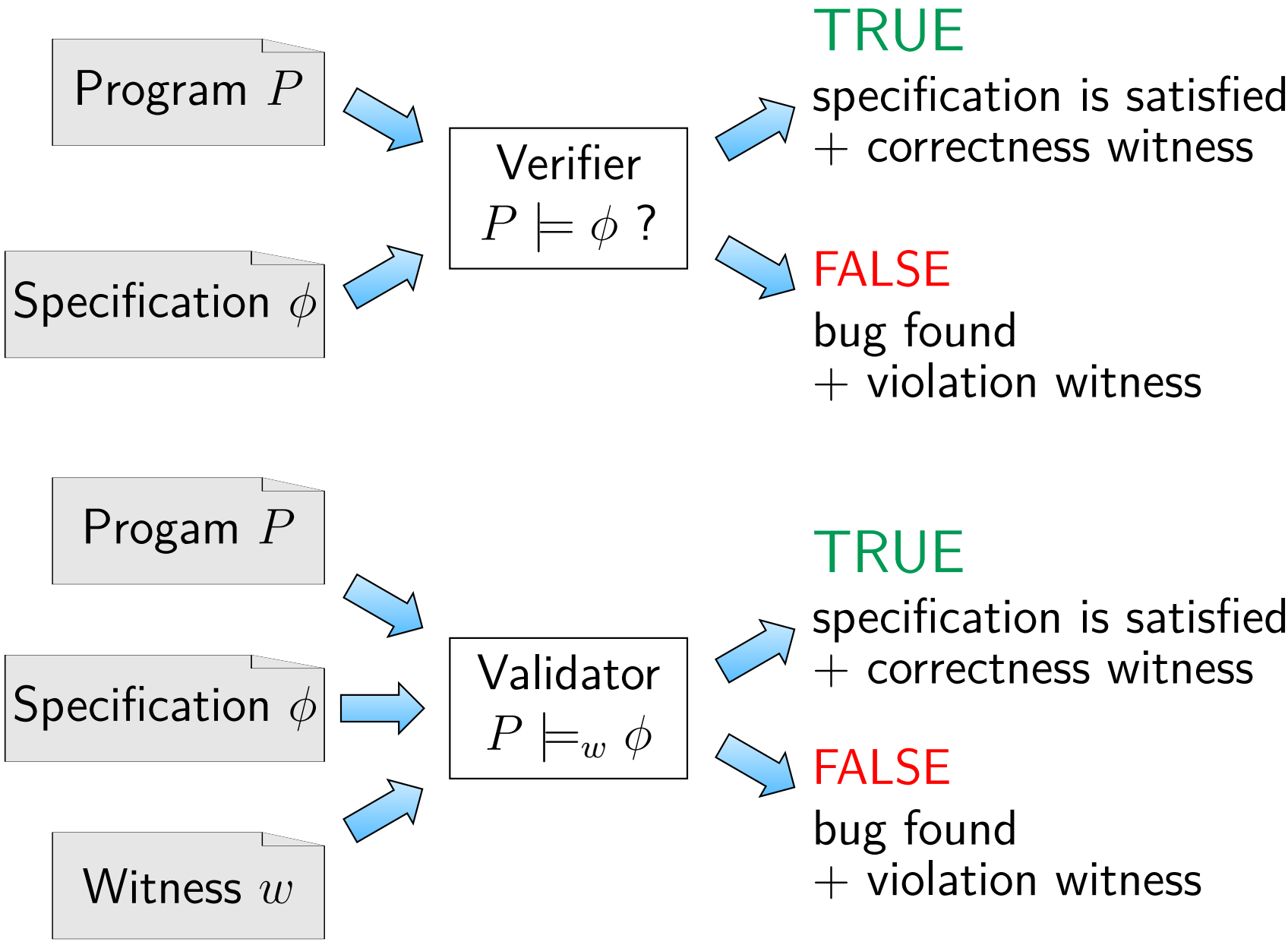
LMU Munich, Germany

ISoLA 2021

Software Verification and Validation



Software Verification and Validation



Why Witnesses?

- Result validation [2, 1, 3]
  (eliminate wrong results, increase trust)
- Interface for component-based verifier construction
  (exchange of information for cooperative verification)
- Decomposition of verification into components
  (for example, a CEGAR verifier can be constructed from off-the-shelf components)

Advance Summary

- There was no validator for multi-threaded programs
- Implemented a validator for multi-threaded programs
- In the following more details

Witness-Based Result Validation

Witness can be an 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

Results validation is required to eliminate wrong results

- including multi-threaded (MT) tasks

Problems

Results validation is required to eliminate wrong results

- including multi-threaded (MT) tasks

1) Witness format was previously not suitable for MT

- format had no information about threads available

2) No validators for multi-threaded tasks were available

- only validators for sequential programs were available

Witness Validation for Concurrent Programs

Correctness Witnesses

- unbounded number of threads
- invariants over different threads

Witness Validation for Concurrent Programs

Correctness Witnesses

- unbounded number of threads
- invariants over different threads

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 result validator for concurrent tasks
   - Based on already existing components
   - Minimal development overhead for CPAchecker
   - For violation witnesses only

Solution

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

2) CPAchecker as result validator for concurrent tasks
   - Based on already existing components
   - Minimal development overhead for CPAchecker
   - For violation witnesses only

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

Concurrent Programs with Pthreads

Pthreads and Locks

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

What is *important* for a validator?

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

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?

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>
```

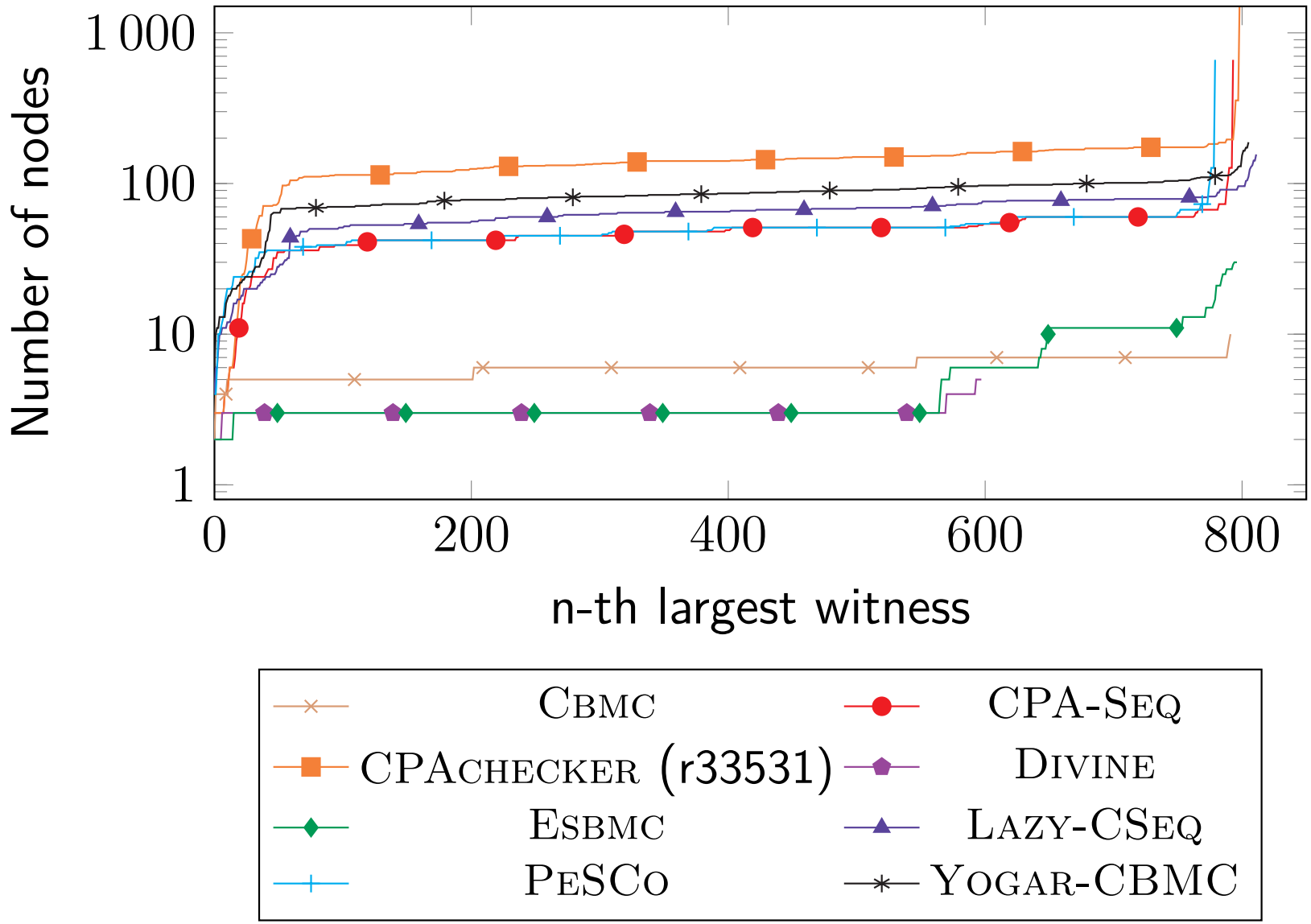
Evaluation

- Tools
  - CPAchecker `r33531`: ThreadingCPA with BDD analysis
  - several participants of SV-COMP 2019
    Cbmc, CPA-Seq, Divine, Esbmc, Lazy-CSeq, PeSCo, Yogar-CBMC
- Environment
  - Intel Xeon E3-1230 v5 CPU
  - over 1000 tasks (concurrency set from SV-COMP)
  - Limitations: 15 GB RAM and 15 minutes

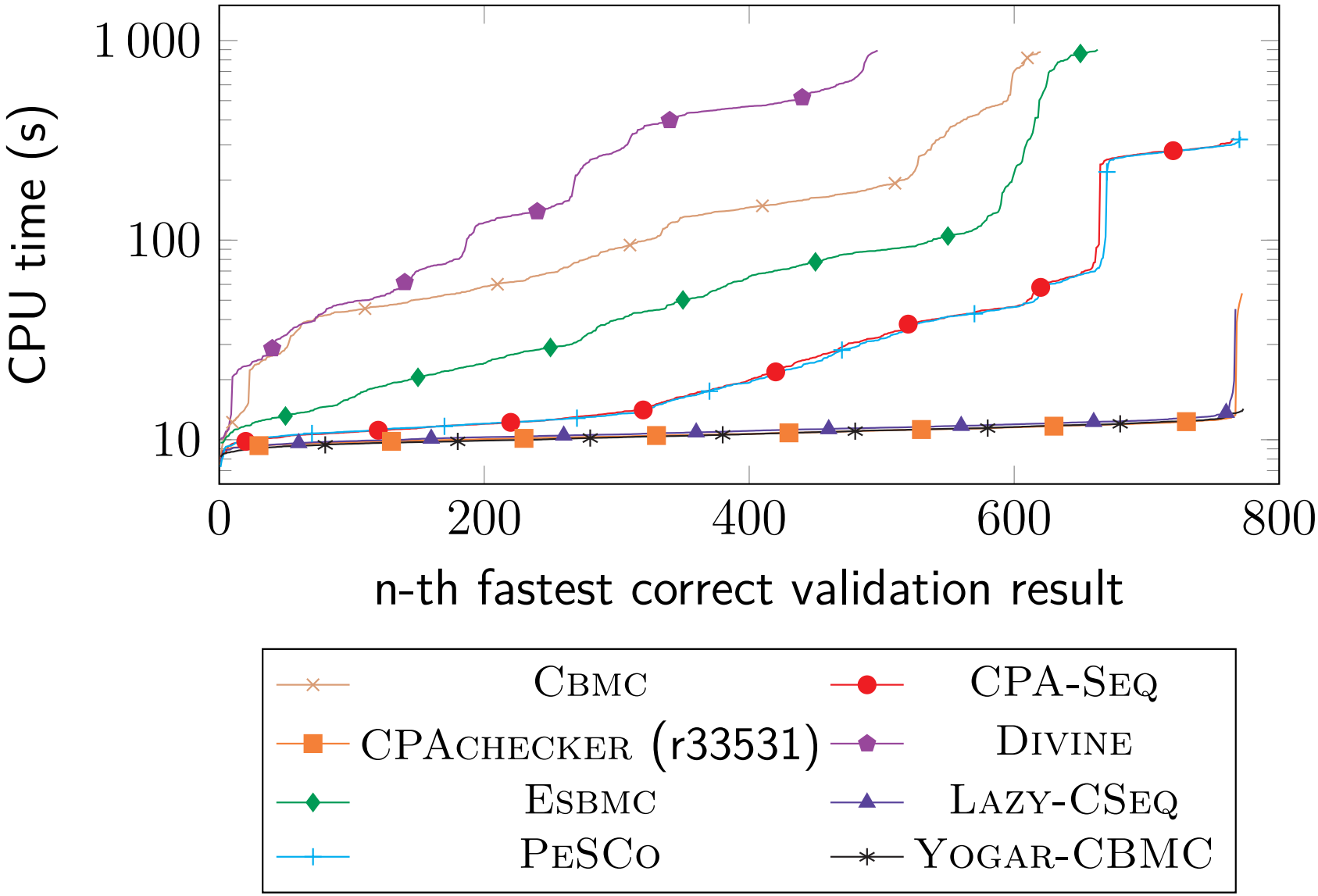
Evaluation: Tools and Features

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

Evaluation: Verifier Performance



Evaluation: Validator Performance (CPAchecker)



Conclusion

- Witness format is extended with threading information

Conclusion

- Witness format is extended with threading information
- CPAchecker sucessfully produces and validates verification results for concurrent programs

Conclusion

- Witness format is extended with threading information
- CPAchecker sucessfully produces and validates verification results for concurrent programs
- New validator in SV-COMP 2022: Dartagnan

Reference: Proc. ISoLA 2020 [4]

Future Work

SMT-based analysis for concurrent programs

- improved pointer analysis

Optimization

- shrink witnesses to only relevant information

Encode more properties into witnesses

- deadlocks: possible, but benchmark programs missing
- data races: ongoing effort in SV-COMP community

References I

[1] Beyer, D., Dangl, M., Dietsch, D., Heizmann, M.: Correctness witnesses: Exchanging verification results between verifiers. In: Proc. FSE. pp. 326–337. ACM (2016). doi:10.1145/2950290.2950351

[2] Beyer, D., Dangl, M., Dietsch, D., Heizmann, M., Stahlbauer, A.: Witness validation and stepwise testification across software verifiers. In: Proc. FSE. pp. 721–733. ACM (2015). doi:10.1145/2786805.2786867

[3] Beyer, D., Dangl, M., Lemberger, T., Tautschnig, M.: Tests from witnesses: Execution-based validation of verification results. In: Proc. TAP. pp. 3–23. LNCS 10889, Springer (2018). doi:10.1007/978-3-319-92994-1_1

[4] Beyer, D., Friedberger, K.: Violation witnesses and result validation for multi-threaded programs. In: Proc. ISoLA (1). pp. 449–470. LNCS 12476, Springer (2020). doi:10.1007/978-3-030-61362-4_26