Check for updates

Software Model Checking: 20 Years and Beyond

Dirk Beyer[1(✉)] and Andreas Podelski[2]

[1] LMU Munich, Munich, Germany
Dirk.Beyer@sosy-lab.org
[2] University of Freiburg, Freiburg im Breisgau, Germany

Abstract. We give an overview of the development of software model checking, a general approach to algorithmic program verification that integrates static analysis, model checking, and deduction. We start with a look backwards and briefly cover some of the important steps in the past decades. The general approach has become a research topic on its own, with a wide range of tools that are based on the approach. Therefore, we discuss the maturity of the research area of software model checking in terms of looking at competitions, at citations, and most importantly, at the tools that were build in this area: we count 76 verification systems for software written in C or Java. We conclude that software model checking has quickly grown to a significant field of research with a high impact on current research directions and tools in software verification.

Keywords: History · Software Verification · Programming · Formal Methods · Program Correctness · Automatic Verification · Verification Tools · Provers

1 Introduction

This paper is meant as a journey through the development of a technology that started as a completely intractable endeavor and now plays a key role in the success of various commercial projects (e.g., [10,14,51,69,125]).

We contribute this report to the Festschrift for Tom Henzinger, who has influenced the development in several ways. In particular, he led the Blast project with Ranjit Jhala and Rupak Majumdar, and he pushed for the convergence of data-flow analysis, model checking, and software testing.

Dirk came to UC Berkeley as a young postdoc to join Tom's group. Dirk thought that he would work on topics such as timed and hybrid systems, but Tom had asked him whether he would have a problem with working on software model checking instead. He became immediately infected with the charm of the Blast project and since then he has never stopped working in this area, instantiating some of the joint ideas in the CPAchecker project and in the competition on software verification.

Andreas would discuss possible approaches with Tom in an earlier period of time, when software model checking did not exist yet but many people thought about it. Andreas distinctly remembers one discussion in front of the coffee machine at the Max Planck Institute for Computer Science in Saarbrücken,

J.-F. Raskin and K. Chatterjee (Eds.): Principles of Systems Design, LNCS 13660, pp. 554–582, 2022.
https://doi.org/10.1007/978-3-031-22337-2_27

when he and Tom concluded that abstracting a program to a finite-state system seemed a bad idea (Tom, matter-of-factly: "a loser right from the start"). How inspiring a bad idea can be.

2 Timeline of Formal Verification of Software

This section outlines a few milestones in the area of software verification which we think were instrumental to the success and led to the breakthrough in technology.

2.1 Before 1962

First Insights (1880–1940). Mathematicians were concerned with the verification of consistency of arithmetic axioms since a long time. Giuseppe Peano described an arithmetic system of axioms [158], and David Hilbert was interested to know whether a contradiction can be generated after finitely many proof steps [113, 2nd problem]. The dream of a machine that can generate all truth ended after only a few decades, when Kurt Gödel showed that there are certain theorems that cannot be proven [101], and Alonzo Church and Alan Turing showed that our abilities to prove the correctness of programs are limited [62,191]. Despite these initial 'bad' news, software verification can solve many interesting and practical problems. One of the approaches is to restrict the proof system to a decidable theory, for which Presburger arithmetics [167] is a prominent example, which is still often-used today.

Computing Machinery (1940s). Z3, the first working digital, automatic, and programmable computer, was constructed by Konrad Zuse and was ready to be used in 1941. It was a binary computer, built using relays. The second computer, ENIAC, was completed in 1944, based on vacuum tubes. Leibniz and Babbage also constructed computers, but they were not digital, automatic, and programmable. The unavailability of good hardware foundations had hindered the development of computers for a long time. Not only the hardware foundations were missing as an enabling technology: The enabling theories in logics were also not yet sufficiently developed. Predicate logic was needed to prove that the halting problem of Turing machines is undecidable [191]. In parallel, Shannon showed how to implement Boolean algebra using electric circuits [185], which is still how computers are built today (just using transistors instead of relays and somewhat smaller).

Assertions, Proof Decomposition, and Abstraction. As early as 1949, Alan Turing published a method —based on *assertions*— to prove the correctness of computer programs [192]. He wrote: *"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."* Assertions are nowadays one of the most common notations to write invariants in software development.

Abstraction was considered the key for proving correctness, and assertions are abstracting the states at a certain location in the program. Konrad Zuse understood that programming requires abstract languages, and developed the first high-level programming language designed for a computer (Plankalkül [180]).

Craig Interpolation (1950s). But how to automatically compute abstractions? William Craig defined interpolation for logic formulas in 1957 [75]. Given two formulas ϕ_1 and ϕ_2 such that ϕ_1 implies ϕ_2, an interpolant for ϕ_1 and ϕ_2 is a formula ψ that is implied by ϕ_1 and that implies ϕ_2, and contains only symbols that occur in both ϕ_1 and ϕ_2. Applied to program verification, if ϕ_1 represents a program path and ϕ_2 represents a safety property, then the interpolant ψ is an abstraction of the program path ϕ_1 that (1) can be automatically constructed and (2) makes it potentially easier to prove that the property ϕ_2 holds.

2.2 After 1962

Decision Procedures (1960s). The advent of programmable computers and continuous advancements of the theory made it possible to implement automatic theorem proving [79,80,95,166,170]. The algorithm of Davis, Putnam, Logemann, and Loveland is still used today and led to the notion of *decision procedures*. Such decision procedures where further extended to combinations with other theories [153,154] and led to the theorem prover Simplify [81], which was used as backend in the Extended Static Checkers ESC/Java [88] and ESC/Modula-3 [139].

Program Correctness. In the 1960s, the availability of computers led to an enormous growth of software production. At the same time, the fundamental principles of programming and engineering of large software systems were not yet sufficiently studied. The term *software engineering* was established and a conference held: the first NATO Software-Engineering Conference took place 1968 in Garmisch, Germany. One of the solutions was to support the software development by *formal methods* [82,83,89,114,143,200], which were establishing mathematically precise foundations of computer programming.

Data-Flow Analysis and Abstract States (1970s). In his famous POPL 1973 paper "A Unified Approach to Global Program Optimization" [126], Gary Kildall provided many of the technical ingredients of data-flow analysis that we still use today (fixed-point iteration, lattice operations, ...). The mathematical foundation for more general forms of program analysis was then given by Patrick and Radhia Cousot [74]. The general idea is to define an abstract domain via a lattice and then compute a fixed point in order to construct an overapproximation of the behavior of the program (see also [155]).

LTL and Model Checking (1980s). Zohar Manna, Amir Pnueli, Ed Clarke, Allen Emerson, and Joseph Sifakis contributed theoretical and conceptional foundations to the verification of systems (not only software systems), leading to the notion of *model checking* [68,169]. Manna and Pnueli developed LTL as a specification language and used it to formally specify the behavior of a system using temporal logic [142]. Tools based on model checking became more and more important. Binary decision diagrams [3,137] were extended to their shared and reduced versions by Randy Bryant [47], and he introduced BDDs as a data structure with a wide applicability in formal methods. For many years, the article by Randy Bryant was the most cited article in computer science. We refer to the Handbook of Model Checking [67] for overviews on specific topics on model checking, specifically temporal logic [160] and binary decision diagrams [48].

Symbolic Model Checking (1990s). While the 80s produced many of the theoretical foundations, the 90s brought verification algorithms to practice. Ken McMillan introduced BDDs as the data structure for symbolic model checking [50,144]. BDDs and other symbolic state-space representations became an enabling technology to verify large systems.

Predicate Abstraction. In 1997, as a step towards connecting model checking with program verification, Susanne Graf and Hassen Saïdi developed a deduction-based method to partition the state space of a program according to an equivalence relation defined by a given finite set of state predicates [96]. We obtain a finite abstract system if we associate each block in the partition with an abstract state. The abstract system contains a transition between two blocks if the program has a transition between a state from one block to a state from the other block. If the state predicates and the program's transition relation are represented by logical formulas, then the existence of such states in each of the two blocks reduces to the satisifiability of a logical formula (and this is how deduction comes in).

2.3 Software Model Checking

Tools for Software Model Checking (2000s). The time was ripe for software model checking. In summer 2000, Tom Ball and Sriram Rajamani, with help from others, notably Rupak Majumdar and Todd Millstein, developed SLAM [10,11,14], a tool that performs an *abstraction-refinement loop*. In each iteration of the loop, the tool, using a first-order logic theorem prover, abstracts the given C program (with procedures, possibly recursive) for a given set of predicates. If an error path is found, it checks the feasibility of the sequence of transitions that corresponds to the error path in the abstract system by checking satisifiability. If the error path is infeasible, it uses the proof of unsatisfiability to derive new predicates for the refined abstraction in the next iteration of the loop. The notion of **counterexample-guided abstraction refinement (CEGAR)** was born, developed around the same time in the context of software programs [13] and in the context of finite-state systems [65,66]. In fall 2000, Tom Henzinger, Ranjit Jhala, Rupak Majumdar, and Grégoire Sutre, with help from others, developed BLAST [32,111,112], a tool that implements a similar abstraction refinement but circumvents the abstraction of the whole C program by *lazily* constructing an abstract reachability tree.

These early developments received a lot of attention, and software model checking became a research topic on its own. The SLAM project paved the road for the success of software model checking in industrial software development. The success of SLAM is witnessed by the Static Driver Verifier project[1], which is based

[1] https://www.microsoft.com/en-us/research/project/slam

on Slam and was used as part of Microsoft's Windows Driver Development Kit in daily software production. The Blast project showed the effects of applying Craig interpolation to the abstraction-refinement process in program analysis [111], first using McMillan's original FOCI library [146] and later the independently developed SMT solver CSIsat [40]. Later versions of Blast, which was by that time maintained by a different group [188], received gold medals in the category *Device Drivers* of the competition on software verification[2] in 2012, 2014, and 2015. Both projects were highly influencial in the research community: the PLDI '01 paper on Slam [11] received a PLDI test-of-time award in 2011[3], and the POPL '04 paper on Blast [111] received a POPL test-of-time award in 2014[4].

Also, as a sign of maturity, survey papers appeared, on software verification [85], on software model checking [121], and on deductive verification [19]. A recent survey addresses the current status of formal methods [92], and competition reports give an overview over the status of tools for software verification [30].

Satisfiability Modulo Theory. In the early 2000s, there was an enormous progress in research on satisfiability (SAT), with the appearance of efficient implementations of algorithms for SAT solving, most notably Chaff [149]. Theory combinations led to the notion of *satisfiability modulo theories* (SMT), an integration of SAT with theories like linear arithmetics, bitvectors, and arrays. The SMTLIB format for input formulas [16] facilitated the use of SMT tools. Some SMT solvers support interpolation; examples are CSIsat [40], MathSAT [46], and SMTInterpol [61].

Boolean and Cartesian Abstraction. At the beginning, when predicate abstraction was first used for the abstraction of C programs (by Slam and Blast), it was implemented by Cartesian abstraction [12]. At that time, disjunctions were not efficiently supported by automatic solvers such as Simplify [81]. Only later, interpolating SMT solvers such as CSIsat [40] and MathSAT [46] could handle disjunctions efficiently. Cartesian predicate abstraction seemed suitable as long as only simple program paths were encoded in path formulas. In connection with *large-block encoding* [31,36], however, when it was possible to delegate large amounts of work (i.e., large formulas) to the SMT solvers, it became feasible to use Boolean abstraction [129] to implement predicate abstraction, as done in CPAchecker [35].

Verification with Interpolants. Ken McMillan published how to use Craig interpolation [75] for finding abstract descriptions of the behavior of transition systems [145]. Later, Craig interpolation was also applied to program paths, in order to automatically learn abstractions for the verification of computer programs [111]. For every program path we can construct a formula such that the program path is infeasible (there is no execution of all statements along the path) if and only if the formula is unsatisfiable. Assume that we have split a given infeasible program path at a certain program location, and the path prefix and the path suffix correspond to the path formulas ϕ^{pre} and ϕ^{post}, respectively. Then ϕ^{pre} implies the

[2] https://sv-comp.sosy-lab.org
[3] https://www.sigplan.org/Awards/PLDI
[4] https://www.sigplan.org/Awards/POPL

negation of ϕ^{post}. The interpolant $\psi = itp(\phi^{pre}, \neg\phi^{post})$ represents an abstraction; i.e., it describes what we need to know about the states after executing the path prefix in order to derive that continuing the execution of the path suffix is not possible. Ken McMillan also developed the first tool for automatically computing interpolants [146]. An overview on the use of interpolation for verification is given in a chapter in the Handbook on Model Checking [147].

Trace Abstraction. As an alternative to constructing a sequence of (more and more refined) abstractions (abstract systems or abstract reachability graphs), the approach of trace abstraction [108] is to construct a sequence of programs until all paths of the input program are covered. Each program in the sequence is constructed from the proof of the infeasibility of a (spurious) counterexample. The covering check can be reduced to automata inclusion.

Termination. After a series of breakthroughs in making safety analysis of large software systems practically relevant, also liveness properties were investigated. Algorithmic approaches for constructing ranking functions [161] made it possible to perform termination analysis. Since termination of functions in the operating system are a major concern, e.g., for Microsoft, tool support for termination analysis [70, 162] became important.

Competition on Software Verification (2010s). In order to make progress explicit and show that there are many good tools for software verification available, a competition on software verification (SV-COMP) was developed 2010–2011, with the first results published in 2012 [20]. Such competitions create awareness of tools and the available technology, provide comparative evaluations, and establish standards (e.g., input formats, results formats, comparability, reproducibility). The most recent instance of the competition evaluated 47 verification tools.

Property-Directed Reachability. Property-directed reachability (PDR) is a SAT/SMT-based reachability algorithm that incrementally constructs inductive invariants. After it was successfully applied to hardware model checking [43, 44], several adaptations to software model checking have been proposed [42, 63, 64, 130, 131].

Interpolation-Based Model Checking. While interpolation became a key ingredient in many verification approaches for software, the original algorithm from 2003 [145] was adopted to the verification of software only recently [37].

Approaches Used in Tools. Current tools usually combine a set of approaches. We report in Table 4 which approaches are used by tools for software verification.

2.4 Current Developments (2022)

While the focus of the past decades was on contributing tools that implement verification approaches in order to make the research results practically usable, we today observe a move from a *lack* of tools to an *abundance* of tools (see Sect. 3.3).

A new research question arises in this context: How can we integrate existing verification systems in order to maximally benefit from their respective strengths. To enable cooperation between verification tools, we need standardized interfaces that make it possible to pass artifacts with valuable information from one tool to another. Such verification artifacts include, besides the programs and their specifications, also transformed or reduced programs, error paths, invariants, witnesses, and partial verification results in general [39,41].

3 Maturity of the Research Area

The area of software model checking is 22 years old at the time of writing, and several aspects indicate that software model checking is a mature research area. We outline a few such indicators in the following.

3.1 Competitions

It is well understood that competitions are an important scientific method. Competitions provide regular comparative evaluations. In the area of formal methods, there are plenty of competitions [17], most of them being concerned with comparisons of tools that solve a certain kind of problem, most prominently, SAT and SMT solving. Five competitions are concerned with the verification of software: RERS, SV-COMP, Test-Comp, VerifyThis, and TermComp (Table 1).

Table 1. Competitions in the area of software verification

Competition	Where	What	How	Reference
RERS	off-site	tools	open	[118]
SV-COMP	off-site	automatic tools	controlled	[20]
Test-Comp	off-site	automatic tools	controlled	[27]
VerifyThis	on-site	teams and tools	interactive	[119]
TermComp	off-site	automatic tools	controlled	[94]

The Competition on Software Verification (SV-COMP) provides annually a comparative evaluation of automatic tools for software verification. The first results were published in 2012 [20]. The objectives of the competition include:

- create awareness of tools,
- provide yearly comparative evaluations,
- create and maintain a benchmark collection (SV-Benchmarks repository),
- establish standards (input, exchange, comparability, reproducibility),
- conserve tools at a central place and make them available, and
- educate PhD students and postdocs on benchmarking and reproducibility.

The competition was a success, in that all of the above-mentioned objectives were achieved. Over the last ten years, more and more verification tools participated, and the last edition was comparing 47 verification tools.

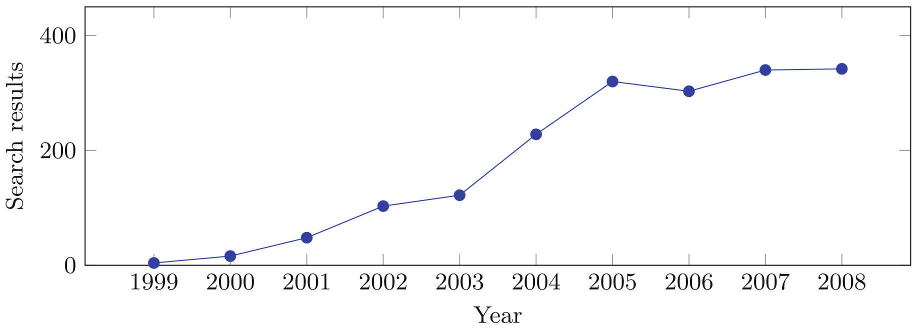

Fig. 1. Number y of search results found by Google Scholar for "software model checking" per year x; illustrates growing interest in the topic in the first 10 years

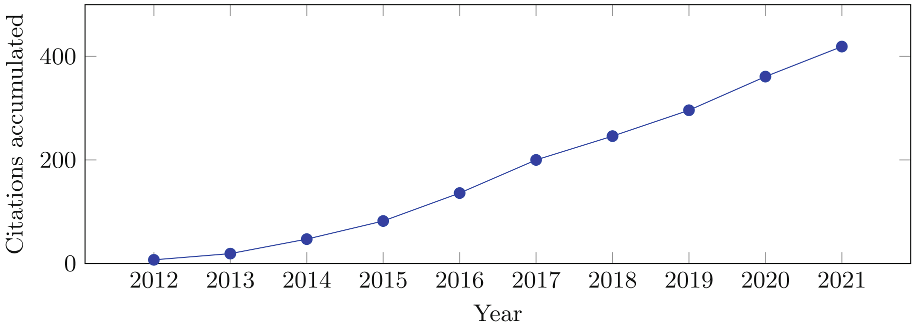

Fig. 2. Number y of citations up to year x of SV-COMP reports according to the COCI CSV data set [157]; illustrates constant interest in verifier competitions

3.2 Publication Venues and Research Activity

Software model checking is a research field at the intersection of programming languages, software engineering, and theory of computation. Thus, the research results are mainly published in outlets in the area of programming languages, such as POPL, PLDI, and OOPSLA, of software engineering, such as ICSE, ESEC/FSE, ASE, and ISSTA, and of formal methods, such as CAV, TACAS, and ATVA.

Figure 1 illustrates the development of the research area. We created a mapping from each year in the range 1999–2008 to the number of search results of Google Scholar for the search term "software model checking" in its first 10 years. The graph drawn in Fig. 1 illustrates how the interest in software model checking was growing in the early 2000s s years, and how it stabilized afterwards.

As mentioned above, the competition SV-COMP serves as a platform for creation and maintenance of benchmark sets and community standards. To illustrate the continuous interest in the topics of the competition, we counted the number y of citations up to year x and draw the function in Fig. 2. We used COCI [106], an open citation index that is regularly extended by OpenCitations.

We used the data set version 16 (2022-08-31) [157], and counted the number of citations of any of the SV-COMP reports [20–26,28–30].

3.3 Verification Tools and Artifacts

The research community developed new approaches, and implemented them in readily available tools. As shown by the competition SV-COMP, there are many verification systems available. Table 2 illustrates the rich set of verification tools, by listing the tool names, the language that they are mainly used for, references to literature, contact persons, and the location where the tools are developed, maintained, and hosted. All listed tools participated at least once in the competition on software verification SV-COMP. This is also a sign of maturity: Researchers develop tool implementations and hand them in for evaluation. Table 3 shows which tool participated when in the competition. It is interesting to see that there are verification systems that are long-term maintained and participate often, and there are some research prototypes, made to explore an idea, participated once, and then abandoned. The overview in Table 4 shows that there are many different technologies implemented and used.

Table 2. Tools for software verification, with the progamming language for which they participated (J for Java), main references, contact, and origin (assembled from SV-COMP reports 2012–2022)

Verifier	L. Ref.	Contact	Location
2LS	C [45,141]	Viktor Malík	Brno, Czechia
AProVE	C [110,189]	Jera Hensel	Aachen, Germany
Beagle	C	Dexi Wang	Beijing, China
Blast	C [32,188]	Vadim Mutilin	Moscow, Russia
Brick	C [49]	Lei Bu	Nanjing, China
Cascade	C [198]	Wei Wang	New York, USA
Cbmc	C [127]	Michael Tautschnig	London, UK
Ceagle	C	Guang Chen	Beijing, China
Civl	C [203]	Stephen Siegel	Newark, USA
Coastal	J [194]	Willem Visser	Stellenbosch, South Africa
Consequence	C	Anand Yeolekar	Pune, India
CoVeriTeam	C [33,34]	Sudeep Kanav	Munich, Germany
CPAchecker	C [35,77]	Thomas Bunk	Munich, Germany
CPA-BAM	C [6,197]	Vadim Mutilin	Moscow, Russia
CPAlien	C [152]	Petr Muller	Brno, Czechia
CPALockator	C [7,8]	Pavel Andrianov	Moscow, Russia
CPArec	C [59]	Ming-Hsien Tsai	Teipei, Taiwan
Crux	C [84,183]	Ryan Scott	Portland, USA
CSeq	C [73,120]	Omar Inverso	L'Aquila, Italy
Dartagnan	C [93,163]	Hernán Ponce de León	Munich, Germany

(*continues on next page*)

Table 2. Tools for software verification (*continued*)

Verifier	L. Ref.	Contact	Location
DEAGLE	C [105]	Fei He	Bejing, China
DEPTHK	C [177,179]	Omar Alhawi	Manchester, UK
DIVINE	C [15,132]	Henrich Lauko	Brno, Czechia
EBF	C	Fatimah Aljaafari	Manchester, UK
ESBMC	C [90,91]	Rafael Sá Menezes	Manchester, UK
FOREST	C [5]	Pablo Sanchez	Santander, Spain
FORESTER	C [115]	Martin Hruska	Brno, Czechia
FRAMA-C-SV	C [38,76]	Martin Spiessl	Munich, Germany
FRANKENBIT	C [99]	Arie Gurfinkel	Pittsburgh, USA
FSHELL	C [117]	Helmut Veith	Vienna, Austria
FUNCTION	C [193]	Caterina Urban	Paris, France
GACAL	C [171]	Benjamin Quiring	Boston, USA
GAZER-THETA	C [1,103]	Ákos Hajdu	Budapest, Hungary
GDART	J [151]	Falk Howar	Dortmund, Germany
GOBLINT	C [181,196]	Simmo Saan	Tartu, Estonia
GRAVES-CPA	C [138]	Will Leeson	Charlottesville, USA
HIPREC	C [135]	Quang Loc Le	Singapore, Singapore
HIPTNT+	C [136]	Ton Chanh Le	Singapore, Singapore
HSF(C)	C [97]	Andrey Rybalchenko	Munich, Germany
IMPARA	C	Björn Wachter	Oxford, UK
INFER	C [52,124]	Thomas Lemberger	Munich, Germany
INTERPCHECKER	C	Zhao Duan	Xi'an, China
JAVA-RANGER	J [186,187]	Soha Hussein	Minnesota, USA
JAYHORN	J [122,184]	Ali Shamakhi	Tehran, Iran
JBMC	J [71,72]	Peter Schrammel	Sussex, UK
JDART	J [140,150]	Falk Howar	Dortmund, Germany
JPF	J [9,195]	Cyrille Artho	Stockholm, Sweden
KORN	C [86]	Gidon Ernst	Munich, Germany
LART	C [133,134]	Henrich Lauko	Brno, Czechia
LCTD	C [182]	Keijo Heljanko	Espoo, Finland
LLBMC	C [87]	Stephan Falke	Karlsruhe, Germany
LOCKSMITH	C [165]	Vesal Vojdani	Tartu, Estonia
LPI	C [123]	George Karpenkov	Grenoble, France
MAP2CHECK	C [176,178]	Herbert Rocha	Boa Vista, Brazil
PAC-MAN	C [58]	Ming-Hsien Tsai	Taipei, Taiwan
PERENTIE	C [53]	Franck Cassez	Sydney, Australia
PESCO	C [174,175]	Cedric Richter	Oldenburg, Germany
PINAKA	C [57]	Saurabh Joshi	Hyderabad, India
PREDATOR	C [116,159]	Veronika Šoková	Brno, Czechia
SATABS	C [18]	Michael Tautschnig	Oxford, UK
SEAHORN	C [100]	Jorge Navas	Mountain View, USA
SESL	C	Xie Li	Beijing, China

(*continues on next page*)

Table 2. Tools for software verification (*continued*)

Verifier	L. Ref.	Contact	Location
SKINK	C [54]	Franck Cassez	Sydney, Australia
SMACK	C [104, 173]	Zvonimir Rakamaric	Salt Lake City, USA
SPF	J [156, 168]	Willem Visser	New York, USA
SYMBIOTIC	C [55, 56]	Marek Chalupa	Brno, Czechia
THETA	C [190, 204]	Vince Molnár	Budapest, Hungary
THREADER	C [164]	Corneliu Popeea	Munich, Germany
UFO	C [4, 98]	Aws Albarghouthi	Toronto, Canada
ULTIMATE	C [107, 109]	Matthias Heizmann	Freiburg, Germany
VERIABS	C [2, 78]	Priyanka Darke	Pune, India
VERIFUZZ	C [60, 148]	Raveendra Kumar M.	Pune, India
VIAP	C [172]	Pritom Rajkhowa	Hong Kong, China
VVT	C [102]	Alfons Laarman	Vienna, Austria
WOLVERINE	C [128, 199]	Georg Weissenbacher	Vienna, Austria
YOGARCBMC	C [201, 202]	Liangze Yin	Bejing, China

Table 3. Participation in SV-COMP evaluations 2012–2022

Verifier	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022
2LS					■	■	■	■	■	■	■
APROVE				■	■	■	■	■			■
BEAGLE				■							
BLAST	■	■	■	■	■	■					
BRICK									■	■	■
CASCADE				■	■						
CBMC			■	■	■	■	■	■	■	■	■
CEAGLE					■	■					
CIVL					■	■					
COASTAL									■	■	■
CONSEQUENCE						■					
COVERITEAM											■
CPACHECKER	■	■	■	■	■	■	■	■	■	■	■
CPA-BAM	■				■	■	■	■	■	■	■
CPALIEN			■								
CPALOCKATOR								■	■	■	■
CPAREC				■							
CRUX											■
CSEQ		■	■	■	■	■		■	■	■	■
DARTAGNAN									■	■	■
DEAGLE											■
DEPTHK						■	■	■			
DIVINE					■	■		■	■	■	■

(*continues on next page*)

Table 3. Participation in SV-COMP evaluations 2012–2022 (*continued*)

Verifier	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022
Ebf											■
Esbmc	■	■	■	■	■	■	■	■	■	■	■
Forest				■	■						
Forester				■	■	■	■				
Frama-C-SV										■	■
FrankenBit			■								
FShell	■										
FuncTion				■							
Gacal									■		
Gazer-Theta									■	■	■
GDart											■
Goblint										■	■
Graves-CPA											■
HIPrec					■						
HipTNT+				■		■					
Hsf(C)	■										
Impara					■						
Infer											■
InterpChecker							■				
Java-Ranger									■	■	■
JayHorn								■	■	■	■
Jbmc								■	■	■	■
JDart									■	■	■
Jpf								■			
Korn										■	■
Lart											■
Lctd					■						
Llbmc	■	■	■								
Locksmith											■
Lpi					■						
Map2Check				■	■		■	■	■		
Pac-Man					■						
Perentie				■							
PeSCo								■	■	■	■
Pinaka								■	■	■	■
Predator	■	■	■	■	■	■	■	■	■	■	■
SATabs	■										
SeaHorn				■	■						
Sesl											■
Skink					■	■	■	■			
Smack				■	■	■		■		■	■
Spf								■	■	■	■

(*continues on next page*)

Table 3. Participation in SV-COMP evaluations 2012–2022 (*continued*)

Verifier	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022
Symbiotic		✓	✓		✓	✓	✓	✓	✓	✓	✓
Theta											✓
Threader		✓	✓								
Ufo		✓	✓								
Ultimate		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
VeriAbs						✓	✓	✓	✓	✓	✓
VeriFuzz								✓	✓		✓
VIAP							✓	✓			
VVT					✓						
Wolverine	✓										
YogarCBMC						✓	✓	✓	✓		

Table 4. Algorithms and techniques, by verifier (assembled from SV-COMP reports)

Verifier	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	Algorithm Selection	Portfolio
2LS				✓	✓			✓	✓		✓						✓			
AProVE			✓				✓	✓		✓	✓						✓			
Beagle	✓	✓		✓																
Blast	✓	✓					✓					✓	✓	✓						
Brick	✓		✓	✓				✓								✓				
Cascade			✓	✓							✓									
CBMC				✓							✓					✓				
Ceagle	✓	✓		✓							✓	✓	✓							
CIVL			✓	✓				✓								✓				
Coastal			✓																	
Consequence				✓							✓					✓				
CoVeriTeam	✓	✓	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓		✓	✓
CPAchecker	✓	✓	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓		✓	✓		✓	✓
CPA-BAM	✓	✓					✓				✓	✓	✓	✓						
CPALien							✓		✓											
CPALockator	✓	✓					✓				✓	✓	✓	✓		✓				
CPARec	✓	✓										✓	✓	✓						
Crux			✓																	

(*continues on next page*)

Table 4. Algorithms and techniques (*continued*)

Verifier	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	Algorithm Selection	Portfolio
CSeq				✓							✓					✓				
Dartagnan				✓							✓					✓				
Deagle																				
DepthK				✓	✓						✓					✓				
Divine			✓				✓				✓					✓			✓	✓
Ebf				✓																
Esbmc				✓	✓			✓			✓					✓				
Forest			✓	✓							✓									
Forester	✓								✓						✓					
Frama-C-SV								✓												
FrankenBit				✓							✓			✓						
FShell				✓																
FuncTion								✓									✓			
Gacal																				
Gazer-Theta	✓	✓		✓			✓				✓	✓	✓	✓						✓
GDart			✓								✓									✓
Goblint								✓								✓				
Graves-CPA	✓	✓		✓	✓		✓	✓	✓		✓	✓	✓	✓		✓	✓		✓	✓
HIPRec									✓	✓										
HipTNT+									✓	✓							✓			
Hsf(C)	✓	✓										✓		✓		✓				
Impara			✓				✓	✓			✓	✓	✓	✓		✓				
Infer								✓	✓	✓										✓
InterpChecker																				
Java-Ranger			✓								✓									
JayHorn	✓	✓				✓		✓					✓	✓						
Jbmc				✓							✓					✓				
JDart			✓								✓									✓
Jpf				✓			✓	✓			✓					✓				
Korn		✓	✓				✓													✓
Lart			✓				✓				✓									✓
Lctd	✓	✓	✓								✓									
Llbmc				✓																
Locksmith																✓				
Lpi	✓				✓			✓				✓	✓							

(*continues on next page*)

Table 4. Algorithms and techniques (*continued*)

Verifier	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	Algorithm Selection	Portfolio
Map2Check				✓							✓									
Pac-Man			✓												✓					
Perentie	✓		✓				✓							✓	✓					
PeSCo	✓	✓		✓	✓		✓	✓	✓		✓	✓	✓	✓		✓	✓		✓	✓
Pinaka			✓	✓							✓									
Predator									✓											
SATabs	✓	✓														✓				
SeaHorn				✓		✓		✓				✓	✓	✓			✓			
Sesl			✓							✓										
Skink	✓						✓							✓	✓					
Smack				✓							✓		✓			✓				
Spf			✓						✓							✓				
Symbiotic			✓		✓			✓	✓		✓					✓				✓
Theta	✓	✓					✓				✓	✓		✓		✓			✓	✓
Threader	✓	✓										✓		✓		✓				
Ufo	✓	✓		✓				✓				✓	✓	✓						
Ultimate	✓	✓					✓	✓			✓		✓	✓	✓	✓	✓		✓	✓
VeriAbs	✓			✓	✓		✓	✓										✓	✓	✓
VeriFuzz				✓				✓										✓		
Viap																				
Vvt	✓	✓		✓		✓	✓							✓		✓				
Wolverine	✓											✓	✓	✓						
YogarCbmc	✓			✓							✓		✓			✓				

4 Conclusion

We have given an overview over several mile stones in the history and the development of software verification, and have illustrated the maturity of the research area. This report also show-cases the research area by providing a comprehensive collection of competition-evaluated verification systems for the programming languages C and Java. We will not speculate about the future of software verification, but current trends are concerned with, for example, verification witnesses, concurrent programs, unbounded parallelism, termination, cooperative verification, machine-learning-based invariant generation, hyper-properties, and quantum programs.

Funding Information. This project was funded in part by the Deutsche Forschungsgemeinschaft (DFG) – 378803395 (ConVeY).

Data Availability Statement. The search results in Fig. 1 were manually looked up from Google Scholar on 2022-04-30. The citation counts in Fig. 2 were calculated from the COCI CSV data set [157]. The data to assemble Tables 2, 3 and 4 were looked up from SV-COMP reports [20–26, 28–30] and the SV-COMP web site at https://sv-comp.sosy-lab.org.

References

1. Ádám, Zs., Sallai, Gy., Hajdu, Á.: Gazer-Theta: LLVM-based verifier portfolio with BMC/CEGAR (competition contribution). In: Proc. TACAS (2). pp. 433–437. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_27
2. Afzal, M., Asia, A., Chauhan, A., Chimdyalwar, B., Darke, P., Datar, A., Kumar, S., Venkatesh, R.: VeriAbs: Verification by abstraction and test generation. In: Proc. ASE. pp. 1138–1141 (2019). https://doi.org/10.1109/ASE.2019.00121
3. Akers, S.B.: Binary decision diagrams. IEEE Trans. Computers **27**(6), 509–516 (1978). https://doi.org/10.1109/TC.1978.1675141
4. Albarghouthi, A., Li, Y., Gurfinkel, A., Chechik, M.: Ufo: A framework for abstraction- and interpolation-based software verification. In: Proc. CAV, pp. 672–678. LNCS 7358, Springer (2012). https://doi.org/10.1007/978-3-642-31424-7_48
5. de Aledo, P.G., Sanchez, P.: Framework for embedded system verification (competition contribution). In: Proc. TACAS. pp. 429–431. LNCS 9035, Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_36
6. Andrianov, P., Friedberger, K., Mandrykin, M.U., Mutilin, V.S., Volkov, A.: CPA-BAM-BnB: Block-abstraction memoization and region-based memory models for predicate abstractions (competition contribution). In: Proc. TACAS. pp. 355–359. LNCS 10206, Springer (2017). https://doi.org/10.1007/978-3-662-54580-5_22
7. Andrianov, P., Mutilin, V., Khoroshilov, A.: CPALockator: Thread-modular approach with projections (competition contribution). In: Proc. TACAS (2). pp. 423–427. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_25
8. Andrianov, P.S.: Analysis of correct synchronization of operating system components. Program. Comput. Softw. **46**, 712–730 (2020). https://doi.org/10.1134/S0361768820080022
9. Artho, C., Visser, W.: Java Pathfinder at SV-COMP 2019 (competition contribution). In: Proc. TACAS (3). pp. 224–228. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_18
10. 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
11. Ball, T., Majumdar, R., Millstein, T., Rajamani, S.K.: Automatic predicate abstraction of C programs. In: Proc. PLDI. pp. 203–213. ACM (2001). https://doi.org/10.1145/378795.378846

12. Ball, T., Podelski, A., Rajamani, S.K.: Boolean and Cartesian abstraction for model checking C programs. In: Proc. TACAS. pp. 268–283. LNCS 2031, Springer (2001). https://doi.org/10.1007/3-540-45319-9_19
13. Ball, T., Rajamani, S.K.: Boolean programs: A model and process for software analysis. Tech. Rep. MSR Tech. Rep. 2000-14, Microsoft Research (2000). https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-2000-14.pdf
14. Ball, T., Rajamani, S.K.: The Slam project: Debugging system software via static analysis. In: Proc. POPL. pp. 1–3. ACM (2002). https://doi.org/10.1145/503272.503274
15. Baranová, Z., Barnat, J., Kejstová, K., Kučera, T., Lauko, H., Mrázek, J., Ročkai, P., Štill, V.: Model checking of C and C++ with Divine 4. In: Proc. ATVA. pp. 201–207. LNCS 10482, Springer (2017). https://doi.org/10.1007/978-3-319-68167-2_14
16. 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/
17. Bartocci, E., Beyer, D., Black, P.E., Fedyukovich, G., Garavel, H., Hartmanns, A., Huisman, M., Kordon, F., Nagele, J., Sighireanu, M., Steffen, B., Suda, M., Sutcliffe, G., Weber, T., Yamada, A.: TOOLympics 2019: An overview of competitions in formal methods. In: Proc. TACAS (3). pp. 3–24. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_1
18. Basler, G., Donaldson, A.F., Kaiser, A., Kröning, D., Tautschnig, M., Wahl, T.: SATAbs: A bit-precise verifier for C programs (competition contribution). In: Proc. TACAS. pp. 552–555. LNCS 7214, Springer (2012). https://doi.org/10.1007/978-3-642-28756-5_47
19. Beckert, B., Hähnle, R.: Reasoning and verification: State of the art and current trends. IEEE Intelligent Systems **29**(1), 20–29 (2014). https://doi.org/10.1109/MIS.2014.3
20. 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
21. Beyer, D.: Second competition on software verification (Summary of SV-COMP 2013). In: Proc. TACAS. pp. 594–609. LNCS 7795, Springer (2013). https://doi.org/10.1007/978-3-642-36742-7_43
22. Beyer, D.: Status report on software verification (Competition summary SV-COMP 2014). In: Proc. TACAS. pp. 373–388. LNCS 8413, Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_25
23. Beyer, D.: Software verification and verifiable witnesses (Report on SV-COMP 2015). In: Proc. TACAS. pp. 401–416. LNCS 9035, Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_31
24. Beyer, D.: Reliable and reproducible competition results with BenchExec and witnesses (Report on SV-COMP 2016). In: Proc. TACAS. pp. 887–904. LNCS 9636, Springer (2016). https://doi.org/10.1007/978-3-662-49674-9_55
25. Beyer, D.: Software verification with validation of results (Report on SV-COMP 2017). In: Proc. TACAS. pp. 331–349. LNCS 10206, Springer (2017). https://doi.org/10.1007/978-3-662-54580-5_20
26. 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
27. 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

28. Beyer, D.: Advances in automatic software verification: SV-COMP 2020. In: Proc. TACAS (2). pp. 347–367. LNCS 12079, Springer (2020). https://doi.org/10.1007/978-3-030-45237-7_21
29. Beyer, D.: Software verification: 10th comparative evaluation (SV-COMP 2021). In: Proc. TACAS (2). pp. 401–422. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_24
30. Beyer, D.: Progress on software verification: SV-COMP 2022. In: Proc. TACAS (2). pp. 375–402. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_20
31. 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
32. 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
33. Beyer, D., Kanav, S.: CoVeriTeam: On-demand composition of cooperative verification systems. In: Proc. TACAS. pp. 561–579. LNCS 13243, Springer (2022). https://doi.org/10.1007/978-3-030-99524-9_31
34. Beyer, D., Kanav, S., Richter, C.: Construction of Verifier Combinations Based on Off-the-Shelf Verifiers. In: Proc. FASE. pp. 49–70. Springer (2022). https://doi.org/10.1007/978-3-030-99429-7_3
35. 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
36. 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
37. Beyer, D., Lee, N.Z., Wendler, P.: Interpolation and SAT-based model checking revisited: Adoption to software verification. arXiv/CoRR **2208**(05046) (July 2022). https://doi.org/10.48550/arXiv.2208.05046
38. Beyer, D., Spiessl, M.: The static analyzer Frama-C in SV-COMP (competition contribution). In: Proc. TACAS (2). pp. 429–434. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_26
39. 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
40. Beyer, D., Zufferey, D., Majumdar, R.: CSIsat: Interpolation for LA+EUF. In: Proc. CAV. pp. 304–308. LNCS 5123, Springer (2008). https://doi.org/10.1007/978-3-540-70545-1_29
41. Beyer, D.: Cooperative verification: Towards reliable safety-critical systems (invited talk). In: Proc. FTSCS. pp. 1–2. ACM (2022). https://doi.org/10.1145/3563822.3572548
42. Birgmeier, J., Bradley, A.R., Weissenbacher, G.: Counterexample to induction-guided abstraction-refinement (CTIGAR). In: Proc. CAV. pp. 831–848. LNCS 8559, Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_55
43. Bradley, A.R.: SAT-based model checking without unrolling. In: Proc. VMCAI. pp. 70–87. LNCS 6538, Springer (2011). https://doi.org/10.1007/978-3-642-18275-4_7

44. Bradley, A.R., Manna, Z.: Property-directed incremental invariant generation. Formal Asp. Comput. **20**(4–5), 379–405 (2008). https://doi.org/10.1007/s00165-008-0080-9
45. Brain, M., Joshi, S., Kröning, D., Schrammel, P.: Safety verification and refutation by k-invariants and k-induction. In: Proc. SAS. pp. 145–161. LNCS 9291, Springer (2015). https://doi.org/10.1007/978-3-662-48288-9_9
46. Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R.: The MathSAT 4 SMT solver. In: Proc. CAV. pp. 299–303. LNCS 5123, Springer (2008). https://doi.org/10.1007/978-3-540-70545-1_28
47. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers **35**(8), 677–691 (1986). https://doi.org/10.1109/TC.1986.1676819
48. Bryant, R.E.: Binary decision diagrams. In: Handbook of Model Checking, pp. 191–217. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_7
49. Bu, L., Xie, Z., Lyu, L., Li, Y., Guo, X., Zhao, J., Li, X.: Brick: Path enumeration-based bounded reachability checking of C programs (competition contribution). In: Proc. TACAS (2). pp. 408–412. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_22
50. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. In: Proc. LICS. pp. 428–439. IEEE (1990). https://doi.org/10.1109/LICS.1990.113767
51. Calcagno, C., Distefano, D., Dubreil, J., Gabi, D., Hooimeijer, P., Luca, M., O'Hearn, P.W., Papakonstantinou, I., Purbrick, J., Rodriguez, D.: Moving fast with software verification. In: Proc. NFM. pp. 3–11. LNCS 9058, Springer (2015). https://doi.org/10.1007/978-3-319-17524-9_1
52. Calcagno, C., Distefano, D., O'Hearn, P.W., Yang, H.: Compositional shape analysis by means of bi-abduction. ACM **58**(6), 26:1–26:66 (2011). https://doi.org/10.1145/2049697.2049700
53. Cassez, F., Matsuoka, T., Pierzchalski, E., Smyth, N.: Perentie: Modular trace refinement and selective value tracking (competition contribution). In: Proc. TACAS. pp. 439–442. LNCS 9035, Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_39
54. Cassez, F., Sloane, A.M., Roberts, M., Pigram, M., Suvanpong, P., de Aledo Marugán, P.G.: Skink: Static analysis of programs in LLVM intermediate representation (competition contribution). In: Proc. TACAS. pp. 380–384. LNCS 10206, Springer (2017). https://doi.org/10.1007/978-3-662-54580-5_27
55. Chalupa, M., Strejček, J., Vitovská, M.: Joint forces for memory safety checking. In: Proc. SPIN. pp. 115–132. Springer (2018). https://doi.org/10.1007/978-3-319-94111-0_7
56. Chalupa, M., Řechtáčková, A., Mihalkovič, V., Zaoral, L., Strejček, J.: Symbiotic 9: String analysis and backward symbolic execution with loop folding (competition contribution). In: Proc. TACAS (2). pp. 462–467. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_32
57. Chaudhary, E., Joshi, S.: Pinaka: Symbolic execution meets incremental solving (competition contribution). In: Proc. TACAS (3). pp. 234–238. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_20
58. Chen, Y.F., Hsieh, C., Lengál, O., Lii, T.J., Tsai, M.H., Wang, B.Y., Wang, F.: PAC learning-based verification and model synthesis. In: Proc. ICSE. pp. 714–724. ACM (2016). https://doi.org/10.1145/2884781.2884860

59. Chen, Y.F., Hsieh, C., Tsai, M.H., Wang, B.Y., Wang, F.: CPArec: Verifying recursive programs via source-to-source program transformation (competition contribution). In: Proc. TACAS. pp. 426–428. LNCS 9035, Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_35
60. 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
61. Christ, J., Hoenicke, J., Nutz, A.: SMTInterpol: An interpolating SMT solver. In: Proc. SPIN. pp. 248–254. LNCS 7385, Springer (2012). https://doi.org/10.1007/978-3-642-31759-0_19
62. Church, A.: A note on the Entscheidungsproblem. Journal of Symbolic Logic **1**(1), 40–41 (1936). https://doi.org/10.2307/2269326
63. Cimatti, A., Griggio, A.: Software model checking via IC3. In: Proc. CAV. pp. 277–293. LNCS 7358, Springer (2012). https://doi.org/10.1007/978-3-642-31424-7_23
64. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: Infinite-state invariant checking with IC3 and predicate abstraction. Formal Methods in System Design **49**(3), 190–218 (2016). https://doi.org/10.1007/s10703-016-0257-4
65. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Proc. CAV. pp. 154–169. LNCS 1855, Springer (2000). https://doi.org/10.1007/10722167_15
66. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM **50**(5), 752–794 (2003). https://doi.org/10.1145/876638.876643
67. 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
68. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Proc. Logic of Programs 1981. pp. 52–71. LNCS 131, Springer (1982). https://doi.org/10.1007/BFb0025774
69. Cook, B.: Formal reasoning about the security of Amazon web services. In: Proc. CAV (2). pp. 38–47. LNCS 10981, Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_3
70. Cook, B., Podelski, A., Rybalchenko, A.: Terminator: Beyond safety. In: Proc. CAV. pp. 415–418. LNCS 4144, Springer (2006). https://doi.org/10.1007/11817963_37
71. Cordeiro, L.C., Kesseli, P., Kröning, D., Schrammel, P., Trtík, M.: JBmc: A bounded model checking tool for verifying Java bytecode. In: Proc. CAV. pp. 183–190. LNCS 10981, Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_10
72. Cordeiro, L.C., Kröning, D., Schrammel, P.: Jbmc: Bounded model checking for Java bytecode (competition contribution). In: Proc. TACAS (3). pp. 219–223. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_17
73. Coto, A., Inverso, O., Sales, E., Tuosto, E.: A prototype for data race detection in CSeq 3 (competition contribution). In: Proc. TACAS (2). pp. 413–417. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_23
74. Cousot, P., Cousot, R.: Systematic design of program-analysis frameworks. In: Proc. POPL. pp. 269–282. ACM (1979). https://doi.org/10.1145/567752.567778
75. 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

76. Cuoq, P., Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.: Frama-C. In: Proc. SEFM. pp. 233–247. Springer (2012). https://doi.org/10.1007/978-3-642-33826-7_16
77. Dangl, M., Löwe, S., Wendler, P.: CPAchecker with support for recursive programs and floating-point arithmetic (competition contribution). In: Proc. TACAS. pp. 423–425. LNCS 9035, Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_34
78. Darke, P., Agrawal, S., Venkatesh, R.: VeriAbs: A tool for scalable verification by abstraction (competition contribution). In: Proc. TACAS (2). pp. 458–462. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_32
79. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem proving. Commun. ACM **5**(7), 394–397 (1962). https://doi.org/10.1145/368273.368557
80. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. ACM **7**(3), 201–215 (1960). https://doi.org/10.1145/321033.321034
81. Detlefs, D., Nelson, G., Saxe, J.B.: Simplify: A theorem prover for program checking. J. ACM **52**(3), 365–473 (2005). https://doi.org/10.1145/1066100.1066102
82. Dijkstra, E.W.: A constructive approach to the problem of program correctness. BIT Numerical Mathematics **8**, 174–186 (1968). https://doi.org/10.1007/BF01933419
83. Dijkstra, E.W.: Guarded commands, nondeterminacy, and formal derivation of programs. Comm. ACM **18**(8), 453–457 (1975). https://doi.org/10.1145/360933.360975
84. Dockins, R., Foltzer, A., Hendrix, J., Huffman, B., McNamee, D., Tomb, A.: Constructing semantic models of programs with the software analysis workbench. In: Proc. VSTTE. pp. 56–72. LNCS 9971, Springer (2016). https://doi.org/10.1007/978-3-319-48869-1_5
85. D'Silva, V., Kröning, D., Weissenbacher, G.: A survey of automated techniques for formal software verification. IEEE Trans. on CAD of Integrated Circuits and Systems **27**(7), 1165–1178 (2008). https://doi.org/10.1109/TCAD.2008.923410
86. Ernst, G.: A complete approach to loop verification with invariants and summaries. Tech. Rep. arXiv:2010.05812v2, arXiv (January 2020). https://doi.org/10.48550/arXiv.2010.05812
87. Falke, S., Merz, F., Sinz, C.: Llbmc: Improved bounded model checking of C programs using LLVM (competition contribution). In: Proc. TACAS. pp. 623–626. LNCS 7795, Springer (2013). https://doi.org/10.1007/978-3-642-36742-7_48
88. Flanagan, C., Leino, K.R.M., Lillibridge, M., Nelson, G., Saxe, J.B., Stata, R.: Extended static checking for Java. In: Proc. PLDI. pp. 234–245. ACM (2002). https://doi.org/10.1145/512529.512558
89. Floyd, R.W.: Assigning meanings to programs. Mathematical Aspects of Computer Science, Proc. Symposia in Applied Mathematics **19**, 19–32 (1967), Republished: https://doi.org/10.1007/978-94-011-1793-7_4
90. Gadelha, M.Y.R., Monteiro, F.R., Cordeiro, L.C., Nicole, D.A.: Esbmc v6.0: Verifying C programs using k-induction and invariant inference (competition contribution). In: Proc. TACAS (3). pp. 209–213. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_15
91. Gadelha, M.Y.R., Ismail, H.I., Cordeiro, L.C.: Handling loops in bounded model checking of C programs via k-induction. Int. J. Softw. Tools Technol. Transf. **19**(1), 97–114 (2015). https://doi.org/10.1007/s10009-015-0407-9
92. Garavel, H., ter Beek, M.H., van de Pol, J.: The 2020 expert survey on formal methods. In: Proc. FMICS. pp. 3–69. LNCS 12327, Springer (2020). https://doi.org/10.1007/978-3-030-58298-2_1

93. Gavrilenko, N., Ponce de León, H., Furbach, F., Heljanko, K., Meyer, R.: BMC for weak memory models: Relation analysis for compact SMT encodings. In: Proc. CAV. pp. 355–365. LNCS 11561, Springer (2019). https://doi.org/10.1007/978-3-030-25540-4_19
94. Giesl, J., Mesnard, F., Rubio, A., Thiemann, R., Waldmann, J.: Termination competition (termCOMP 2015). In: Proc. CADE. pp. 105–108. LNCS 9195, Springer (2015). https://doi.org/10.1007/978-3-319-21401-6_6
95. Gilmore, P.C.: A proof method for quantification theory: Its justification and realization. IBM J. Res. Dev. **4**(1), 28–35 (1960). https://doi.org/10.1147/rd.41.0028
96. 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
97. Grebenshchikov, S., Gupta, A., Lopes, N.P., Popeea, C., Rybalchenko, A.: Hsf(c): A software verifier based on Horn clauses (competition contribution). In: Proc. TACAS. pp. 549–551. LNCS 7214, Springer (2012). https://doi.org/10.1007/978-3-642-28756-5_46
98. Gurfinkel, A., Albarghouthi, A., Chaki, S., Li, Y., Chechik, M.: Ufo: Verification with interpolants and abstract interpretation (competition contribution). In: Proc. TACAS. pp. 637–640. LNCS 7795, Springer (2013). https://doi.org/10.1007/978-3-642-36742-7_52
99. Gurfinkel, A., Belov, A.: FrankenBit: Bit-precise verification with many bits (competition contribution). In: Proc. TACAS. pp. 408–411. LNCS 8413, Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_32
100. Gurfinkel, A., Kahsai, T., Navas, J.A.: SeaHorn: A framework for verifying C programs (competition contribution). In: Proc. TACAS. pp. 447–450. LNCS 9035, Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_41
101. Gödel, K.: Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. Monatsh. f. Mathematik und Physik **38**(1), 173–198 (1931). https://doi.org/10.1007/BF01700692
102. Günther, H., Laarman, A., Weissenbacher, G.: Vienna Verification Tool: IC3 for parallel software (competition contribution). In: Proc. TACAS. pp. 954–957. LNCS 9636, Springer (2016). https://doi.org/10.1007/978-3-662-49674-9_69
103. Hajdu, Á., Micskei, Z.: Efficient strategies for CEGAR-based model checking. J. Autom. Reasoning **64**(6), 1051–1091 (2019). https://doi.org/10.1007/s10817-019-09535-x
104. Haran, A., Carter, M., Emmi, M., Lal, A., Qadeer, S., Rakamarić, Z.: Smack+Corral: A modular verifier (competition contribution). In: Proc. TACAS. pp. 451–454. LNCS 9035, Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_42
105. He, F., Sun, Z., Fan, H.: Deagle: An SMT-based verifier for multi-threaded programs (competition contribution). In: Proc. TACAS (2). pp. 424–428. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_25
106. Heibi, I., Peroni, S., Shotton, D.: Software review: COCI, the OpenCitations Index of Crossref open DOI-to-DOI citations. Scientometrics **121**(2), 1213–1228 (11 2019). https://doi.org/10.1007/s11192-019-03217-6
107. Heizmann, M., Chen, Y.F., Dietsch, D., Greitschus, M., Hoenicke, J., Li, Y., Nutz, A., Musa, B., Schilling, C., Schindler, T., Podelski, A.: Ultimate Automizer and the search for perfect interpolants (competition contribution). In: Proc. TACAS (2). pp. 447–451. LNCS 10806, Springer (2018). https://doi.org/10.1007/978-3-319-89963-3_30

108. Heizmann, M., Hoenicke, J., Podelski, A.: Refinement of trace abstraction. In: Proc. SAS. pp. 69–85. LNCS 5673, Springer (2009). https://doi.org/10.1007/978-3-642-03237-0_7
109. Heizmann, M., Hoenicke, J., Podelski, A.: Software model checking for people who love automata. In: Proc. CAV. pp. 36–52. LNCS 8044, Springer (2013). https://doi.org/10.1007/978-3-642-39799-8_2
110. Hensel, J., Mensendiek, C., Giesl, J.: AProVE: Non-termination witnesses for C programs (competition contribution). In: Proc. TACAS (2). pp. 403–407. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_21
111. Henzinger, T.A., Jhala, R., Majumdar, R., McMillan, K.L.: Abstractions from proofs. In: Proc. POPL. pp. 232–244. ACM (2004). https://doi.org/10.1145/964001.964021
112. 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
113. Hilbert, D.: Mathematische Probleme. Vortrag, gehalten auf dem internationalen Mathematiker-Kongreß zu Paris 1900. Nachrichten von der Königl. Gesellschaft der Wissenschaften zu Göttingen. Mathematisch-Physikalische Klasse. **1900**(3), 253–297 (1900), https://www.deutschestextarchiv.de/book/show/hilbert_mathematische_1900
114. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM **12**(10), 576–580 (1969). https://doi.org/10.1145/363235.363259
115. Holík, L., Hruška, M., Lengál, O., Rogalewicz, A., Simácek, J., Vojnar, T.: Forester: From heap shapes to automata predicates (competition contribution). In: Proc. TACAS. pp. 365–369. LNCS 10206, Springer (2017). https://doi.org/10.1007/978-3-662-54580-5_24
116. Holík, L., Kotoun, M., Peringer, P., Šoková, V., Trtík, M., Vojnar, T.: Predator shape analysis tool suite. In: Hardware and Software: Verification and Testing. pp. 202–209. LNCS 10028, Springer (2016). https://doi.org/10.1007/978-3-319-49052-6
117. Holzer, A., Kröning, D., Schallhart, C., Tautschnig, M., Veith, H.: Proving reachability using FShell (competition contribution). In: Proc. TACAS. pp. 538–541. LNCS 7214, Springer (2012). https://doi.org/10.1007/978-3-642-28756-5_43
118. 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
119. 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
120. Inverso, O., Trubiani, C.: Parallel and distributed bounded model checking of multi-threaded programs. In: Proc. PPoPP. pp. 202–216. ACM (2020). https://doi.org/10.1145/3332466.3374529
121. Jhala, R., Majumdar, R.: Software model checking. ACM Computing Surveys **41**(4) (2009). https://doi.org/10.1145/1592434.1592438
122. Kahsai, T., Rümmer, P., Sanchez, H., Schäf, M.: JayHorn: A framework for verifying Java programs. In: Proc. CAV. pp. 352–358. LNCS 9779, Springer (2016). https://doi.org/10.1007/978-3-319-41528-4_19
123. Karpenkov, E.G., Monniaux, D., Wendler, P.: Program analysis with local policy iteration. In: Proc. VMCAI. pp. 127–146. LNCS 9583, Springer (2016). https://doi.org/10.1007/978-3-662-49122-5_6

124. Kettl, M., Lemberger, T.: The static analyzer Infer in SV-COMP (competition contribution). In: Proc. TACAS (2). pp. 451–456. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_30
125. 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
126. 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
127. Kröning, D., Tautschnig, M.: Cbmc: C bounded model checker (competition contribution). In: Proc. TACAS. pp. 389–391. LNCS 8413, Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_26
128. Kröning, D., Weissenbacher, G.: Interpolation-based software verification with Wolverine. In: Proc. CAV. pp. 573–578. LNCS 6806, Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_45
129. Lahiri, S.K., Nieuwenhuis, R., Oliveras, A.: SMT techniques for fast predicate abstraction. In: Proc. CAV. pp. 424–437. LNCS 4144, Springer (2006). https://doi.org/10.1007/11817963_39
130. Lange, T., Prinz, F., Neuhäußer, M.R., Noll, T., Katoen, J.: Improving generalization in software IC3. In: Proc. SPIN'18. pp. 85–102. LNCS 10869, Springer (2018). https://doi.org/10.1007/978-3-319-94111-0_5
131. Lange, T., Neuhäußer, M.R., Noll, T.: IC3 software model checking on control flow automata. In: Proc. FMCAD. pp. 97–104 (2015)
132. Lauko, H., Ročkai, P., Barnat, J.: Symbolic computation via program transformation. In: Proc. ICTAC. pp. 313–332. Springer (2018). https://doi.org/10.1007/978-3-030-02508-3_17
133. Lauko, H., Ročkai, P.: Lart: Compiled abstract execution (competition contribution). In: Proc. TACAS (2). pp. 457–461. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_31
134. Lauko, H., Ročkai, P., Barnat, J.: Symbolic computation via program transformation. In: Proc. ICTAC. pp. 313–332. LNCS 11187, Springer (2018). https://doi.org/10.1007/978-3-030-02508-3_17
135. Le, Q.L., Tran, M., Chin, W.N.: HIPrec: Verifying recursive programs with a satisfiability solver (2016), https://loc.bitbucket.io/papers/hiprec.pdf
136. Le, T.C., Ta, Q.T., Chin, W.N.: HipTNT+: A termination and non-termination analyzer by second-order abduction (competition contribution). In: Proc. TACAS. pp. 370–374. LNCS 10206, Springer (2017). https://doi.org/10.1007/978-3-662-54580-5_25
137. Lee, C.Y.: Representation of switching circuits by binary-decision programs. Bell Syst. Tech. J. **38**(4), 985–999 (1959). https://doi.org/10.1002/j.1538-7305.1959.tb01585.x
138. Leeson, W., Dwyer, M.: Graves-CPA: A graph-attention verifier selector (competition contribution). In: Proc. TACAS (2). pp. 440–445. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_28
139. Leino, K.R.M., Nelson, G.: An extended static checker for Modula-3. In: Proc. CC. pp. 302–305. LNCS 1383, Springer (1998). https://doi.org/10.1007/BFb0026441
140. Luckow, K.S., Dimjasevic, M., Giannakopoulou, D., Howar, F., Isberner, M., Kahsai, T., Rakamaric, Z., Raman, V.: JDart: A dynamic symbolic analysis framework. In: Proc. TACAS. pp. 442–459. LNCSS 9636, Springer (2016). https://doi.org/10.1007/978-3-662-49674-9_26

141. Malík, V., Schrammel, P., Vojnar, T.: 2LS: Heap analysis and memory safety (competition contribution). In: Proc. TACAS (2). pp. 368–372. LNCS 12079, Springer (2020). https://doi.org/10.1007/978-3-030-45237-7_22
142. Manna, Z., Pnueli, A.: Temporal verification of reactive systems: Safety. Springer (1995). https://doi.org/10.1007/978-1-4612-4222-2
143. McCarthy, J.: Towards a mathematical science of computation. In: Information Processing, Proc. DFIP Congress. pp. 21–28. North-Holland (1962), Republished: https://doi.org/10.1007/978-94-011-1793-7_2
144. McMillan, K.L.: Symbolic Model Checking. Springer (1993). https://doi.org/10.1007/978-1-4615-3190-6
145. 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
146. McMillan, K.L.: An interpolating theorem prover. Theor. Comput. Sci. **345**(1), 101–121 (2005). https://doi.org/10.1016/j.tcs.2005.07.003
147. McMillan, K.L.: Interpolation and model checking. In: Handbook of Model Checking, pp. 421–446. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_14
148. Metta, R., Medicherla, R.K., Chakraborty, S.: BMC+FUZZ: Efficient and effective test generation. In: Proc. DATE. pp. 1419–1424. IEEE (2022). https://doi.org/10.23919/DATE54114.2022.9774672
149. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proc. DAC. pp. 530–535. ACM (2001). https://doi.org/10.1145/378239.379017
150. Mues, M., Howar, F.: JDART: Portfolio solving, breadth-first search and SMT-Lib strings (competition contribution). In: Proc. TACAS (2). pp. 448–452. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_30
151. Mues, M., Howar, F.: GDart (competition contribution). In: Proc. TACAS (2). pp. 435–439. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_27
152. Müller, P., Vojnar, T.: CPALIEN: Shape analyzer for CPACHECKER (competition contribution). In: Proc. TACAS. pp. 395–397. LNCS 8413, Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_28
153. Nelson, G., Oppen, D.C.: Simplification by cooperating decision procedures. ACM Trans. Program. Lang. Syst. **1**(2), 245–257 (1979). https://doi.org/10.1145/357073.357079
154. Nelson, G., Oppen, D.C.: Fast decision procedures based on congruence closure. J. ACM **27**(2), 356–364 (1980). https://doi.org/10.1145/322186.322198
155. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer (1999). https://doi.org/10.1007/978-3-662-03811-6
156. Noller, Y., Păsăreanu, C.S., Le, X.B.D., Visser, W., Fromherz, A.: Symbolic PATHFINDER for SV-COMP (competition contribution). In: Proc. TACAS (3). pp. 239–243. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_21
157. OpenCitations: COCI CSV dataset of all the citation data, version 16 (2022). https://doi.org/10.6084/m9.figshare.6741422.v16
158. Peano, G.: Arithmetices Principia: Nova Methodo Exposita. Fratres Bocca (1889), https://n2t.net/ark:/13960/t0xp7g625
159. Peringer, P., Šoková, V., Vojnar, T.: PREDATORHP revamped (not only) for interval-sized memory regions and memory reallocation (competition contribution). In: Proc. TACAS (2). pp. 408–412. LNCS 12079, Springer (2020). https://doi.org/10.1007/978-3-030-45237-7_30

160. Piterman, N., Pnueli, A.: Temporal logic and fair discrete systems. In: Handbook of Model Checking, pp. 27–73. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_2
161. Podelski, A., Rybalchenko, A.: A complete method for the synthesis of linear ranking functions. In: Proc. VMCAI. pp. 239–251. LNCS 2937, Springer (2004). https://doi.org/10.1007/978-3-540-24622-0_20
162. Podelski, A., Rybalchenko, A.: Transition predicate abstraction and fair termination. In: Proc. POPL. pp. 132–144. ACM (2005). https://doi.org/10.1145/1040305.1040317
163. Ponce-De-Leon, H., Haas, T., Meyer, R.: Dartagnan: Leveraging compiler optimizations and the price of precision (competition contribution). In: Proc. TACAS (2). pp. 428–432. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_26
164. Popeea, C., Rybalchenko, A.: Threader: A verifier for multi-threaded programs (competition contribution). In: Proc. TACAS. pp. 633–636. LNCS 7795, Springer (2013). https://doi.org/10.1007/978-3-642-36742-7_51
165. Pratikakis, P., Foster, J.S., Hicks, M.: Locksmith: Practical static race detection for C. ACM Trans. Program. Lang. Syst. **33**(1) (January 2011). https://doi.org/10.1145/1889997.1890000
166. Prawitz, D.: An improved proof procedure. Theoria **26**(2), 102–139 (1960). https://doi.org/10.1111/j.1755-2567.1960.tb00558.x
167. Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. Comptes Rendus du I Congrès de Mathématiciens des Pays Slaves, Warszawa pp. 92–101 (1929)
168. Păsăreanu, C.S., Visser, W., Bushnell, D.H., Geldenhuys, J., Mehlitz, P.C., Rungta, N.: Symbolic PathFinder: integrating symbolic execution with model checking for Java bytecode analysis. Autom. Software Eng. **20**(3), 391–425 (2013). https://doi.org/10.1007/s10515-013-0122-2
169. Queille, J.P., Sifakis, J.: Specification and verification of concurrent systems in CESAR. In: Proc. Symposium on Programming. pp. 337–351. LNCS 137, Springer (1982). https://doi.org/10.1007/3-540-11494-7_22
170. Quine, W.V.: A proof procedure for quantification theory. J. Symbolic Logic **20**(2), 141–149 (1955). https://doi.org/10.2307/2266900
171. Quiring, B., Manolios, P.: Gacal: Conjecture-based verification (competition contribution). In: Proc. TACAS (2). pp. 388–392. LNCS 12079, Springer (2020). https://doi.org/10.1007/978-3-030-45237-7_26
172. Rajkhowa, P., Lin, F.: Viap 1.1 (competition contribution). In: Proc. TACAS (3). pp. 250–255. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_23
173. Rakamarić, Z., Emmi, M.: SMACK: Decoupling source language details from verifier implementations. In: Proc. CAV. pp. 106–113. LNCS 8559, Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_7
174. Richter, C., Hüllermeier, E., Jakobs, M.C., Wehrheim, H.: Algorithm selection for software validation based on graph kernels. Autom. Softw. Eng. **27**(1), 153–186 (2020). https://doi.org/10.1007/s10515-020-00270-x
175. Richter, C., Wehrheim, H.: PeSCo: Predicting sequential combinations of verifiers (competition contribution). In: Proc. TACAS (3). pp. 229–233. LNCS 11429, Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_19

176. Rocha, H.O., Barreto, R.S., Cordeiro, L.C.: Memory management test-case generation of C programs using bounded model checking. In: Proc. SEFM. pp. 251–267. LNCS 9276, Springer (2015). https://doi.org/10.1007/978-3-319-22969-0_18
177. Rocha, H.O., Ismail, H., Cordeiro, L.C., Barreto, R.S.: Model checking embedded C software using k-induction and invariants. In: Proc. SBESC. pp. 90–95. IEEE (2015). https://doi.org/10.1109/SBESC.2015.24
178. Rocha, H.O., Menezes, R., Cordeiro, L., Barreto, R.: Map2Check: Using symbolic execution and fuzzing (competition contribution). In: Proc. TACAS (2). pp. 403–407. LNCS 12079, Springer (2020). https://doi.org/10.1007/978-3-030-45237-7_29
179. Rocha, W., Rocha, H.O., Ismail, H., Cordeiro, L.C., Fischer, B.: DepthK: A k-induction verifier based on invariant inference for C programs (competition contribution). In: Proc. TACAS. pp. 360–364. LNCS 10206, Springer (2017). https://doi.org/10.1007/978-3-662-54580-5_23
180. Rojas, R., Göktekin, C., Friedland, G., Krüger, M., Scharf, L., Kunif̈, D., Langmack, O.: Konrad Zuses Plankalkül – Seine Genese und eine moderne Implementierung, pp. 215–235. Springer (2004). https://doi.org/10.1007/978-3-642-18631-8_9
181. Saan, S., Schwarz, M., Apinis, K., Erhard, J., Seidl, H., Vogler, R., Vojdani, V.: Goblint: Thread-modular abstract interpretation using side-effecting constraints (competition contribution). In: Proc. TACAS (2). pp. 438–442. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_28
182. Saarikivi, O., Heljanko, K.: LCTD: Tests-guided proofs for C programs on LLVM (competition contribution). In: Proc. TACAS. pp. 927–929. LNCS 9636, Springer (2016). https://doi.org/10.1007/978-3-662-49674-9_62
183. Scott, R., Dockins, R., Ravitch, T., Tomb, A.: Crux: Symbolic execution meets SMT-based verification (competition contribution). Zenodo (February 2022). https://doi.org/10.5281/zenodo.6147218
184. Shamakhi, A., Hojjat, H., Rümmer, P.: Towards string support in JayHorn (competition contribution). In: Proc. TACAS (2). pp. 443–447. LNCS 12652, Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_29
185. Shannon, C.E.: A symbolic analysis of relay and switching circuits. Transactions of the American Institute of Electrical Engineers **57**, 713–723 (1938). https://doi.org/10.1109/T-AIEE.1938.5057767
186. Sharma, V., Hussein, S., Whalen, M.W., McCamant, S.A., Visser, W.: Java Ranger at SV-COMP 2020 (competition contribution). In: Proc. TACAS (2). pp. 393–397. LNCS 12079, Springer (2020). https://doi.org/10.1007/978-3-030-45237-7_27
187. Sharma, V., Hussein, S., Whalen, M.W., McCamant, S.A., Visser, W.: Java Ranger: Statically summarizing regions for efficient symbolic execution of Java. In: Proc. ESEC/FSE. pp. 123–134. ACM (2020). https://doi.org/10.1145/3368089.3409734
188. Shved, P., Mandrykin, M.U., Mutilin, V.S.: Predicate analysis with Blast 2.7 (competition contribution). In: Proc. TACAS. pp. 525–527. LNCS 7214, Springer (2012). https://doi.org/10.1007/978-3-642-28756-5_39
189. Ströder, T., Giesl, J., Brockschmidt, M., Frohn, F., Fuhs, C., Hensel, J., Schneider-Kamp, P., Aschermann, C.: Automatically Proving Termination and Memory Safety for Programs with Pointer Arithmetic. J. Autom. Reasoning **58**(1), 33–65 (2016). https://doi.org/10.1007/s10817-016-9389-x

190. Tóth, T., Hajdu, A., Vörös, A., Micskei, Z., Majzik, I.: Theta: A framework for abstraction refinement-based model checking. In: Proc. FMCAD. pp. 176–179 (2017). https://doi.org/10.23919/FMCAD.2017.8102257
191. Turing, A.: On computable numbers, with an application to the Entscheidungsproblem. In: Proc. LMS. vol. s2-42, pp. 230–265. London Mathematical Society (1937). https://doi.org/10.1112/plms/s2-42.1.230
192. 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), https://turingarchive.kings.cam.ac.uk/publications-lectures-and-talks-amtb/amt-b-8
193. Urban, C.: FuncTion: An abstract domain functor for termination (competition contribution). In: Proc. TACAS. pp. 464–466. LNCS 9035, Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_46
194. Visser, W., Geldenhuys, J.: Coastal: Combining concolic and fuzzing for Java (competition contribution). In: Proc. TACAS (2). pp. 373–377. LNCS 12079, Springer (2020). https://doi.org/10.1007/978-3-030-45237-7_23
195. Visser, W., Havelund, K., Brat, G.P., Park, S., Lerda, F.: Model checking programs. Autom. Softw. Eng. **10**(2), 203–232 (2003). https://doi.org/10.1023/A:1022920129859
196. Vojdani, V., Apinis, K., Rõtov, V., Seidl, H., Vene, V., Vogler, R.: Static race detection for device drivers: The Goblint approach. In: Proc. ASE. pp. 391–402. ACM (2016). https://doi.org/10.1145/2970276.2970337
197. Volkov, A.R., Mandrykin, M.U.: Predicate abstractions memory modeling method with separation into disjoint regions. Proceedings of the Institute for System Programming (ISPRAS) **29**, 203–216 (2017). https://doi.org/10.15514/ISPRAS-2017-29(4)-13
198. Wang, W., Barrett, C.: Cascade (competition contribution). In: Proc. TACAS. pp. 420–422. LNCS 9035, Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_33
199. Weissenbacher, G., Kröning, D., Malik, S.: Wolverine: Battling bugs with interpolants (competition contribution). In: Proc. TACAS. pp. 556–558. LNCS 7214, Springer (2012). https://doi.org/10.1007/978-3-642-28756-5_48
200. Wirth, N.: Program development by stepwise refinement. Commun. ACM **14**(4), 221–227 (1971). https://doi.org/10.1145/362575.362577
201. Yin, L., Dong, W., Liu, W., Li, Y., Wang, J.: Yogar-Cbmc: Cbmc with scheduling constraint based abstraction refinement (competition contribution). In: Proc. TACAS. pp. 422–426. LNCS 10806, Springer (2018). https://doi.org/10.1007/978-3-319-89963-3_25
202. Yin, L., Dong, W., Liu, W., Wang, J.: On scheduling constraint abstraction for multi-threaded program verification. IEEE Trans. Softw. Eng. (2018). https://doi.org/10.1109/TSE.2018.2864122
203. Zheng, M., Edenhofner, J.G., Luo, Z., Gerrard, M.J., Dwyer, M.B., Siegel, S.F.: Civl: Applying a general concurrency verification framework to C/Pthreads programs (competition contribution). In: Proc. TACAS. pp. 908–911. LNCS 9636, Springer (2016). https://doi.org/10.1007/978-3-662-49674-9_57
204. Ádám, Z., Bajczi, L., Dobos-Kovács, M., Hajdu, A., Molnár, V.: Theta: Portfolio of cegar-based analyses with dynamic algorithm selection (competition contribution). In: Proc. TACAS (2). pp. 474–478. LNCS 13244, Springer (2022). https://doi.org/10.1007/978-3-030-99527-0_34