Software Verification

Historical Landmarks and Current Developments

Dirk Beyer
LMU Munich, Germany
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.”
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 program 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
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,
  → software model checking

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.
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 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
Which techniques are used?

<table>
<thead>
<tr>
<th>Participant</th>
<th>CEGAR</th>
<th>Predicate Abstraction</th>
<th>Symbolic Execution</th>
<th>Bounded Model Checking</th>
<th>k-Induction</th>
<th>Property-Directed Reach.</th>
<th>Explicit-Value Analysis</th>
<th>Numeric Interval Analysis</th>
<th>Shape Analysis</th>
<th>Separation Logic</th>
<th>Bit-Precise Analysis</th>
<th>ARC-Based Analysis</th>
<th>Lazy Abstraction</th>
<th>Interpolation</th>
<th>Automata-Based Analysis</th>
<th>Concurrency Support</th>
<th>Ranking Functions</th>
<th>Evolutionary Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>2LS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>AProVE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>CBMC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>CBMC-Path</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>CPA-BAM-BnB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>CPA-Lockator</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>CPA-Seq</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>DepthK</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>DIVINE-EXPLICIT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>DIVINE-SMT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>ESBMC-kind</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>JAYHORN</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>JBMC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>JPf</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Lazy-CSiq</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Map2Check</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>PeSeCo</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Pinaka</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>PredatorHP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Skink</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Smack</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>SPF</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>SYMBIOTIC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>UAutomizer</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>UKojal</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>UTobisk</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

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]

Verification Approach

Basic

Combination

Off-the-Shelf Components

- Portfolio
- Selection
- Cooperative

Source-Code Components

- Conceptual Integration
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 II

https://doi.org/10.1007/978-3-030-17502-3_9

https://doi.org/10.1007/978-3-030-17502-3_11

https://doi.org/10.1109/FMCAD.2009.5351147

https://doi.org/10.1007/s10817-017-9432-6
References III

https://doi.org/10.1007/978-3-319-10575-8_16

https://doi.org/10.1007/s10009-007-0044-z


References V


References VI


