Software Verification

Historical Landmarks and Current Developments

Dirk Beyer

LMU Munich, Germany

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

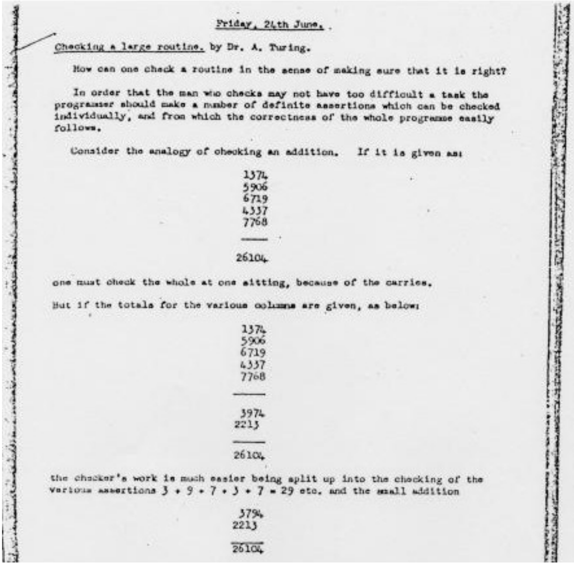
CPA✓

SoSy-Lab Software Systems

Some Historical Landmarks

- **70 years ago: Assertions and Proof Decomposition**, Alan Turing, 1949 [29]
  "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 [29]



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
2213
 ----
26104
```

Some Historical Landmarks

- 70 years ago: Assertions and Proof Decomposition
- **60 years ago: Logic Interpolation**,
  William Craig, J. Symb. Log. 1957 [22]
  "Linear reasoning. A new form of the Herbrand-Gentzen theorem."
  Defines an interpolation for logic formulas
  Made popular by Ken McMillan [28]

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 [27]
  "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 [27]

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 [19]

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 [23]
  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 [24, 10]
  - 2004: SATABS [20]

  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, ... [21]

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

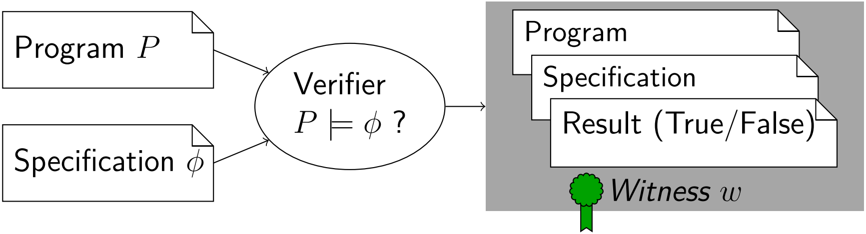
- 70 years ago: Assertions and Proof Decomposition
- 60 years ago: Logic Interpolation
- 50 years ago: Data-Flow Analysis and Abstract States
- 40 years ago: LTL and Model Checking
- 25 years ago: Predicate Abstraction
- 20 years ago: Tools for Software Model Checking
- 15 years ago: Satisfiability Modulo 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 [25]
- SV-COMP: off-site, automatic tools, controlled [4]
- Test-Comp: off-site, automatic tools, controlled [6]
- VerifyThis: on-site, interactive, teams [26]

(alphabetic order)

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

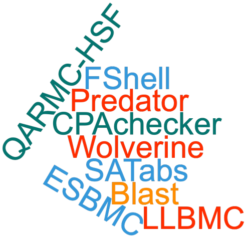
Theorem for P and S: $P \models S$
If true, a proof of correctness was constructed.
If false, a proof by counterexample was constructed.

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

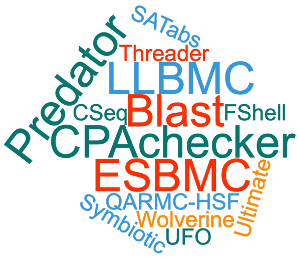
Note on **testing** and **BMC**:
$\Rightarrow$ verifiers for finding proofs by counterexample

Note on **explicit-state model checking**:
$\Rightarrow$ verifiers that ‘produce’ huge invariants

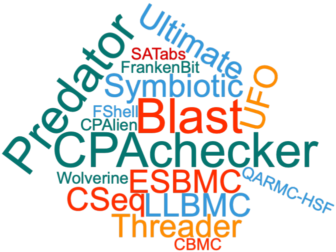
SV-COMP (Automatic Tools 2012)



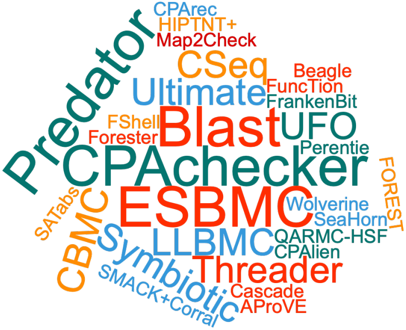
SV-COMP (Automatic Tools 2013, cumulative)



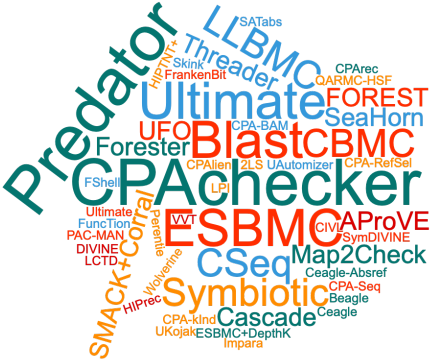
SV-COMP (Automatic Tools 2014, cumulative)



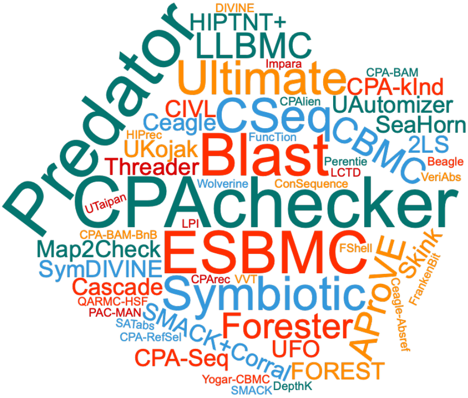
SV-COMP (Automatic Tools 2015, cumulative)



SV-COMP (Automatic Tools 2016, cumulative)

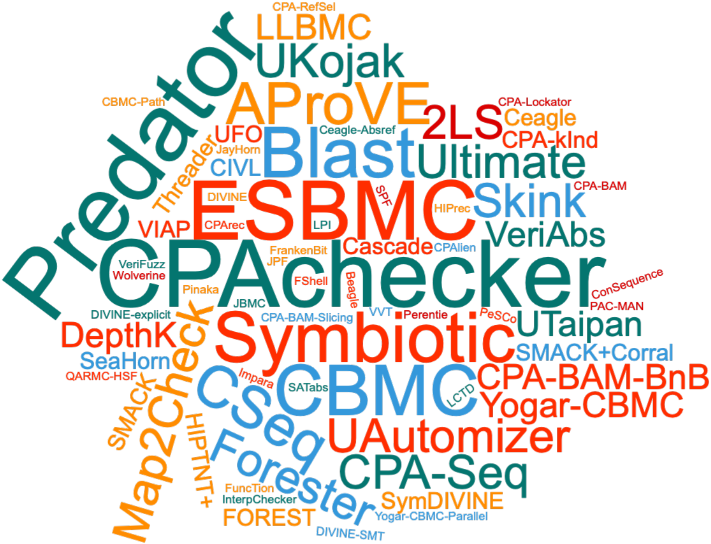


SV-COMP (Automatic Tools 2017, cumulative)



SV-COMP (Automatic Tools 2018, cumulative)

SV-COMP (Automatic Tools 2019, cumulative)

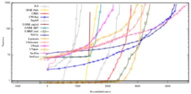


What is the best verifier?

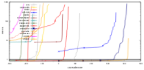
- Many different kinds of programs seem to require many different good tools with different strengths

SV-COMP (Automatic Tools)

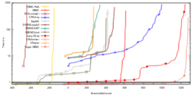
ReachSafety
1. **VeriAbs**
2. **CPA-Seq**
3. **PeSCo**



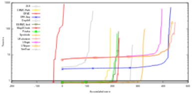
MemSafety
1. **Symbiotic**
2. **PredatorHP**
3. **CPA-Seq**



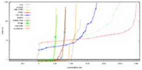
ConcurrencySafety
1. **Yogar-CBMC**
2. **Lazy-CSeq**
3. **CPA-Seq**



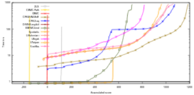
NoOverflows
1. **UAutomizer**
2. **UTaipan**
3. **CPA-Seq**



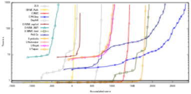
Termination
1. **UAutomizer**
2. **AProVE**
3. **CPA-Seq**



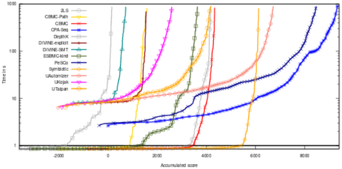
SoftwareSystems
1. **CPA-BAM-BnB**
2. **CPA-Seq**
3. **VeriAbs**



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	✓	✓									✓		✓	✓				

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

Algorithms

17	Bounded Model Checking
13	CEGAR
8	Predicate Abstraction
5	k-Induction
4	Symbolic Execution
3	Automata-Based Analysis
2	Property-Directed Reachability (IC3)

Abstract Domains

24	Bit-Precise Analysis
10	Explicit-Value Analysis
9	Numerical Interval Analysis
4	Shape Analysis
1	Separation Logic

Testing

- Fuzzing (VeriFuzz [18], based on AFL)
- Symbolic execution (KLEE [17])
- Software model checking (CoVeriTest [13])

SMT-based Software Model Checking

- Predicate Abstraction
  (Blast, CPAchecker, Slam, ...)
- Impact
  (CPAchecker, Impact, Wolverine, ...)
- Bounded Model Checking
  (Cbmc, CPAchecker, Esbmc, ...)
- *k*-Induction
  (CPAchecker, Esbmc, 2ls, ...)
- Property-Directed Reachability (PDR, also known as IC3)
  (CPAchecker, SeaHorn, Vvt, ...)
- Trace Abstraction
  (Ultimate Automizer, CPAchecker in progress, ...)

Unification Efforts

A Unifying View on SMT-Based Software Verification

Dirk Beyer, Matthias Dangl, Philipp Wendler

Journal of Automated Reasoning, 2018 [8]

based on

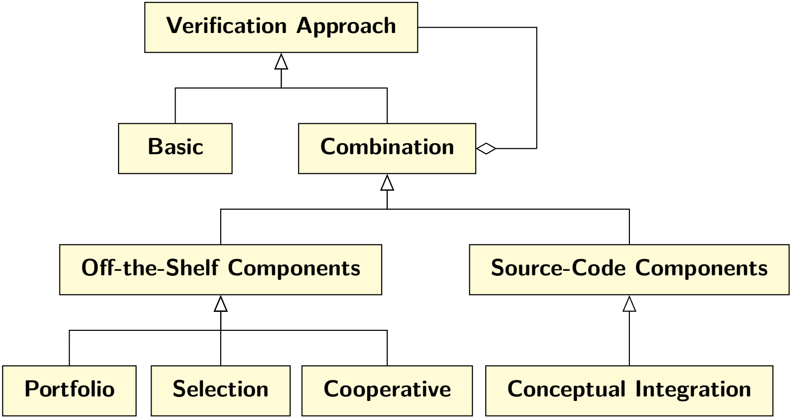
- Configurable Program Analysis [11, 12]
- Large-Block Encoding [7, 15]
- Satisfiability Modulo Theories [3]

Combining Model Checking and Data-Flow Analysis

Dirk Beyer, Sumit Gulwani, David Schmidt

Handbook of Model Checking, 2018 [9]

Combinations and Cooperation [16]



Decompose Software Verification [14]

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

Conclusion

- 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

References II

[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

[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., 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

[8] 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

[9] 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

[10] 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

[11] 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

[12] 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

References IV

[13] Beyer, D., Jakobs, M.C.: CoVeriTest: Cooperative verifier-based testing. In: Proc. FASE. pp. 389–408. LNCS 11424, Springer (2019). https://doi.org/10.1007/978-3-030-16722-6_23

[14] 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

[15] Beyer, D., Keremoglu, M.E., Wendler, P.: Predicate abstraction with adjustable-block encoding. In: Proc. FMCAD. pp. 189–197. FMCAD (2010)

[16] 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

References V

[17] Cadar, C., Dunbar, D., Engler, D.R.: KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proc. OSDI. pp. 209–224. USENIX Association (2008)

[18] Chowdhury, A.B., Medicherla, R.K., Venkatesh, R.: VERIFUZZ: Program-aware fuzzing (competition contribution). In: Proc. TACAS (3). pp. 244–249. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_22

[19] 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

[20] 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

References VI

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

[22] 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

[23] 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

[24] 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

[25] 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

References VII

[26] 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

[27] 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

[28] 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

[29] 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)