Efficient Software Model Checking with Block-Abstraction Memoization

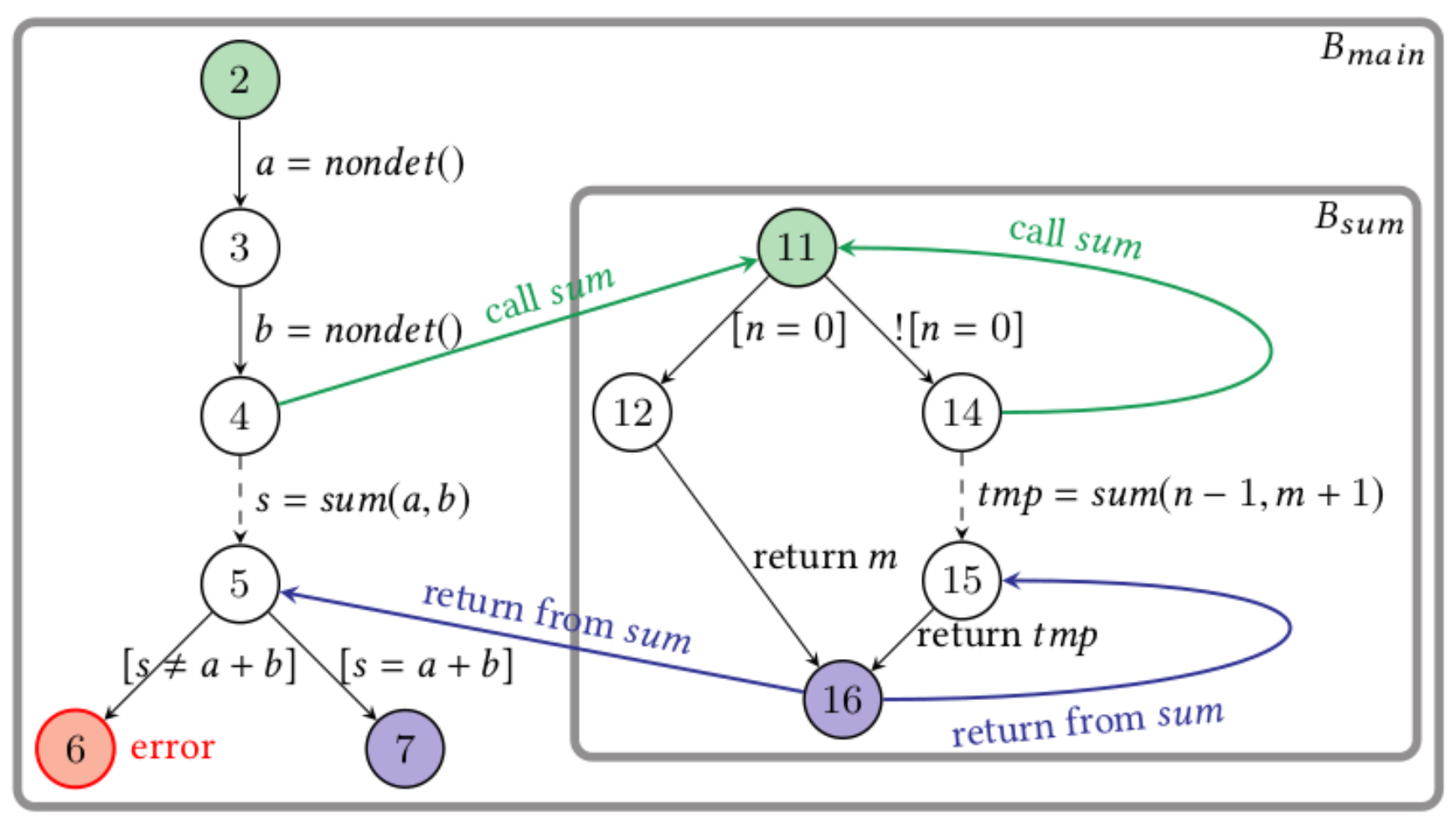
Karlheinz Friedberger
karlheinz.friedberger@ifi.lmu.de

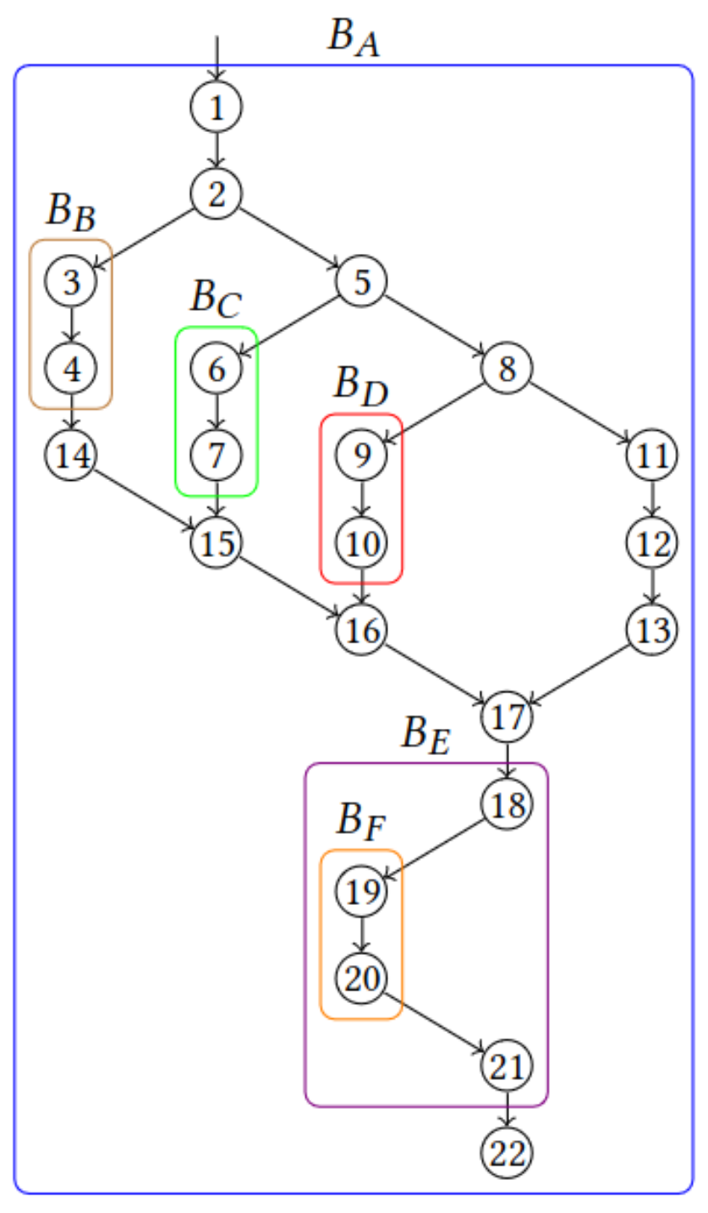
Supervisors: Dirk Beyer

Collaborators: Daniel Baier (ConVeY)

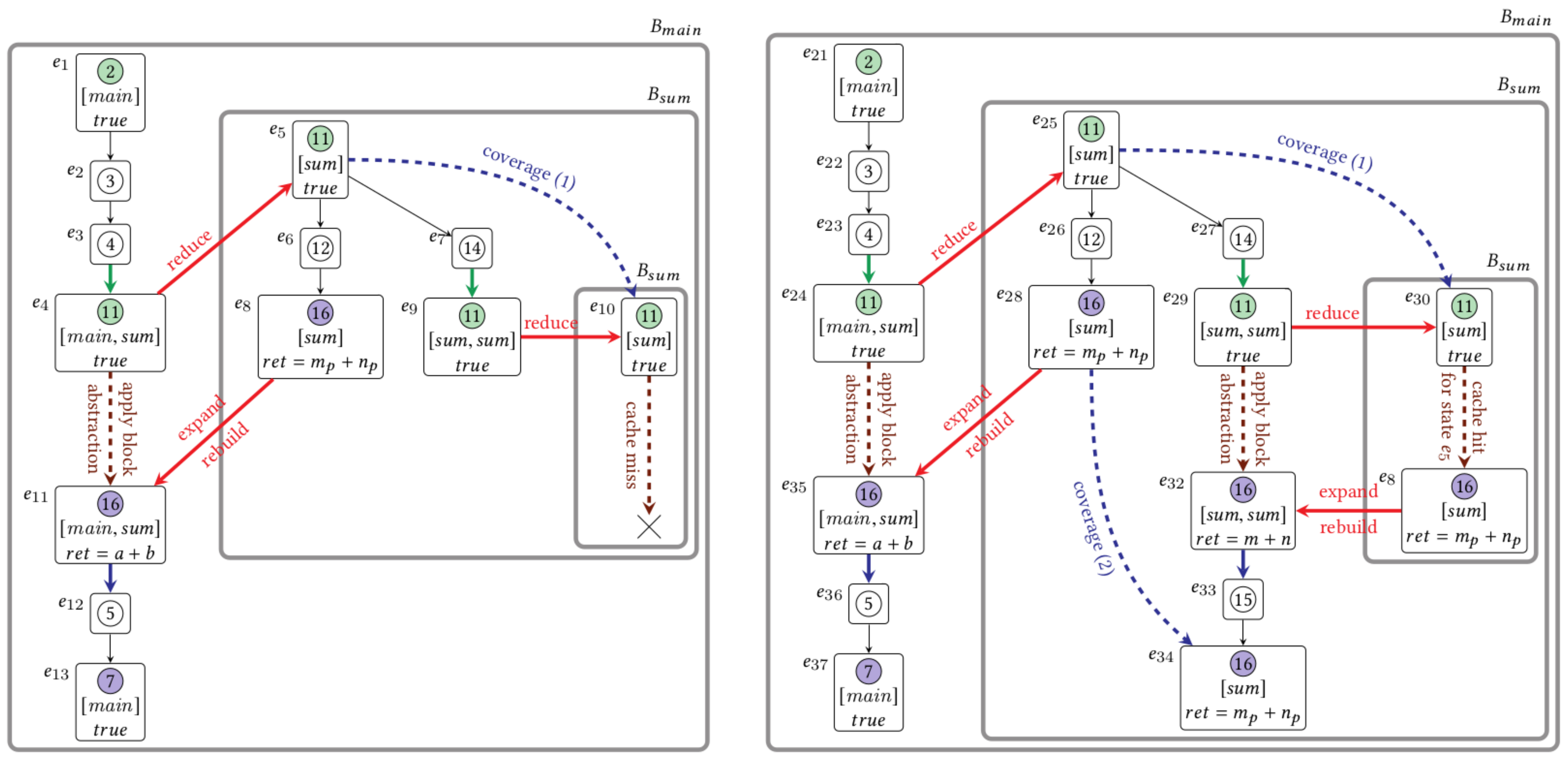
Block-Abstraction Memoization (BAM)

BAM applies a divide-and-conquer strategy for analyzing programs, splitting them into smaller blocks that are then analyzed. We extended the approach for an interprocedural analysis and for a multi-threaded approach. BAM works on a domain-independent level and has a low overhead.



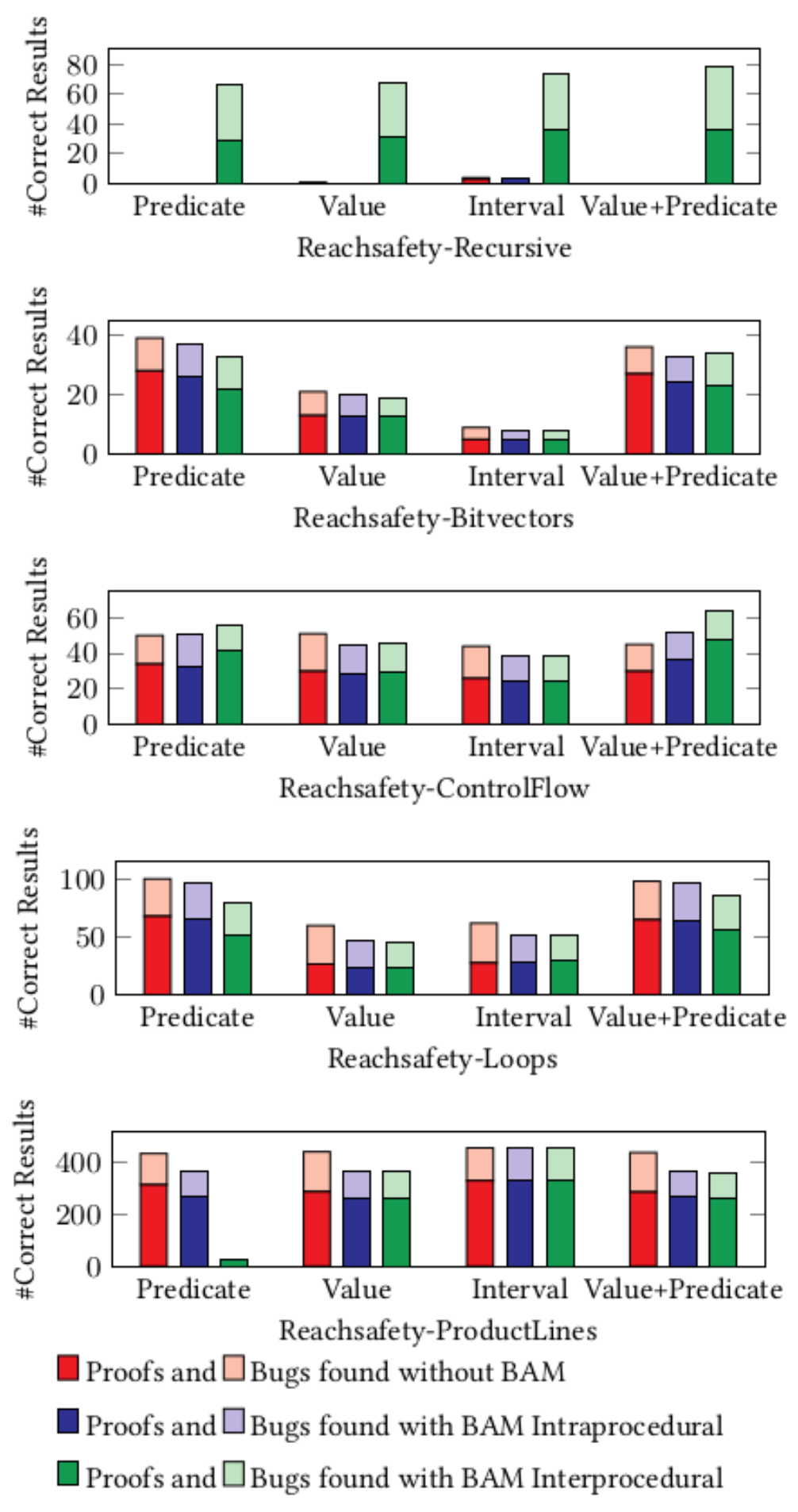


Interprocedural BAM [2]



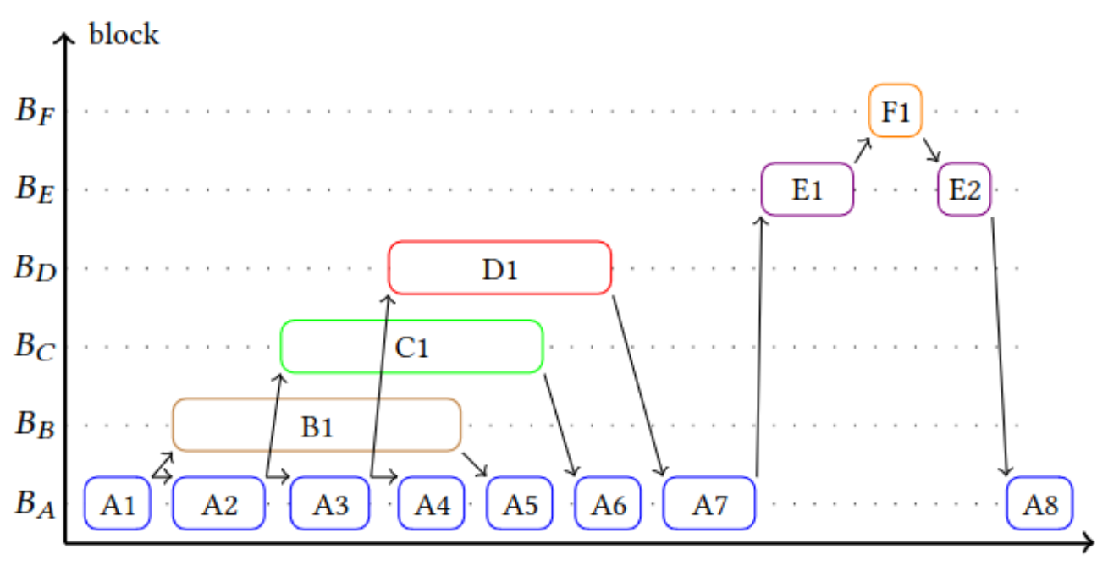
(a) After first iteration; cache miss leads to second iteration

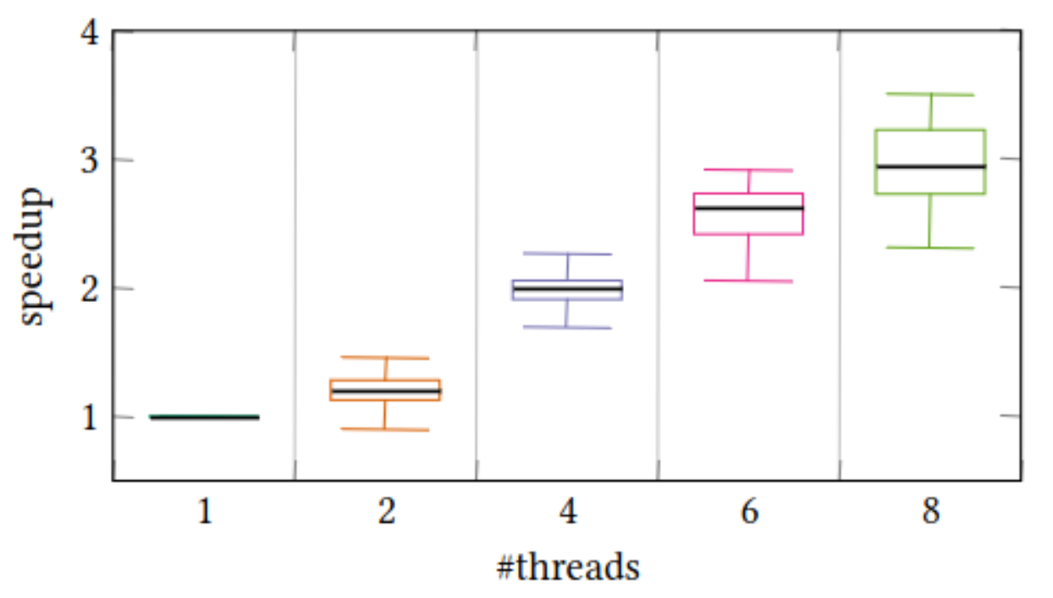
(b) After second iteration; fixed point is reached



Supports the analysis of recursive programs

Multithreaded BAM [3]

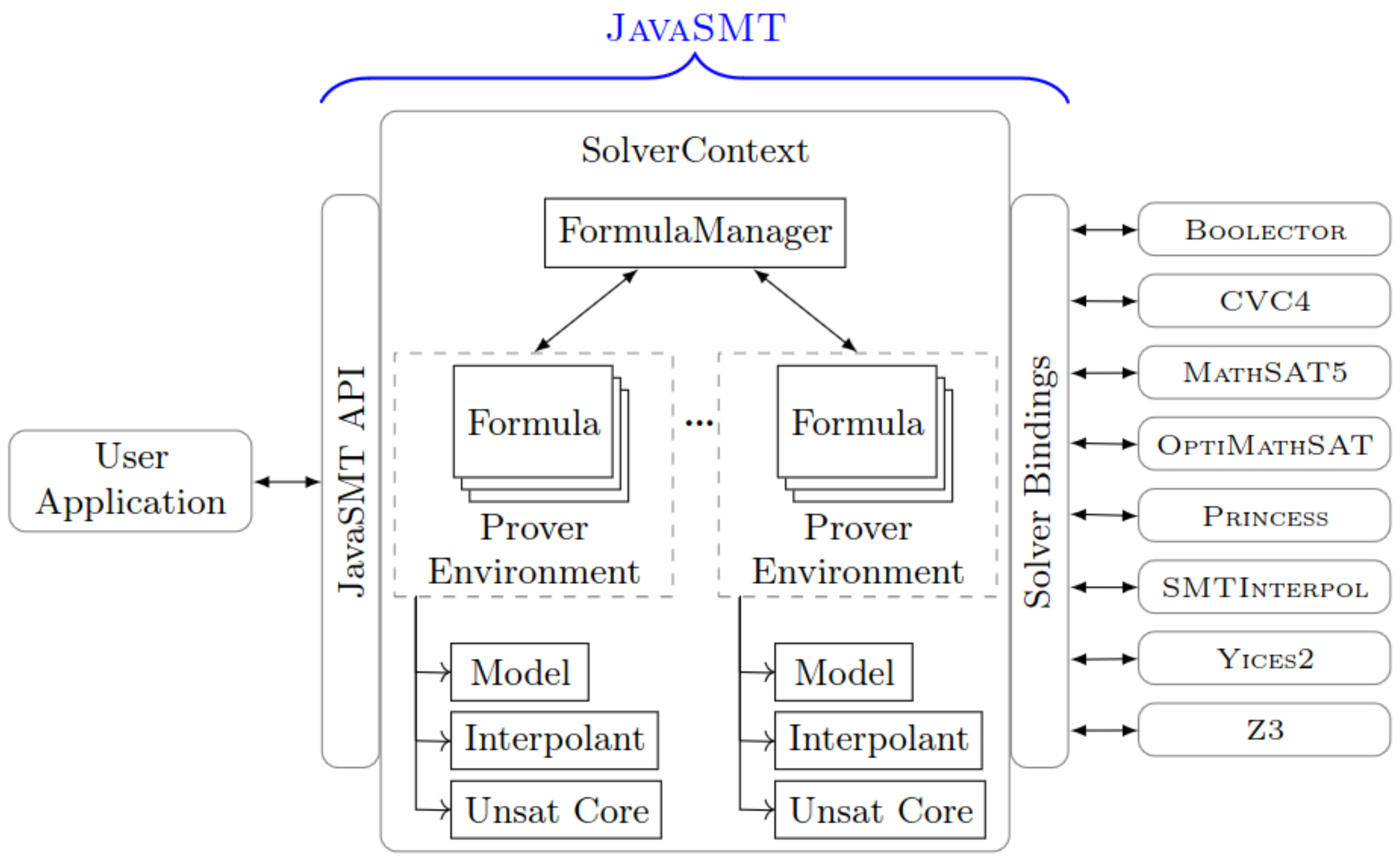




Distributes workload on several independent threads

JavaSMT 3 [1]

- Common Java API for SMT solvers
- Supports most used SMT theories
- Provides the most common API features
- 8 SMT solvers
- Typesafe
- Used in several software projects, including CPAchecker



JavaSMT

User Application

JavaSMT API

SolverContext

FormulaManager

Formula

Prover Environment

Model

Interpolant

Unsat Core

Solver Bindings

Boolector

CVC4

MathSAT5

OptiMathSAT

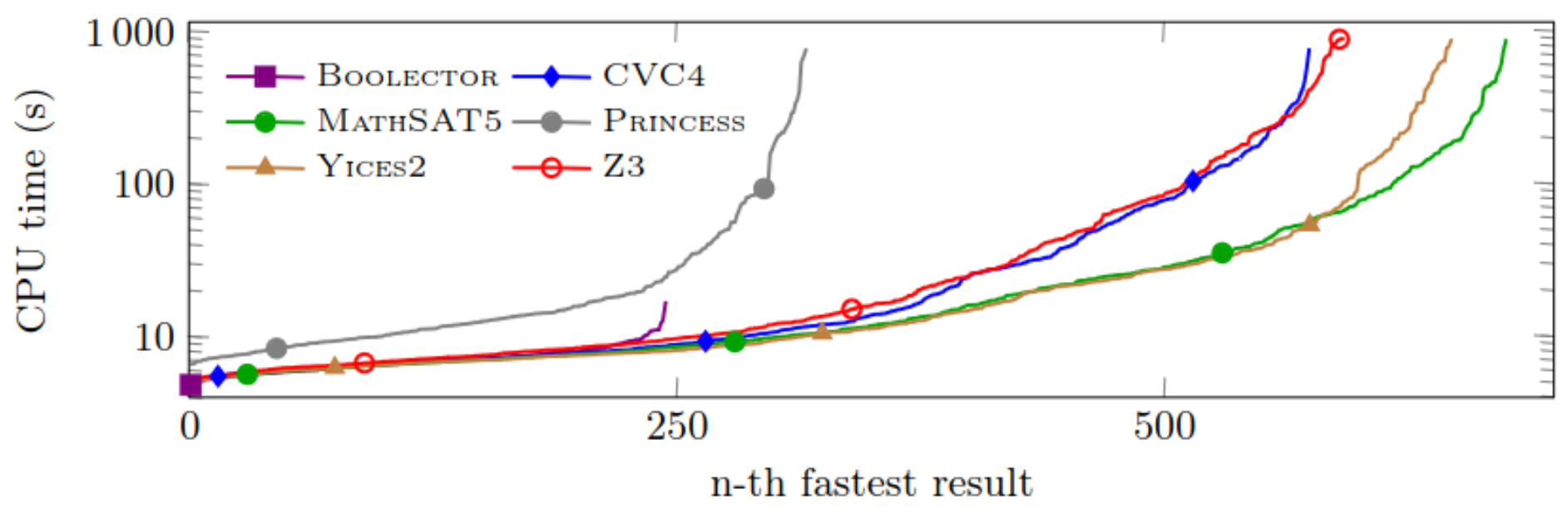
Princess

SMTInterpol

Yices2

Z3

		Boolector	CVC4	MathSAT5	OptiMathSAT	Princess	SMTInterpol	Yices2	Z3
SMT Theories	Integer	✗	✓	✓	✓	✓	✓	✓	✓
	Rational	✗	✓	✓	✓	✓	✓	✓	✓
	Array	✓	✓	✓	✓	✓	✓	✗	✓
	Bitvector	✓	✓	✓	✓	✓	✗	✓	✓
	Float	✗	✓	✓	✓	✗	✗	✗	✓
	UF	✓	✓	✓	✓	✓	✓	✓	✓
	Quantifier	✓	✓	✗	✗	✓	✗	✓	✓
Features	Incremental Solving	✓	✓	✓	✓	✓	✓	✓	✓
	Model	✓	✓	✓	✓	✓	✓	✓	✓
	Assumption Solving	✓	✗	✓	✓	✗	✗	✓	✓
	Interpolation	✗	✗	✓	✓	✓	✓	✗	✗
	Optimization	✗	✗	✗	✓	✗	✗	✗	✓
	UnsatCore	✗	✓	✓	✓	✓	✓	✓	✓
	UnsatCore with Assumptions	✗	✗	✓	✓	✗	✓	✓	✓
	SMT-LIB2 (plain text input)	✓	✓	✓	✓	✓	✓	✓	✓
	SMT-LIB2 (via API)	✗	✗	✓	✓	✓	✓	✗	✓
	Quantifier Elimination	✗	✓	✗	✗	✓	✗	✗	✓
	Formula Decomposition	✗	✓	✓	✓	✓	✓	✓	✓



We evaluated all SMT solvers available in JavaSMT using several software verification techniques against the same set of tasks, using the same hardware. The results support our claim that each solver has its own fingerprint of features and results.

References

[1] D. Baier, D. Beyer, and K. Friedberger. "JavaSMT 3: Interacting with SMT Solvers in Java". In: *Proceedings of the 33rd International Conference on Computer-Aided Verification (CAV 2021, Los Angeles, California, USA, July 18-24)*. Ed. by A. Silva and K. R. M. Leino. LNCS 12760. Springer, 2021, pp. 1–13.

[2] D. Beyer and K. Friedberger. "Domain-Independent Interprocedural Program Analysis using Block-Abstraction Memoization". In: *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2020, Virtual Event, USA, November 8-13)*. Ed. by P. Devanbu, M. Cohen, and T. Zimmermann. ACM, 2020, pp. 50–62.

[3] D. Beyer and K. Friedberger. "Domain-Independent Multi-threaded Software Model Checking". In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*. Ed. by M. Huchard, C. Kästner, and G. Fraser. ACM, 2018, pp. 634–644.