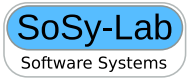
Software Model Checking: 20 Years and Beyond

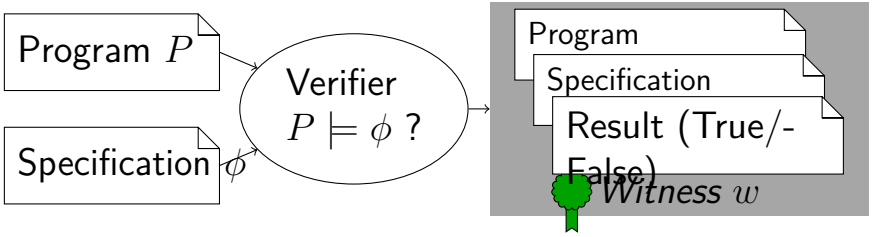
Dirk Beyer

LMU Munich, Germany

CPAchecker Workshop 2023, Luxembourg, 2023-09-11



Automatic Software Verification



Theorem for P and S: $P \models S$
If true, a proof of correctness was constructed.
If false, a proof by counterexample was constructed.

Automatic Software Verification

Theorem for P and S: $P \models S$
If true, a proof of correctness was constructed.
If false, a proof by counterexample was constructed.

Software often contains bugs. Thus, we appreciate tools that can answer with either outcome.

Verifiers are constructing invariants or counterexamples:
$\Rightarrow$ prove of correctness and incorrectness

Testers and **BMC** are important part of verification ecosystem:
$\Rightarrow$ prove of incorrectness

Validators: are important to ensure that results are correct
$\Rightarrow$ verify invariants and counterexamples

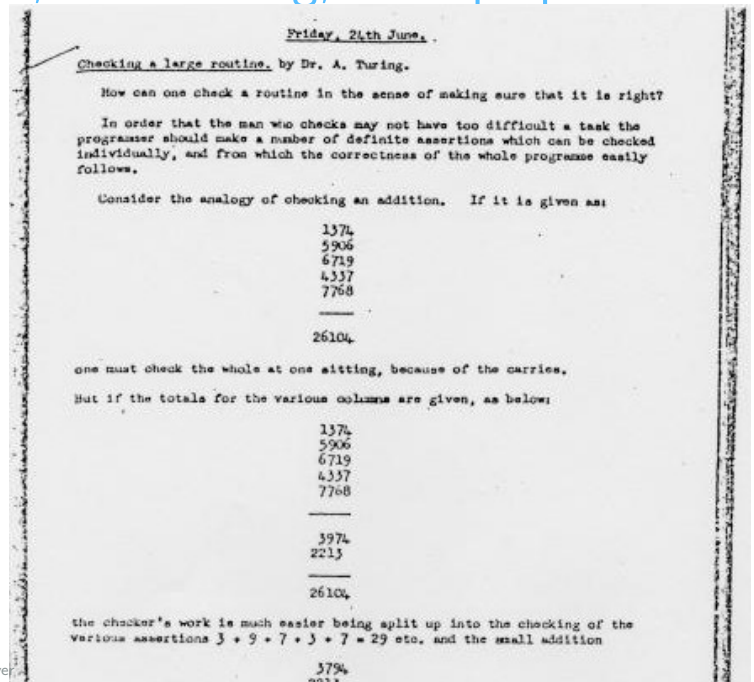
Part 1: Overview of SMC

Based on a joint paper with Andreas Podelski:
Software Model Checking: 20 Years and Beyond
In Principles of Systems Design, LNCS 13660, pages 554-582, 2022.
Springer. doi:10.1007/978-3-031-22337-2_27

Some Historical Landmarks

- **70 years ago: Assertions and Proof Decomposition**, Alan Turing, 1949 [44]
  "In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole program easily follows."

Landmarks, Alan Turing, 1949 [44]



Friday, 24th June.

Checking a large routine. by Dr. A. Turing.

How can one check a routine in the sense of making sure that it is right?

In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.

Consider the analogy of checking an addition. If it is given as:

```
 1374
 5906
 6719
 4337
 7768
------
26104
```

one must check the whole at one sitting, because of the carries.

But if the totals for the various columns are given, as below:

```
 1374
 5906
 6719
 4337
 7768
------

 3974
 2213
------
26104
```

the checker's work is much easier being split up into the checking of the various assertions $3 + 9 + 7 + 3 + 7 = 29$ etc. and the small addition

```
 3794
```

Some Historical Landmarks

- 70 years ago: Assertions and Proof Decomposition
- **60 years ago: Logic Interpolation**,
  William Craig, J. Symb. Log. 1957 [33]
  "Linear reasoning. A new form of the Herbrand-Gentzen theorem."
  Defines an interpolation for logic formulas
  Made popular by Ken McMillan [43]

Some Historical Landmarks

- 70 years ago: Assertions and Proof Decomposition
- 60 years ago: Logic Interpolation
- **50 years ago: Data-Flow Analysis and Abstract States**, Gary Kildall, POPL 1973 [42]
  "A Unified Approach to Global Program Optimization"
  Defines algorithm, meet operation, lattice, etc.
  Extended and made popular for program analysis by
  F. Nielson, H. R. Nielson, C. Hankin, P. Cousot, R. Cousot

Landmarks, Gary Kildall, POPL 1973 [42]

A UNIFIED APPROACH TO GLOBAL
PROGRAM OPTIMIZATION

Gary A. Kildall

Computer Science Group
Naval Postgraduate School
Monterey, California

Abstract

A technique is presented for global analysis of program structure in order to perform compile time optimization of object code generated for expressions. The global expression optimization presented includes constant propagation, common subexpression elimination, elimination of redundant register load operations, and live expression analysis. A general purpose program flow analysis algorithm is developed which depends upon the existence of an "optimizing function." The algorithm is defined formally using a directed graph model of program flow structure, and is shown to be correct. Several optimizing functions are defined which, when used in conjunction with the flow analysis algorithm, provide the various forms of code optimization. The flow analysis algorithm is sufficiently general that additional functions can easily be defined for other forms of global code optimization.

Some Historical Landmarks

- 70 years ago: Assertions and Proof Decomposition
- 60 years ago: Logic Interpolation
- 50 years ago: Data-Flow Analysis and Abstract States
- **40 years ago: LTL and Model Checking**,
  Pnueli, Clarke, Emerson, Sifakis, 1981
  Specification languages, modeling languages, algorithms, theory, tools
  LTL, CTL, automata, Kripke structures, model checking, $\rightarrow$ software model checking

  "Handbook of Model Checking", Springer, 2018 [30]

Some Historical Landmarks

- 70 years ago: Assertions and Proof Decomposition
- 60 years ago: Logic Interpolation
- 50 years ago: Data-Flow Analysis and Abstract States
- 40 years ago: LTL and Model Checking
- **25 years ago: Predicate Abstraction**,
  Graf, Saïdi, 1997 [35]
  Enabling idea to project software to
  a (smaller) finite state space

Some Historical Landmarks

- 70 years ago: Assertions and Proof Decomposition
- 60 years ago: Logic Interpolation
- 50 years ago: Data-Flow Analysis and Abstract States
- 40 years ago: LTL and Model Checking
- 25 years ago: Predicate Abstraction
- **20 years ago: Tools for Software Model Checking**
  "1st generation" of tools:
  - Summer 2000: SLAM [2, 1]
  - Fall 2000: BLAST [37, 14]
  - 2004: SATABS [31]

  SLAM paper received Test-of-Time Award from PLDI, first to apply predicate abstraction + CEGAR to software. BLAST received gold medals in competitions.

Some Historical Landmarks

- 70 years ago: Assertions and Proof Decomposition
- 60 years ago: Logic Interpolation
- 50 years ago: Data-Flow Analysis and Abstract States
- 40 years ago: LTL and Model Checking
- 25 years ago: Predicate Abstraction
- 20 years ago: Tools for Software Model Checking
- **15 years ago: Satisfiability Modulo Theory**
  Standard formula format SMTLIB [3]
  Enormous breakthrough, many tools, ... [32]

Some Historical Landmarks

- 70 years ago: Assertions and Proof Decomposition
- 60 years ago: Logic Interpolation
- 50 years ago: Data-Flow Analysis and Abstract States
- 40 years ago: LTL and Model Checking
- 25 years ago: Predicate Abstraction
- 20 years ago: Tools for Software Model Checking
- 15 years ago: Satisfiability Modulo Theory
- **10 years ago: Competition on Software Verification**
  2012 [4]
  Competitions create awareness of tools,
  provide comparative evaluations, establish standards
  (input, exchange, comparability, reproducibility)

Some Historical Landmarks

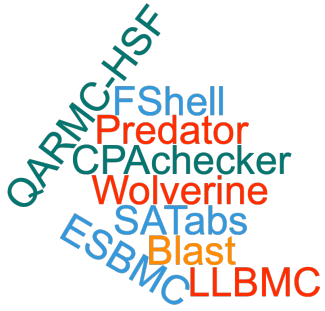
- 70 years ago: Assertions and Proof Decomposition
- 60 years ago: Logic Interpolation
- 50 years ago: Data-Flow Analysis and Abstract States
- 40 years ago: LTL and Model Checking
- 25 years ago: Predicate Abstraction
- 20 years ago: Tools for Software Model Checking
- 15 years ago: Satisfiability Modulo Theory
- 10 years ago: Competition on Software Verification
- **today:** From **lack** of tools to **abundance** of tools
  Problem: missing standard interfaces, missing cooperation

Competitions in Software Verification and Testing

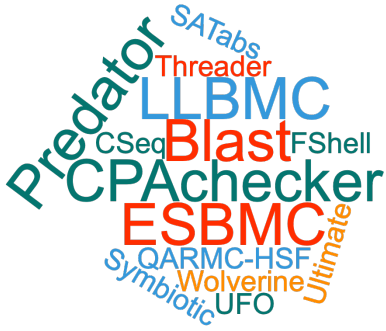
- RERS: off-site, tools, free-style [39]
- SV-COMP: off-site, automatic tools, controlled [4]
- Test-Comp: off-site, automatic tools, controlled [6]
- VerifyThis: on-site, interactive, teams [40]

(alphabetic order)

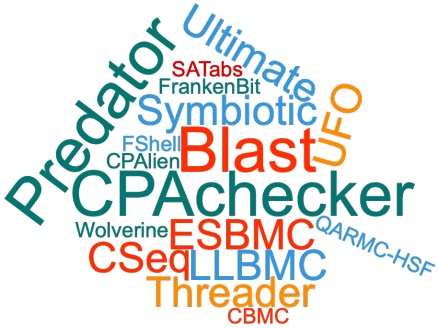
SV-COMP (Automatic Tools 2012)



SV-COMP (Automatic Tools 2013, cumulative)

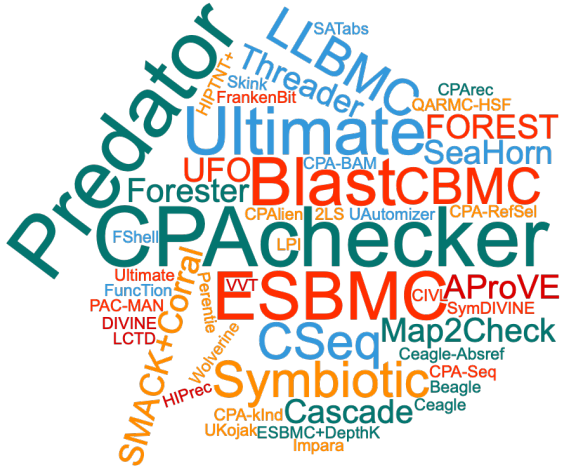


SV-COMP (Automatic Tools 2014, cumulative)

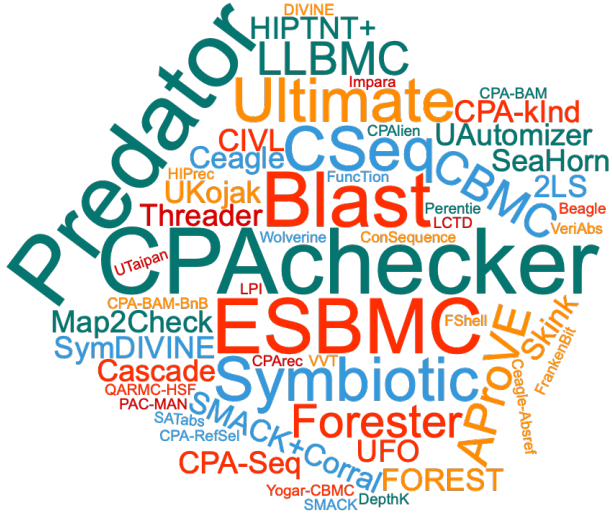


SV-COMP (Automatic Tools 2015, cumulative)

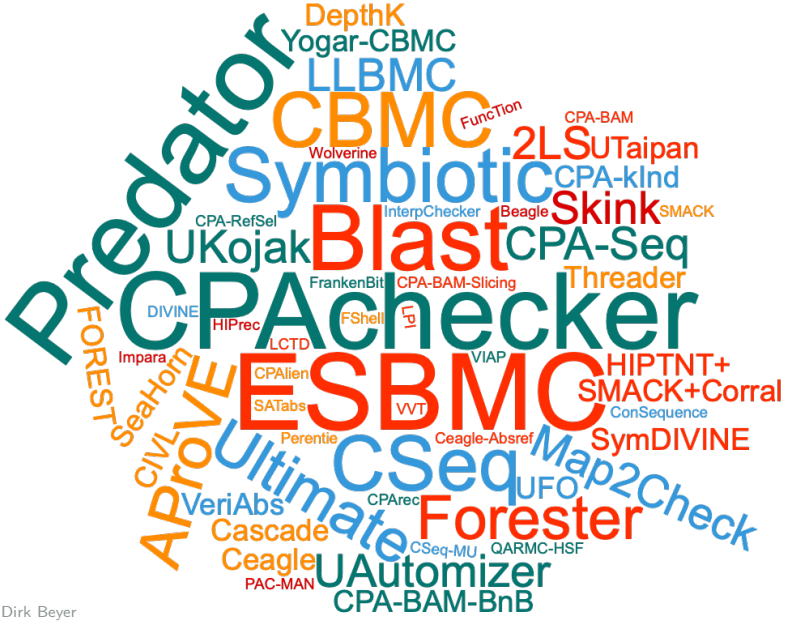
SV-COMP (Automatic Tools 2016, cumulative)



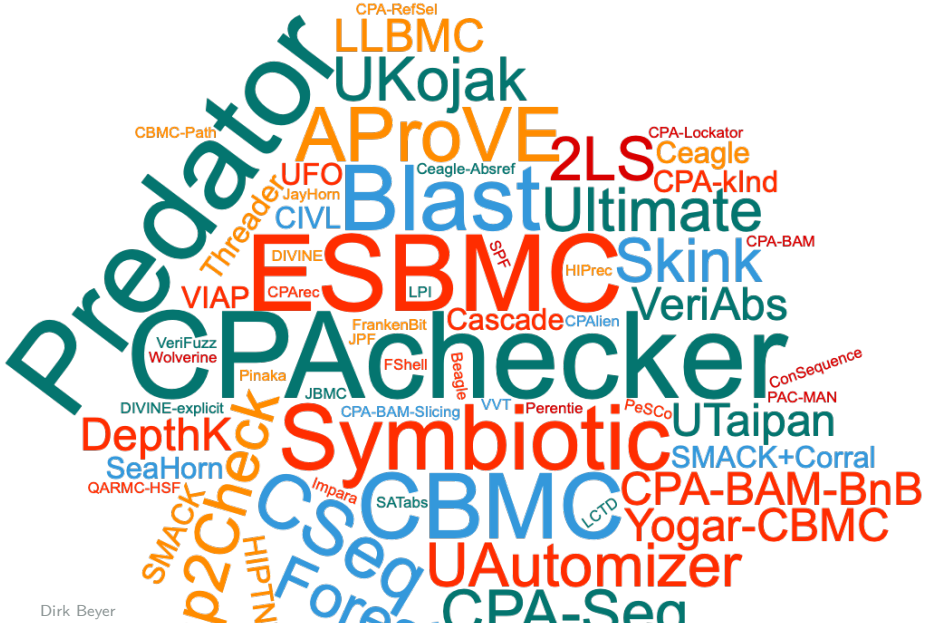
SV-COMP (Automatic Tools 2017, cumulative)



SV-COMP (Automatic Tools 2018, cumulative)



SV-COMP (Automatic Tools 2019, cumulative)

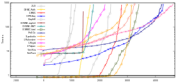


What is the best verifier?

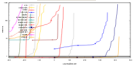
- Many different kinds of programs seem to require many different good tools with different strengths

SV-COMP (Automatic Tools)

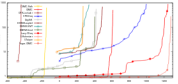
ReachSafety
1. VeriAbs
2. CPA-Seq
3. PeSCo



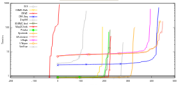
MemSafety
1. Symbiotic
2. PredatorHP
3. CPA-Seq



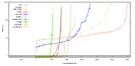
ConcurrencySafety
1. Yogar-CBMC
2. Lazy-CSeq
3. CPA-Seq



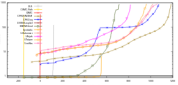
NoOverflows
1. UAutomizer
2. UTaipan
3. CPA-Seq



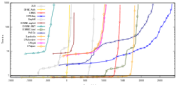
Termination
1. UAutomizer
2. AProVE
3. CPA-Seq



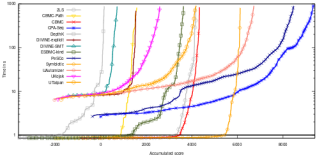
SoftwareSystems
1. CPA-BAM-BnB
2. CPA-Seq
3. VeriAbs



FalsificationOverall
1. CPA-Seq
2. PeSCo
3. ESBMC-kind



Overall
1. CPA-Seq
2. PeSCo
3. UAutomizer



https://sv-comp.sosy-lab.org/2019/results

Which techniques are used?

Participant	CEGAR	Predicate Abstraction	Symbolic Execution	Bounded Model Checking	k-Induction	Property-Directed Reach.	Explicit-Value Analysis	Numeric. Interval Analysis	Shape Analysis	Separation Logic	Bit-Precise Analysis	ARG-Based Analysis	Lazy Abstraction	Interpolation	Automata-Based Analysis	Concurrency Support	Ranking Functions	Evolutionary Algorithms
2LS				✓	✓			✓			✓						✓	
APROVE			✓				✓	✓		✓	✓						✓	
CBMC				✓							✓					✓		
CBMC-PATH				✓							✓					✓		
CPA-BAM-BNB	✓	✓					✓				✓	✓	✓	✓				
CPA-LOCKATOR	✓	✓					✓				✓	✓	✓	✓		✓		
CPA-SEQ	✓	✓		✓	✓		✓	✓	✓		✓		✓	✓		✓	✓	
DEPTHK				✓	✓						✓					✓		
DIVINE-EXPLICIT							✓				✓					✓		
DIVINE-SMT							✓				✓					✓		
ESBMC-KIND				✓	✓						✓					✓		
JAYHORN	✓	✓				✓		✓					✓	✓				
JBMC				✓							✓					✓		
JPF				✓			✓	✓			✓					✓		
LAZY-CSEQ				✓							✓					✓		
MAP2CHECK				✓							✓							
PESCO	✓	✓		✓	✓		✓	✓	✓		✓	✓	✓	✓		✓	✓	
PINAKA			✓	✓							✓							
PREDATORHP									✓									
SKINK	✓						✓							✓	✓			
SMACK	✓			✓		✓					✓		✓			✓		
SPF			✓						✓							✓		
SYMBIOTIC			✓					✓			✓							
UAUTOMIZER	✓	✓									✓		✓	✓	✓		✓	
UKOJAK	✓	✓									✓		✓	✓				
UTAIPAN	✓	✓									✓		✓	✓	✓			
VERIABS	✓			✓	✓		✓	✓										
VERIFUZZ				✓				✓										✓
VIAP																		
YOGAR-CBMC	✓			✓							✓		✓			✓		
YOGAR-CBMC-PAR.	✓			✓							✓		✓			✓		

Competition Report [5]
doi:10.1007/978-3-030-17502-3_9

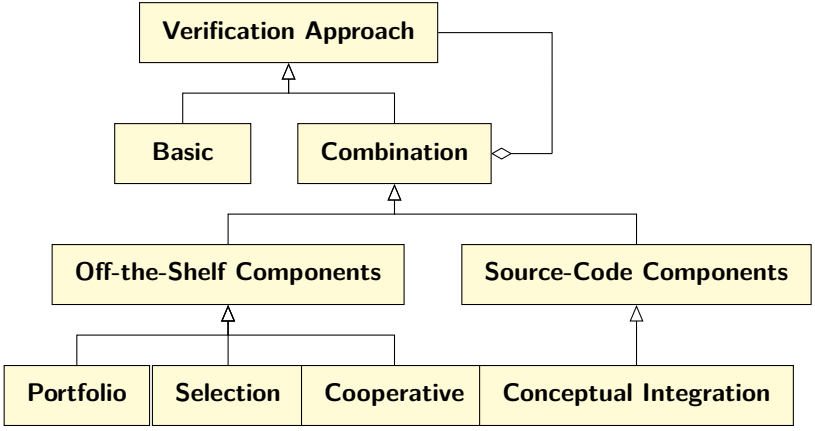
Unification Efforts

A Unifying View on SMT-Based Software Verification
Dirk Beyer, Matthias Dangl, Philipp Wendler
Journal of Automated Reasoning, 2018 [10]
based on

- Configurable Program Analysis [17, 18]
- Large-Block Encoding [8, 21]
- Satisfiability Modulo Theories [3]

Combining Model Checking and Data-Flow Analysis
Dirk Beyer, Sumit Gulwani, David Schmidt
Handbook of Model Checking, 2018 [12]

Combinations and Cooperation [28]



Decompose Software Verification [19]

Step 1: Find invariants
Step 2: Try to prove that invariants hold
Step 3: Try to prove that invariants imply the specification
Repeat if no success

$\Rightarrow$ Validator **LIV** [26] splits programs into straight-line programs of the same programming language, which enables

- use off-the-shelf verifiers to verify the straight-line programs
- parallelize the verification of the straight-line programs

Visit talk at ASE on Thursday before noon
LIV: Invariant Validation Using Straight-Line Programs

Conclusion Software Verification

- Software verification: successful past, bright future
- Competitions solve several problems
- Science as knowledge compression
- Cooperating combinations are the future

Part 2: CPA✓ CPAchecker

- History
- Features

CPAchecker History

- 2002: BLAST with lazy abstraction refinement [14, 38]
- 2003: Multi-threading support [36]
- 2004: Test-case generation, interpolation, spec. lang. [14, 7]
- 2005: Memory safety, predicated lattices [34, 13]
- 2006: Lazy shape analysis [16]
- Maintenance and extensions became extremely difficult because of design choices that were not easy to revert
- 2007: Configurable program analysis [17, 18],
  CPAchecker was started
  as complete reimplementation from scratch [20]

CPAchecker History (2)

- 2009: Large-block encoding [8, FMCAD '09]
- 2010: Adjustable-block encoding [21, FMCAD '10]
- 2012: Conditional model checking [15, FSE '12],
  PredAbs vs. Impact [29, FMCAD '12]
- 2013: Explicit-state MC [22, FASE '13],
  BDDs [27, STTT '14],
  Precision reuse [23, FSE '13]
- 2014: CPAchecker goes cloud [11, CAV '14]
- 2015: k-Induction [9, CAV '15]
  Sliced path prefixes [24, FORTE '15]
- 2020: PDR [9, TACAS '20]
- ...

CPAchecker

- Framework for Software Verification
  - Mostly written in Java
  - Open Source: Apache 2.0 License
    https://cpachecker.sosy-lab.org
  - 122 contributors
  - 386 346 lines of code
  - 32 531 commits
  - 104 years of effort (COCOMO model)
  - Has a well established, mature codebase maintained by a large development team

Source: https://openhub.net/p/cpachecker on 2023-09-11

CPAchecker: Features

- Input language C (experimental: Java)
- Web frontend available:
  https://vcloud.sosy-lab.org/cpachecker/webclient/run
- Counterexample output with graphs
- Benchmarking infrastructure available
  (with large cluster of machines)
- Cross-platform: Linux, Mac, Windows

CPAchecker: Achievements

- Among world's best software verifiers:
  https://sv-comp.sosy-lab.org/2021/results/
- Continuous success in competition SV-COMP since 2012
  (103 medals: 27x gold, 39x silver, 37x bronze)
- Awarded Gödel medal
  by Kurt Gödel Society

- Used for Linux driver verification
  with dozens of real bugs found and fixed in Linux [41, 25]

CPAchecker: Concepts

- Completely modular, and thus flexible and easily extensible
- Every abstract domain is implemented as a "Configurable Program Analysis" (CPA)
- E.g., predicate abstraction, explicit-value analysis, intervals, octagon, BDDs, memory graphs, and more
- Algorithms are central and implemented only once
- Separation of concerns
- Combined with Composite pattern

Try CPAchecker

- Online at SoSy-Lab VerifierCloud:
  https://vcloud.sosy-lab.org/cpachecker/webclient/run
- Download for Linux/Windows:
  https://cpachecker.sosy-lab.org
  - Run `scripts/cpa.sh` | `scripts\cpa.bat`
  - `-default <FILE>`
  - Windows/Mac need to disable bitprecise analysis:
    `-predicateAnalysis-linear`
    `-setprop solver.solver=smtinterpol`
    `-setprop analysis.checkCounterexamples=false`
- Open graphical report in browser: `output/*.html`
- Open `.dot` files with dotty / xdot (www.graphviz.org/)

Conclusions CPAchecker

- Flexible framework
- Open source; large group of contributors
- Most state-of-the-art approaches integrated
- Decent performance (see competitions)
- Many papers describing its concepts

Conclusion Software Verification

- Software verification: successful past, bright future
- Competitions solve several problems
- Science as knowledge compression
- Cooperating combinations are the future

References I

[1] Ball, T., Levin, V., Rajamani, S.K.: A decade of software model checking with SLAM. Commun. ACM **54**(7), 68–76 (2011). https://doi.org/10.1145/1965724.1965743

[2] Ball, T., Majumdar, R., Millstein, T.D., Rajamani, S.K.: Automatic predicate abstraction of C programs. In: Proc. PLDI. pp. 203–213. ACM (2001). https://doi.org/10.1145/378795.378846

[3] Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB Standard: Version 2.5. Tech. rep., University of Iowa (2015), available at https://smtlib.cs.uiowa.edu/

[4] Beyer, D.: Competition on software verification (SV-COMP). In: Proc. TACAS. pp. 504–524. LNCS 7214, Springer (2012). https://doi.org/10.1007/978-3-642-28756-5_38

[5] Beyer, D.: Automatic verification of C and Java programs: SV-COMP 2019. In: Proc. TACAS (3). pp. 133–155. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_9

References II

[6] Beyer, D.: Competition on software testing (Test-Comp). In: Proc. TACAS (3). pp. 167–175. LNCS 11429, Springer (2019).
https://doi.org/10.1007/978-3-030-17502-3_11

[7] Beyer, D., Chlipala, A.J., Henzinger, T.A., Jhala, R., Majumdar, R.: Generating tests from counterexamples. In: Proc. ICSE. pp. 326–335. IEEE (2004).
https://doi.org/10.1109/ICSE.2004.1317455

[8] Beyer, D., Cimatti, A., Griggio, A., Keremoglu, M.E., Sebastiani, R.: Software model checking via large-block encoding. In: Proc. FMCAD. pp. 25–32. IEEE (2009).
https://doi.org/10.1109/FMCAD.2009.5351147

[9] Beyer, D., Dangl, M., Wendler, P.: Boosting k-induction with continuously-refined invariants. In: Proc. CAV. pp. 622–640. LNCS 9206, Springer (2015).
https://doi.org/10.1007/978-3-319-21690-4_42

[10] Beyer, D., Dangl, M., Wendler, P.: A unifying view on SMT-based software verification. J. Autom. Reasoning **60**(3), 299–335 (2018).
https://doi.org/10.1007/s10817-017-9432-6

References III

[11] Beyer, D., Dresler, G., Wendler, P.: Software verification in the Google App-Engine cloud. In: Proc. CAV. pp. 327–333. LNCS 8559, Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_21

[12] Beyer, D., Gulwani, S., Schmidt, D.: Combining model checking and data-flow analysis. In: Handbook of Model Checking, pp. 493–540. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_16

[13] Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: Checking memory safety with BLAST. In: Proc. FASE. pp. 2–18. LNCS 3442, Springer (2005). https://doi.org/10.1007/978-3-540-31984-9_2

[14] Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: The software model checker BLAST. Int. J. Softw. Tools Technol. Transfer **9**(5-6), 505–525 (2007). https://doi.org/10.1007/s10009-007-0044-z

[15] Beyer, D., Henzinger, T.A., Keremoglu, M.E., Wendler, P.: Conditional model checking: A technique to pass information between verifiers. In: Proc. FSE. ACM (2012). https://doi.org/10.1145/2393596.2393664

References IV

[16] Beyer, D., Henzinger, T.A., Théoduloz, G.: Lazy shape analysis. In: Proc. CAV. pp. 532–546. LNCS 4144, Springer (2006). https://doi.org/10.1007/11817963_48

[17] Beyer, D., Henzinger, T.A., Théoduloz, G.: Configurable software verification: Concretizing the convergence of model checking and program analysis. In: Proc. CAV. pp. 504–518. LNCS 4590, Springer (2007). https://doi.org/10.1007/978-3-540-73368-3_51

[18] Beyer, D., Henzinger, T.A., Théoduloz, G.: Program analysis with dynamic precision adjustment. In: Proc. ASE. pp. 29–38. IEEE (2008). https://doi.org/10.1109/ASE.2008.13

[19] Beyer, D., Kanav, S.: An interface theory for program verification. In: Proc. ISoLA (1). pp. 168–186. LNCS 12476, Springer (2020). https://doi.org/10.1007/978-3-030-61362-4_9

[20] Beyer, D., Keremoglu, M.E.: CPAchecker: A tool for configurable software verification. In: Proc. CAV. pp. 184–190. LNCS 6806, Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_16

References V

[21] Beyer, D., Keremoglu, M.E., Wendler, P.: Predicate abstraction with adjustable-block encoding. In: Proc. FMCAD. pp. 189–197. FMCAD (2010), https://www.sosy-lab.org/research/pub/2010-FMCAD.Predicate_Abstraction_with_Adjustable-Block_Encoding.pdf

[22] Beyer, D., Löwe, S.: Explicit-state software model checking based on CEGAR and interpolation. In: Proc. FASE. pp. 146–162. LNCS 7793, Springer (2013). https://doi.org/10.1007/978-3-642-37057-1_11

[23] Beyer, D., Löwe, S., Novikov, E., Stahlbauer, A., Wendler, P.: Precision reuse for efficient regression verification. In: Proc. FSE. pp. 389–399. ACM (2013). https://doi.org/10.1145/2491411.2491429

[24] Beyer, D., Löwe, S., Wendler, P.: Sliced path prefixes: An effective method to enable refinement selection. In: Proc. FORTE. pp. 228–243. LNCS 9039, Springer (2015). https://doi.org/10.1007/978-3-319-19195-9_15

[25] Beyer, D., Petrenko, A.K.: Linux driver verification. In: Proc. ISoLA. pp. 1–6. LNCS 7610, Springer (2012). https://doi.org/10.1007/978-3-642-34032-1_1

References VI

[26] Beyer, D., Spiessl, M.: LIV: A loop-invariant validation using straight-line programs. In: Proc. ASE (2023)

[27] Beyer, D., Stahlbauer, A.: BDD-based software verification: Applications to event-condition-action systems. Int. J. Softw. Tools Technol. Transfer **16**(5), 507–518 (2014). https://doi.org/10.1007/s10009-014-0334-1

[28] Beyer, D., Wehrheim, H.: Verification artifacts in cooperative verification: Survey and unifying component framework. In: Proc. ISoLA (1). pp. 143–167. LNCS 12476, Springer (2020). https://doi.org/10.1007/978-3-030-61362-4_8

[29] Beyer, D., Wendler, P.: Algorithms for software model checking: Predicate abstraction vs. IMPACT. In: Proc. FMCAD. pp. 106–113. FMCAD (2012), https://www.sosy-lab.org/research/pub/2012-FMCAD.Algorithms_for_Software_Model_Checking.pdf

[30] Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R.: Handbook of Model Checking. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8

References VII

[31] Clarke, E.M., Kröning, D., Sharygina, N., Yorav, K.: SATABS: SAT-based predicate abstraction for ANSI-C. In: Proc. TACAS. pp. 570–574. LNCS 3440, Springer (2005). https://doi.org/10.1007/978-3-540-31980-1_40

[32] Cok, D.R., Déharbe, D., Weber, T.: The 2014 SMT competition. JSAT **9**, 207–242 (2016)

[33] Craig, W.: Linear reasoning. A new form of the Herbrand-Gentzen theorem. J. Symb. Log. **22**(3), 250–268 (1957). https://doi.org/10.2307/2963593

[34] Fischer, J., Jhala, R., Majumdar, R.: Joining data flow with predicates. In: Proc. FSE. pp. 227–236. ACM (2005). https://doi.org/10.1145/1081706.1081742

[35] Graf, S., Saïdi, H.: Construction of abstract state graphs with PVS. In: Proc. CAV. pp. 72–83. LNCS 1254, Springer (1997). https://doi.org/10.1007/3-540-63166-6_10

[36] Henzinger, T.A., Jhala, R., Majumdar, R., Qadeer, S.: Thread-modular abstraction refinement. In: Proc. CAV. pp. 262–274. LNCS 2725, Springer (2003)

[37] Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. In: Proc. POPL. pp. 58–70. ACM (2002). https://doi.org/10.1145/503272.503279

References VIII

[38] Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Software verification with Blast. In: Proc. SPIN, pp. 235–239. LNCS 2648, Springer (2003)

[39] Howar, F., Isberner, M., Merten, M., Steffen, B., Beyer, D.: The RERS grey-box challenge 2012: Analysis of event-condition-action systems. In: Proc. ISoLA. pp. 608–614. LNCS 7609, Springer (2012). https://doi.org/10.1007/978-3-642-34026-0_45

[40] Huisman, M., Klebanov, V., Monahan, R.: VerifyThis 2012: A program verification competition. STTT **17**(6), 647–657 (2015). https://doi.org/10.1007/s10009-015-0396-8

[41] Khoroshilov, A.V., Mutilin, V.S., Petrenko, A.K., Zakharov, V.: Establishing Linux driver verification process. In: Proc. Ershov Memorial Conference. pp. 165–176. LNCS 5947, Springer (2009). https://doi.org/10.1007/978-3-642-11486-1_14

[42] Kildall, G.A.: A unified approach to global program optimization. In: Proc. POPL. pp. 194–206. ACM (1973). https://doi.org/10.1145/512927.512945

[43] McMillan, K.L.: Interpolation and SAT-based model checking. In: Proc. CAV. pp. 1–13. LNCS 2725, Springer (2003). https://doi.org/10.1007/978-3-540-45069-6_1

References IX

[44] Turing, A.: Checking a large routine. In: Report on a Conference on High Speed Automatic Calculating Machines. pp. 67–69. Cambridge Univ. Math. Lab. (1949)