The Golem Horn Solver

Martin Blicha

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Department of
Distributed and
Dependable
Systems

About me

Software Verification with Constrained Horn Clauses

- Logic-based intermediate language for verification tasks
  - loop invariants, loops summaries, recursion, function summaries, ...
- Separation of concerns: modelling vs solving

Software Verification with Constrained Horn Clauses

- Logic-based intermediate language for verification tasks
  - loop invariants, loops summaries, recursion, function summaries, ...
- Separation of concerns: modelling vs solving
- CHC-based software verifiers
  - C/C++: SeaHorn, Korn
  - Rust: RustHorn
  - Java: JayHorn
  - Android: HornDroid
  - Solidity: SolCMC, SmartACE
- Horn solvers (CHC solvers)
  - Golem, Eldarica, Z3-Spacer, FreqHorn, Ultimate Unihorn, ...

Constrained Horn Clauses—Theory

- Fragment of first-order logic

$$\forall V.(\varphi \wedge P_1(T_1) \wedge P_2(T_2) \wedge \ldots \wedge P_n(T_n) \implies H(T))$$

- φ is a contraint in a background theory $\mathcal{T}$
- $\mathcal{T}$: linear arithmetic, arrays, bit-vectors, or their combinations
- V are variables, T_i are $\mathcal{T}$-terms over V
- P_i, H are uninterpreted predicates (disjoint from the signature of $\mathcal{T}$)

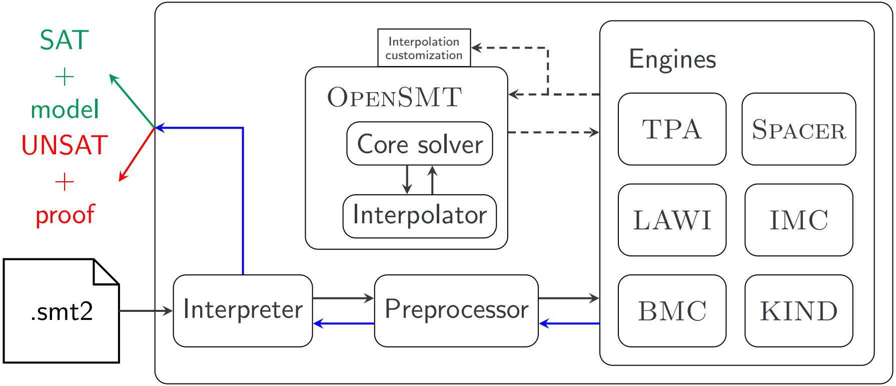
Constrained Horn Clauses—Theory

- Fragment of first-order logic

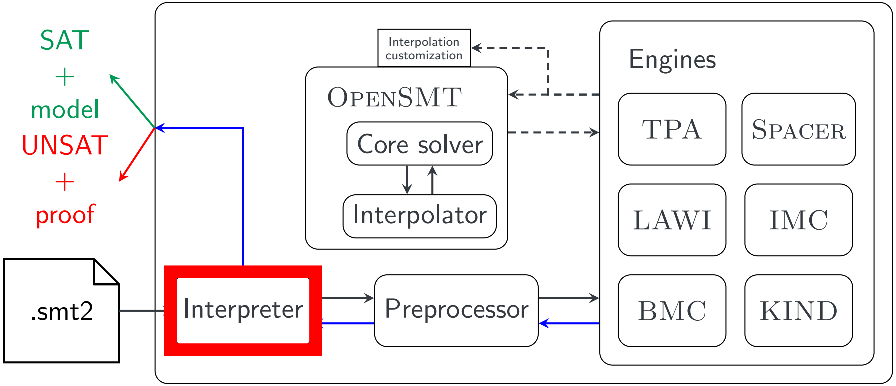
$$\forall V.(\varphi \wedge P_1(T_1) \wedge P_2(T_2) \wedge \ldots \wedge P_n(T_n) \implies H(T))$$

- φ is a contraint in a background theory $\mathcal{T}$
- $\mathcal{T}$: linear arithmetic, arrays, bit-vectors, or their combinations
- V are variables, T_i are $\mathcal{T}$-terms over V
- P_i, H are uninterpreted predicates (disjoint from the signature of $\mathcal{T}$)
- System of CHCs is
  - SAT, if $\exists$ interpretation of predicates that makes all clauses valid
  - UNSAT, if we can derive *false*
    - using instantiation and resolution

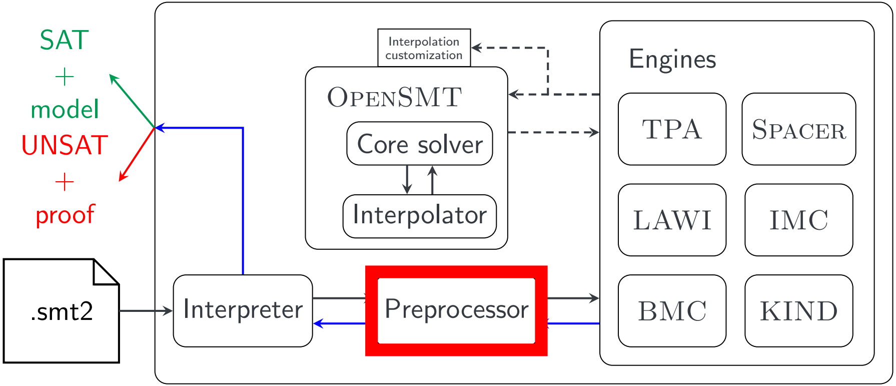
Golem Architecture



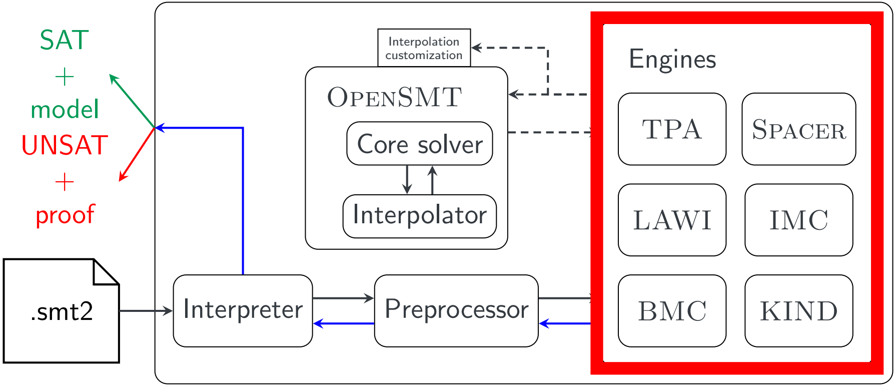
Golem Architecture



Golem Architecture

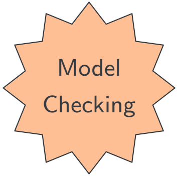


Golem Architecture



Engines

- Bounded Model Checking [BCCZ99]
- *k*-induction [SSS00]
- Interpolation-based Model Checking [McM03]
- Lazy Abstraction with Interpolants [McM06]
- Transition Power Abstraction [BFHS22b, BFHS22a]
- Spacer [KGC16]



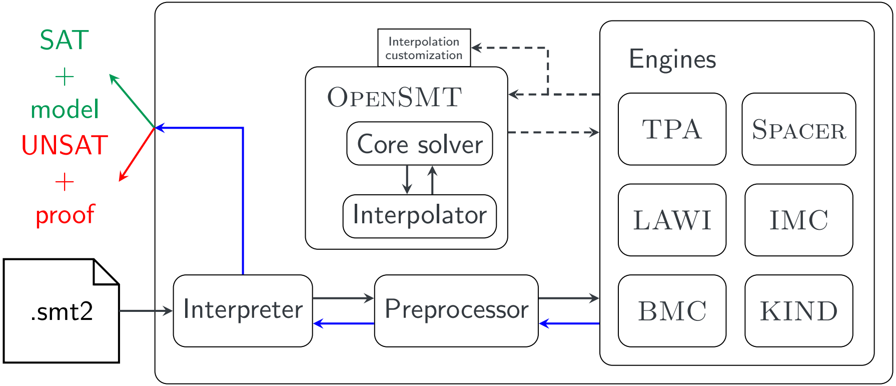
Engines

- Bounded Model Checking [BCCZ99]
- *k*-induction [SSS00]
- Interpolation-based Model Checking [McM03]
- Lazy Abstraction with Interpolants [McM06]
- Transition Power Abstraction [BFHS22b, BFHS22a]
- SPACER [KGC16]

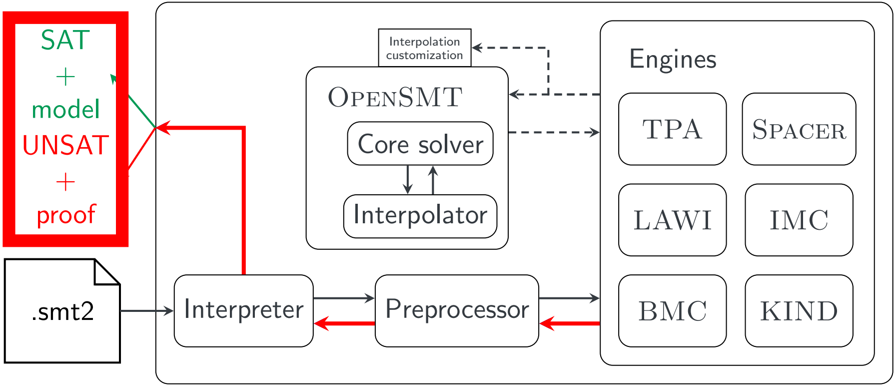
Model Checking

- Common components
  - (Incremental) SMT solving
  - Interpolation
  - Model-based projection

Golem Architecture



Golem Architecture



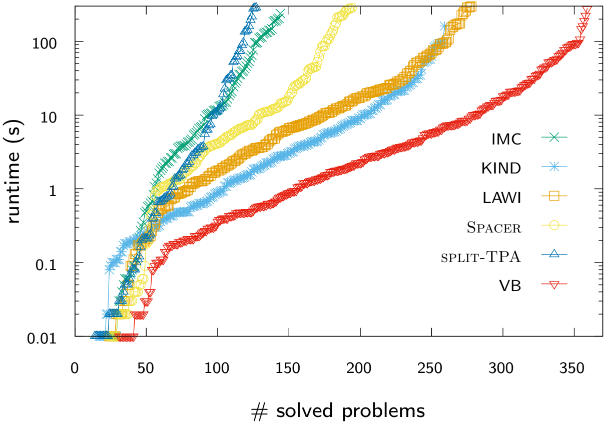
Evaluation

- Participating in CHC-COMP since 2021
- Several medals in LRA-TS, LIA-lin, LIA-nonlin tracks
- `https://chc-comp.github.io/`

Evaluation—LRA-TS

	BMC	KIND	IMC	LAWI	SPACER	SPLIT-TPA	VB
SAT	0	260	145	279	195	128	360
UNSAT	86	84	70	76	69	72	86

Evaluation—LRA-TS



Performance of GOLEM's engines on SAT benchmarks from LRA-TS track

Evaluation—LIA-Nonlin

	Golem-Spacer	Z3-Spacer	Eldarica
SAT	239 (4)	248 (13)	221 (6)
UNSAT	124 (2)	139 (5)	122 (0)

Table: CHC-COMP'22 selection

Future Work

- More theories: Arrays, ADTs, bit-vectors
- More engines: PD-KIND, predicate abstraction, ...
- Liveness
- Termination

Conclusion

GOLEM is

- Modular

Blicha et al., The GOLEM Horn Solver, CAV 2023

Conclusion

GOLEM is

- Modular
- Efficient

Blicha et al., The GOLEM Horn Solver, CAV 2023

Conclusion

GOLEM is

- Modular
- Efficient
- Great playground for SMT-based model-checking

Blicha et al., The GOLEM Horn Solver, CAV 2023

Conclusion

GOLEM is

- Modular
- Efficient
- Great playground for SMT-based model-checking
- Open-source `github.com/usi-verification-and-security/golem`

Blicha et al., The GOLEM Horn Solver, CAV 2023

Want to work on SMT/CHC?

References I

Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu.
Symbolic model checking without BDDs.
In W. Rance Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 193–207, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

Martin Blicha, Grigory Fedyukovich, Antti E. J. Hyvärinen, and Natasha Sharygina.
Split transition power abstractions for unbounded safety.
In Alberto Griggio and Neha Rungta, editors, *Proceedings of the 22nd Conference on Formal Methods in Computer-Aided Design – FMCAD 2022*, pages 349–358, Cham, 2022. TU Wien Academic Press.

References II

Martin Blicha, Grigory Fedyukovich, Antti E. J. Hyvärinen, and Natasha Sharygina.
Transition power abstractions for deep counterexample detection.
In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 524–542, Cham, 2022. Springer International Publishing.

Anvesh Komuravelli, Arie Gurfinkel, and Sagar Chaki.
SMT-based model checking for recursive programs.
Formal Methods in System Design, 48(3):175–205, Jun 2016.

Kenneth L. McMillan.
Interpolation and SAT-based model checking.
In Warren A. Hunt and Fabio Somenzi, editors, *Computer Aided Verification*, pages 1–13, Berlin Heidelberg, 2003. Springer.

References III

Kenneth L. McMillan.
Lazy abstraction with interpolants.
In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification*, pages 123–136, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

Mary Sheeran, Satnam Singh, and Gunnar Stålmarck.
Checking safety properties using induction and a SAT-solver.
In Warren A. Hunt and Steven D. Johnson, editors, *Formal Methods in Computer-Aided Design*, pages 127–144, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.