Proc. TACAS 2017, © Springer

Software Verification with Validation of Results (Report on SV-COMP 2017)

Dirk Beyer

LMU Munich, Germany

Abstract. This report describes the 2017 Competition on Software Verification (SV-COMP), the 6th edition of the annual thorough comparative evaluation of fully-automatic software verifiers. The goal is to reflect the current state of the art in software verification in terms of effectiveness and efficiency. The major achievement of the 6th edition of SV-COMP is that the verification results were validated in most categories. The verifiers have to produce verification witnesses, which contain hints that a validator can later use to reproduce the verification result. The answer of a verifier counts only if the validator confirms the verification result. SV-COMP uses two independent, publicly available witness validators. For 2017, a new category structure was introduced that now orders the verification tasks according to the property to verify on the top level, and by the type of programs (e.g., which kind of data types are used) on a second level. The categories *Overflows* and *Termination* were heavily extended, and the category *SoftwareSystems* now contains also verification tasks from the software system BusyBox. The competition used 8 908 verification tasks that each consisted of a C program and a property (reachability, memory safety, termination). SV-COMP 2017 had 32 participating verification systems from 12 countries.

1 Introduction

Software verification is an increasingly important research area, and the annual Competition on Software Verification (SV-COMP) [1] is the showcase of the state of the art in the area, in particular, of the effectiveness and efficiency that is currently achieved by tool implementations of the most recent ideas, concepts, and algorithms for fully-automatic verification. Every year, the SV-COMP project consists of two parts: (1) The collection of verification tasks and their partition into categories has to take place before the actual experiments start, and requires quality-assurance work on the source code in order to ensure a high-quality evaluation. It is important that the SV-COMP verification tasks reflect what the research and development community considers interesting and challenging for evaluating the effectivity (soundness and completeness) and efficiency (performance) of state-of-the-art verification tools. (2) The actual experiments of the comparative evaluation of the relevant tool implementations is performed

[1] `https://sv-comp.sosy-lab.org`

by the organizer of SV-COMP. Since SV-COMP shall stimulate and showcase new technology, it is necessary to explore and define standards for a reliable and reproducible execution of such a competition: we use BenchExec [10], a modern framework for reliable benchmarking and resource measurement, to run the experiments, and verification witnesses [7, 8] to validate the verification results.

As for every edition, this SV-COMP report describes the (updated) rules and definitions, presents the competition results, and discusses other interesting facts about the execution of the competition experiments. Also, we need to measure the success of SV-COMP by evaluating whether the main objectives of the competition are achieved (list taken from [5]):

1. provide an overview of the state of the art in software-verification technology and increase visibility of the most recent software verifiers,
2. establish a repository of software-verification tasks that is publicly available for free use as standard benchmark suite for evaluating verification software,
3. establish standards that make it possible to compare different verification tools, including a property language and formats for the results, and
4. accelerate the transfer of new verification technology to industrial practice.

As for (1), there were 32 participating software systems from 12 countries, representing a broad spectrum of technology (cf. Table 4). SV-COMP is considered an important event in the research community, and increasingly also in industry. This year, SV-COMP for the first time had two participating verification systems from industry. As for (2), the total set of verification tasks increased in size from 6 661 to 8 908. Still, SV-COMP has an ongoing focus on collecting and constructing verification tasks to ensure even more diversity. Compared to the last years, the level and amount of quality-assurance activities from the SV-COMP community increased significantly, as witnessed by the issue tracker [2] and by the pull requests [3] in the GitHub project. As for (3), the largest step forward was to apply an extension of the standard witness language as a common, exchangeable format to correctness witnesses as well this year (violation witnesses have been used before). This means, if a verifier reports False (claims to know an error path through the program that violates the specification), then it produces a violation witness; if a verifier reports True (claims to know a proof of correctness), then it produces a correctness witness. The two points of the SV-COMP scoring schema for correct answers True are assigned only if the correctness witness was confirmed by a witness validator, i.e., a proof of correctness could be reconstructed by a different tool. As for (4), we continuously received positive feedback from industry.

Related Competitions. It is well-understood that competitions are an important evaluation method, and there are other competitions in the field of software verification: RERS [4] [20] and VerifyThis [5] [22]. While SV-COMP performs replicable experiments in a *controlled* environment (dedicated resources, resource limits), the RERS Challenges give more room for exploring combinations of

[2] `https://github.com/sosy-lab/sv-benchmarks/issues?q=is:issue`
[3] `https://github.com/sosy-lab/sv-benchmarks/pulls?q=is:pr` [4] `http://rers-challenge.org`
[5] `http://etaps2016.verifythis.org`

interactive with automatic approaches without limits on the resources, and the VerifyThis Competition focuses on evaluating approaches and ideas rather than on *fully-automatic* verification. The termination competition termCOMP [6] [16] concentrates on termination but considers a broader range of systems, including logic and functional programs. A more comprehensive list of other competitions is provided in the report on SV-COMP 2014 [4].

2 Procedure

The overall competition organization did not change in comparison to the past editions [2, 3, 4, 5, 6]. SV-COMP is an open competition, where all verification tasks are known before the submission of the participating verifiers, which is necessary due to the complexity of the language C. During the *benchmark submission* phase, new verification tasks were collected and classified, during the *training* phase, the teams inspected the verification tasks and trained their verifiers (also, the verification tasks received fixes and quality improvement), and during the *evaluation* phase, verification runs were preformed with all competition candidates, and the system descriptions were reviewed by the competition jury. The participants received the results of their verifier directly via e-mail, and after a few days of inspection, the results were publicly announced on the competition web site. The *Competition Jury* consisted again of the chair and one member of each participating team. Team representatives of the jury are listed in Table 3.

3 Definitions, Formats, and Rules

Verification Task. The definition of verification task was not changed (taken from [4]). A verification task consists of a C program and a property. A verification run is a non-interactive execution of a competition candidate (verifier) on a single verification task, in order to check whether the following statement is correct: "The program satisfies the property." The result of a verification run is a triple (ANSWER, WITNESS, TIME). ANSWER is one of the following outcomes:

TRUE: The property is satisfied (no path exists that violates the property), and a correctness witness is produced that contains hints to reconstruct the proof.

FALSE: The property is violated (there exists a path that violates the property), and a violation witness is produced that contains hints to replay the error path to the property violation.

UNKNOWN: The tool cannot decide the problem, or terminates abnormally, or exhausts the computing resources time or memory (the competition candidate does not succeed in computing an answer TRUE or FALSE).

The component WITNESS [7, 8] was this year for the first time mandatory for *both* answers TRUE or FALSE; a few categories were excluded from validation if the validators did not sufficiently support a certain kind of program or property. We used the two publicly available witness validators CPAchecker and UAutomizer.

[6] `http://termination-portal.org/wiki/Termination_Competition`

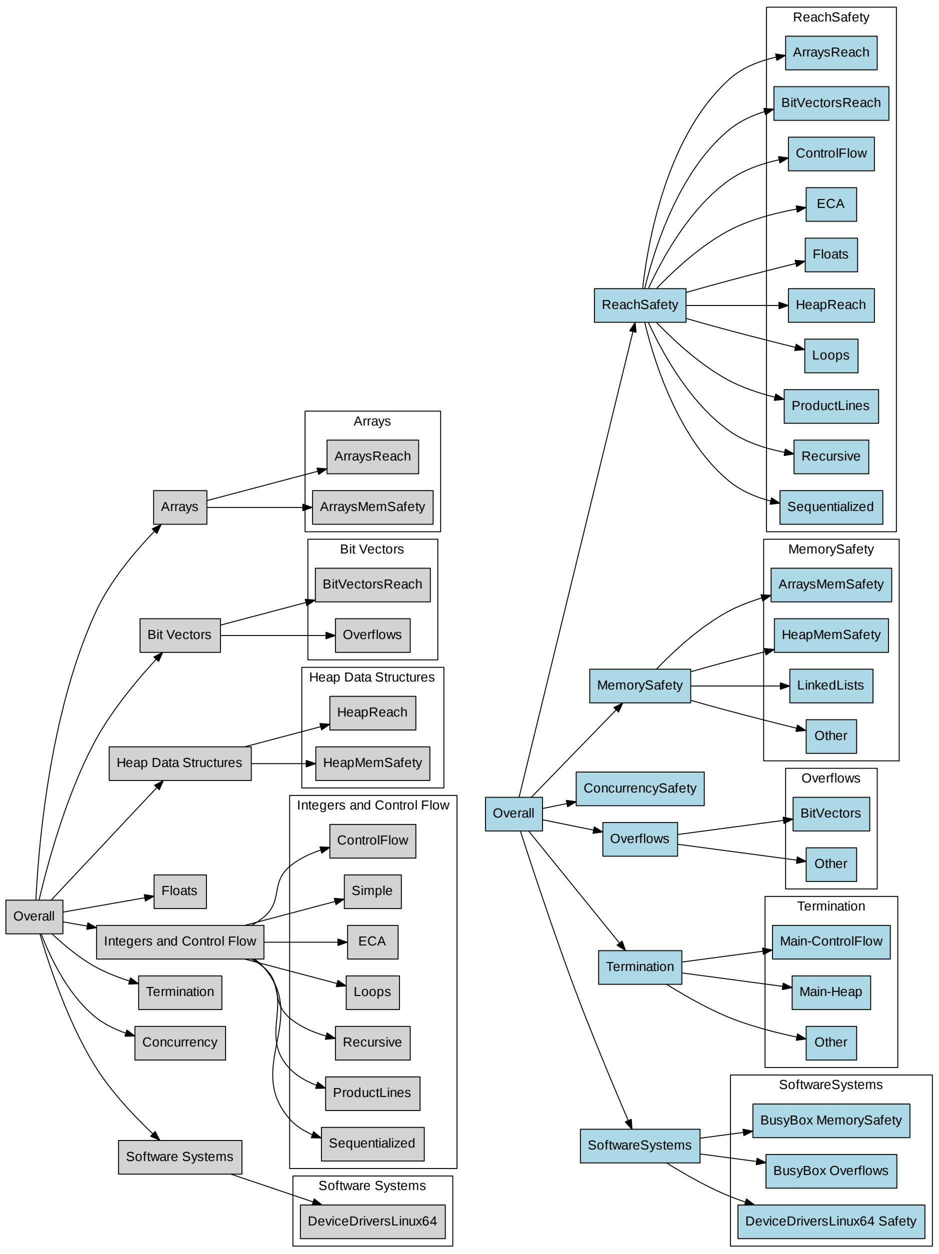


Fig. 1: Categories; left: SV-COMP 2016; right: SV-COMP 2017; category *Falsification* contains all verification tasks of *Overall* without *Termination*

Table 1: Properties used in SV-COMP 2017 (cf. [5] for more details)

Formula	Interpretation
`G ! call(foo())`	A call to function `foo` is not reachable on any finite execution.
`G valid-free`	All memory deallocations are valid (counterexample: invalid free). More precisely: There exists no finite execution of the program on which an invalid memory deallocation occurs.
`G valid-deref`	All pointer dereferences are valid (counterexample: invalid dereference). More precisely: There exists no finite execution of the program on which an invalid pointer dereference occurs.
`G valid-memtrack`	All allocated memory is tracked, i.e., pointed to or deallocated (counterexample: memory leak). More precisely: There exists no finite execution of the program on which the program lost track of some previously allocated memory.
`F end`	All program executions are finite and end on proposition `end`, which marks all program exits (counterexample: infinite loop). More precisely: There exists no execution of the program on which the program never terminates.

TIME is measured as consumed CPU time until the verifier terminates, including the consumed CPU time of all processes that the verifier started [10]. If TIME is equal to or larger than the time limit (15 min), then the verifier is terminated and the ANSWER is set to 'timeout' (and interpreted as UNKNOWN).

Categories. The collection of verification tasks is partitioned into categories. A major update was done on the structure of the categories, in order to support various extensions that were planned for SV-COMP 2017. For example, the categories *Overflows* and *Termination* were considerably extended (*Overflows* from 12 to 328 and *Termination* from 631 to 1 437 verification tasks). Figure 1 shows the previous structure of main and sub-categories on the left, and the new structure is shown on the right. The guideline is to have main categories that correspond to different properties and sub-categories that reflect the type of program. The goal of the category *SoftwareSystems* is to complement the other categories (which sometimes contain small and constructed examples to show certain verification features) by large and complicated verification tasks from real software systems (further structured according to system and property to verify). The category assignment was proposed and implemented by the competition chair, and approved by the competition jury. SV-COMP 2017 has a total of eight categories for which award plaques are handed out, including the six main categories, category *Overall*, which contains the union of all categories, and category *Falsification*. Category *Falsification* consists of all verification tasks with safety properties, and any answers TRUE are not counted for the score (the goal of this category is to show bug-hunting capabilities of verifiers that are not able to construct correctness proofs). The categories are described in more detail on the competition web site.[7]

[7] `https://sv-comp.sosy-lab.org/2017/benchmarks.php`

Table 2: Scoring schema for SV-COMP 2017

Reported result	Points	Description
Unknown	0	Failure to compute verification result
False correct	+1	Violation of property in program was correctly found
False incorrect	−16	Violation reported but property holds (false alarm)
True correct	+2	Correct program reported to satisfy property
True correct unconfirmed	+1	Correct program reported to satisfy property, but the witness was not confirmed by a validator
True incorrect	−32	Incorrect program reported as correct (wrong proof)

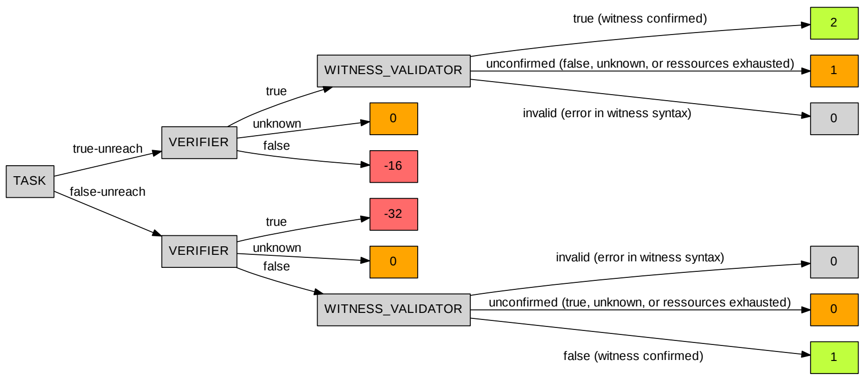

TASK
true-unreach
false-unreach
VERIFIER
true
unknown
false
0
-16
WITNESS_VALIDATOR
true (witness confirmed)
2
unconfirmed (false, unknown, or ressources exhausted)
1
invalid (error in witness syntax)
0
VERIFIER
true
-32
unknown
0
false
WITNESS_VALIDATOR
invalid (error in witness syntax)
0
unconfirmed (true, unknown, or ressources exhausted)
0
false (witness confirmed)
1

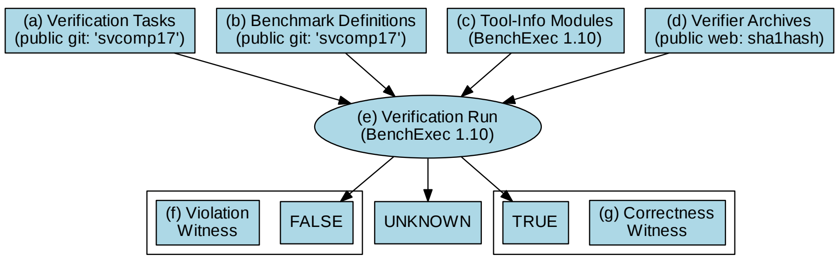
Fig. 2: Visualization of the scoring schema for the reachability property

Properties and Their Format. For the definition of the properties and the property format, we refer to the previous competition report [5]. All specifications are available in the main directory of the benchmark repository. Table 1 lists the properties and their syntax as overview.

Evaluation by Scores and Run Time. The scoring schema of SV-COMP 2017 is similar to the previous scoring schema, except that results with answer True are now assigned two points only if the witness was confirmed by a validator, and one point is assigned if the answer matches the expected result but the witness was not confirmed. Table 2 provides the overview and Fig. 2 visually illustrates the score assignment for one property. The ranking is decided based on the sum of points (normalized for meta categories) and for equal sum of points according to success run time, which is the total CPU time over all verification tasks for which the verifier reported a correct verification result. *Opt-out from Categories* and *Score Normalization for Meta Categories* was done as described previously [3] (page 597).

4 Reproducibility

It is important that the SV-COMP experiments can be independently replicated, and that the results can be reproduced. Therefore, all major components that are used for the competition need to be publicly available. Figure 3 gives an overview over the components that contribute to the reproducible setup of SV-COMP.



(a) Verification Tasks (public git: 'svcomp17')
(b) Benchmark Definitions (public git: 'svcomp17')
(c) Tool-Info Modules (BenchExec 1.10)
(d) Verifier Archives (public web: sha1hash)
(e) Verification Run (BenchExec 1.10)
(f) Violation Witness
FALSE
UNKNOWN
TRUE
(g) Correctness Witness

Fig. 3: Setup: SV-COMP components that support reproducibility

Repositories for Verification Tasks (a), Benchmark Definitions (b), and Tool-Information Modules (c). The previous competition report [6] describes how replicability is ensured by making all essential ingredients available in public archives. The verification tasks (a) are available via the tag 'svcomp17' in a public Git repository.[8] The benchmark definitions (b) define for each verifier (i) on which verification tasks the verifier is to be executed (each verifier can choose which categories to participate in) and (ii) which parameters need to be passed to the verifier (there are global parameters that are specified for all categories, and there are specific parameters such as the bit architecture). The benchmark definitions are available via the tag 'svcomp17' in another public Git repository.[9] The tool-information modules (c) ensure, for each verifier respectively, that the command line to execute the verifier is correctly assembled (including source and property file as well as the options) from the parts specified in the benchmark definition (b), and that the results of the verifier are correctly interpreted and translated into the uniform SV-COMP result (True, False(p), Unknown). The tool-info modules that were used for SV-COMP 2017 are available in BenchExec 1.10.[10]

Reliable Assignment and Controlling of Computing Resources (e). We use BenchExec[11] [10] to satisfy the requirements for scientifically valid experimentation, such as (i) accurate measurement and reliable enforcement of limits for CPU time and memory, and (ii) reliable termination of processes (including all child processes). For the first time in SV-COMP, we used BenchExec's container mode, in order to make sure that read and write operations are properly controlled. For example, it was previously not automatically and reliably enforced that tools do not increase the assigned memory by using a RAM disk. This and some other issues that previously required manual inspection and analysis are now systematically solved.

Violation Witnesses (f) and Correctness Witnesses (g). In SV-COMP, each verification run (if applicable) is followed by a validation run that checks whether the witness adheres to the exchange format and can be confirmed. The

[8] `https://github.com/sosy-lab/sv-benchmarks/tree/svcomp17/c`
[9] `https://github.com/sosy-lab/sv-comp/tree/svcomp17/benchmark-defs`
[10] `https://github.com/sosy-lab/benchexec/tree/1.10/benchexec/tools`
[11] `https://github.com/sosy-lab/benchexec`

resource limits for the witness validators were 2 processing units (one physical CPU core with hyper-threading), 7 GB memory, and 10 % of the verification time (i.e., 1.5 min) for violation witnesses and 100 % (15 min) for correctness witnesses. The purpose of the tighter resource limits is to avoid delegating all verification work to the validator. This witness-based validation process ensures a higher quality of assignment of scores, compared to without witnesses: if a verifier claims a found bug but is not able to provide a witness, then the verifier does not get the full score. The witness format and the validation process is explained on the witness-format web page [12]. The version of the exchange format that was used for SV-COMP 2017 has the tag 'svcomp17'. More details on witness validation is given in two related research articles [7, 8].

Verifier Archives (d). Due to legal issues we do not re-distribute the verifiers on the competition web site, but list for each verifier a URL to an archive that the participants promised to keep publicly available, together with the SHA1 hash of the archive that was used in SV-COMP. An overview table is provided on the systems-description page of the competition web site [13]. For replicating experiments, the archive can be downloaded and verified against the given SHA1 hash. Each archive contains all parts that are needed to execute the verifier (statically-linked executables and all components that are required in a certain version, or for which no standard Ubuntu package is available). The archives are also supposed to contain a license that permits use in SV-COMP, replicating the SV-COMP experiments, that all data that the verifier produces as output are property of the person that executes the verifier, and that the results obtained from the verifier can be published without any restriction.

5 Results and Discussion

For the sixth time, the competition experiments represent the state of the art in fully-automatic software-verification tools. The report shows the improvements of the last year, in terms of effectiveness (number of verification tasks that can be solved, correctness of the results, as accumulated in the score) and efficiency (resource consumption in terms of CPU time). The results that are presented in this article were inspected and approved by the participating teams.

Participating Verifiers. Table 3 provides an overview of the participating competition candidates and Table 4 lists the features and technologies that are used in the verification tools.

Computing Resources. The resource limits were the same as last year [6]: Each verification run was limited to 8 processing units (cores), 15 GB of memory, and 15 min of CPU time. The witness validation was limited to 2 processing units, 7 GB of memory, and 1.5 min of CPU time for violation witnesses and 15 min of CPU time for correctness witnesses. The machines for running the

[12] `https://github.com/sosy-lab/sv-witnesses/tree/svcomp17`
[13] `https://sv-comp.sosy-lab.org/2017/systems.php`

Table 3: Competition candidates with tool references and representing jury members

Participant	**Ref.**	**Jury member**	**Affiliation**
2LS	[34]	Peter Schrammel	U. of Sussex, UK
APROVE	[19]	Jera Hensel	RWTH Aachen, Germany
BLAST	[35]	Vadim Mutilin	ISP RAS, Russia
CBMC	[26]	Michael Tautschnig	Queen Mary, UK
CEAGLE		Guang Chen	Tsinghua U., China
CIVL	[37]	Stephen Siegel	U. of Delaware, USA
CONSEQUENCE		Anand Yeolekar	TCS, India
CPA-BAM-BNB	[1]	Pavel Andrianov	ISP RAS, Russia
CPA-KIND	[9]	Matthias Dangl	U. of Passau, Germany
CPA-SEQ	[14]	Karlheinz Friedberger	U. of Passau, Germany
DEPTHK	[33]	Herbert O. Rocha	Federal U. of Roraima, Brazil
ESBMC	[28]	Lucas Cordeiro	U. of Oxford, UK
ESBMC-FALSI	[28]	Bernd Fischer	Stellenbosch U., ZA
ESBMC-INCR	[28]	Denis Nicole	U. of Southampton, UK
ESBMC-KIND	[15]	Mikhail Ramalho	U. of Southampton, UK
FORESTER	[21]	Martin Hruska	Brno U. of Technology, Czechia
HIPTNT+	[27]	Ton Chanh Le	National U. of Singapore, Singapore
LAZY-CSEQ	[23]	Omar Inverso	Gran Sasso Science Institute, Italy
LAZY-CSEQ-ABS	[30]	Bernd Fischer	Stellenbosch U., ZA
LAZY-CSEQ-SWARM		Truc Nguyen Lam	U. of Southampton, UK
MU-CSEQ	[36]	Salvatore La Torre	U. of Salerno, Italy
PREDATORHP	[25]	Tomas Vojnar	Brno U. of Technology, Czechia
SKINK	[11]	Franck Cassez	Macquarie U. at Sydney, Australia
SMACK	[32]	Zvonimir Rakamarić	U. of Utah, USA
SYMBIOTIC	[12]	Jan Strejček	Masaryk U., Czechia
SYMDIVINE	[24]	Jiří Barnat	Masaryk U., Czechia
UAUTOMIZER	[18]	Matthias Heizmann	U. of Freiburg, Germany
UKOJAK	[31]	Daniel Dietsch	U. of Freiburg, Germany
UL-CSEQ	[29]	Gennaro Parlato	U. of Southampton, UK
UTAIPAN	[17]	Marius Greitschus	U. of Freiburg, Germany
VERIABS	[13]	Priyanka Darke	TCS, India
YOGAR-CBMC		Liangze Yin	National U. of Defense Techn., China

experiments were different from last year, because we now had 168 machines available and each verification run could be executed on a completely unloaded, dedicated machine, in order to achieve precise measurements. Each machine had one Intel Xeon E3-1230 v5 CPU, with 8 processing units each, a frequency of 3.4 GHz, 33 GB of RAM, and a GNU/Linux operating system (x86_64-linux, Ubuntu 16.04 with Linux kernel 4.4).

One complete verification execution of the competition consisted of 421 benchmarks (each verifier on each selected category according to the opt-outs), summing up to 170 417 verification runs. Witness validation required 678 benchmarks (combinations of verifier, category with witness validation, and two validators) summing up to 232 916 validation runs. The consumed total CPU time for one complete competition run for verification required a total of 490 days of CPU time. Each tool was executed several times, in order to make sure no installation issues

Table 4: Technologies and features that the competition candidates offer

Participant	CEGAR	Predicate Abstraction	Symbolic Execution	Bounded Model Checking	k-Induction	Property-Directed Reach.	Explicit-Value Analysis	Numeric. Interval Analysis	Shape Analysis	Separation Logic	Bit-Precise Analysis	ARG-Based Analysis	Lazy Abstraction	Interpolation	Automata-Based Analysis	Concurrency Support	Ranking Functions
2LS				✓	✓			✓			✓						✓
APROVE			✓				✓	✓			✓						✓
BLAST	✓	✓					✓					✓	✓	✓			
CBMC				✓							✓					✓	
CEAGLE	✓	✓		✓							✓	✓	✓				
CIVL			✓	✓				✓								✓	
CONSEQUENCE				✓							✓					✓	
CPA-BAM-BNB	✓	✓					✓				✓	✓	✓	✓			
CPA-KIND	✓	✓		✓	✓			✓			✓	✓	✓				
CPA-SEQ	✓	✓		✓	✓		✓	✓	✓		✓	✓	✓	✓		✓	✓
DEPTHK				✓	✓						✓					✓	
ESBMC				✓							✓					✓	
ESBMC-FALSI				✓							✓					✓	
ESBMC-INCR				✓							✓					✓	
ESBMC-KIND				✓	✓						✓					✓	
FORESTER	✓								✓						✓		
HIPTNT+									✓	✓							✓
LAZY-CSEQ				✓							✓					✓	
LAZY-CSEQ-ABS				✓				✓			✓					✓	
LAZY-CSEQ-SWARM				✓							✓					✓	
MU-CSEQ				✓							✓					✓	
PREDATORHP									✓								
SKINK	✓						✓							✓	✓		
SMACK	✓			✓		✓					✓		✓			✓	
SYMBIOTIC			✓								✓						
SYMDIVINE			✓				✓				✓				✓	✓	
UAUTOMIZER	✓	✓									✓		✓	✓	✓		✓
UKOJAK	✓	✓									✓		✓	✓			
UL-CSEQ	✓	✓										✓	✓	✓		✓	
UTAIPAN	✓	✓									✓		✓	✓	✓		
VERIABS				✓	✓			✓									
YOGAR-CBMC	✓			✓							✓		✓			✓	

Table 5: Quantitative overview over all results; empty cells mark opt-outs

Participant	**ReachSafety** 4696 points 2897 tasks	**MemSafety** 541 points 328 tasks	**ConcurrencySafety** 1293 points 1047 tasks	**Overflows** 533 points 328 tasks	**Termination** 2513 points 1437 tasks	**SoftwareSystems** 5520 points 2871 tasks	**FalsificationOverall** 2908 points 7471 tasks	**Overall** 14553 points 8908 tasks
2LS	1038	-918	0	310	624	720	-4330	-1204
AProVE					**1492**			
Blast						866		
CBMC	2154	219	1135	230		37	**2554**	4766
Ceagle	2170	138		12		352	343	1972
CIVL			1251					
ConSequence			794					
CPA-BAM-BnB						975	-735	
CPA-kInd	2156	0	0	101	0	778	232	1963
CPA-Seq	**2862**	88	1020	101	**974**	1011	1302	**5296**
DepthK	1552	27	548	85	-307	254	976	1894
ESBMC	1125	-85	601	105	0	301	184	1674
ESBMC-falsi	583	-65	552	106	0	-17	1269	1261
ESBMC-incr	1810	80	756	187	0	0	**1482**	3209
ESBMC-kind	1940	191	654	304	0	334	**1610**	4335
Forester								
HipTNT+					835			
Lazy-CSeq			1226					
Lazy-CSeq-Abs			**1293**					
Lazy-CSeq-Swarm			**1293**					
MU-CSeq			1179					
PredatorHP		**319**						
Skink			-102					
SMACK	**3432**	150	1208	**417**	0	**1695**	1154	**6917**
Symbiotic	2063	**304**	0	281	0	-7079	-2698	42
SymDIVINE			389					
UAutomizer	**2372**	**308**	0	**372**	**2184**	**1055**	982	**7099**
UKojak	1564	268	0	356	0	410	900	3837
UL-CSeq			1177					
UTaipan	1894	296	0	**365**	0	**1067**	918	4511
VeriAbs								
Yogar-CBMC			**1293**					

Table 6: Overview of the top-three verifiers for each category (CPU time in h, rounded to two significant digits)

Rank	Participant	Score	CPU Time	Solved Tasks	False Alarms	Wrong Proofs
ReachSafety						
1	**SMACK**	**3432**	100	1 543		
2	CPA-Seq	2862	39	1 874	5	
3	UAutomizer	2372	27	1 344		
MemSafety						
1	**PredatorHP**	**319**	.82	219		
2	UAutomizer	308	1.9	145		
3	Symbiotic	304	.080	233		
ConcurrencySafety						
1	**Yogar-CBMC**	**1293**	.35	1 047		
2	Lazy-CSeq-Abs	1293	2.1	1 047		
3	Lazy-CSeq-Swarm	1293	3.2	1 047		
Overflows						
1	**SMACK**	**417**	18	271		**1**
2	UAutomizer	372	.83	273		
3	UTaipan	365	.85	270		
Termination						
1	**UAutomizer**	**2184**	8.3	1 272		
2	AProVE	1492	3.6	520		
3	CPA-Seq	974	14	821	4	
SoftwareSystems						
1	**SMACK**	**1695**	20	1 391	2	
2	UTaipan	1067	18	1 567	7	**4**
3	UAutomizer	1055	19	1 568	7	**4**
FalsificationOverall						
1	**CBMC**	**2554**	8.1	1 817		
2	ESBMC-kind	1610	27	1 341	20	
3	ESBMC-incr	1482	32	1 400	25	
Overall						
1	**UAutomizer**	**7099**	57	4 602	7	**4**
2	SMACK	6917	180	4 463	12	**2**
3	CPA-Seq	5296	81	5 393	29	

Table 7: Necessary effort to compute results False versus True (measurement values rounded to two significant digits)

Result	True		False	
	CPU Time (avg. in s)	CPU Energy (avg. in J)	CPU Time (avg. in s)	CPU Energy (avg. in J)
UAutomizer	46	450	42	420
SMACK	210	2 200	51	580
CPA-Seq	65	650	39	320

occur during the execution. We used BenchExec [10] to measure and control computing resources (CPU time, memory, CPU energy) and VerifierCloud [14] to distribute, install, run, and clean-up verification runs, and to collect the results.

Quantitative Results. Table 5 presents the quantitative overview over all tools and all categories (Forester participated only in subcategory *ReachSafety-Heap*, *MemSafety-Heap*, and *MemSafety-LinkedLists*; VeriAbs participated only in some subcategories of *ReachSafety*). The head row mentions the category, the maximal score for the category, and the number of verification tasks. The tools are listed in alphabetical order; every table row lists the scores of one verifier for each category. We indicate the top-three candidates by formatting their scores in bold face and in larger font size. An empty table cell means that the verifier opted-out from the respective category. There was one category for which the winner was decided based on the run time: in category *ConcurrencySafety*, all top-three verifiers achieved the maximum score of 1293 points, but the run time differed. More information (including interactive tables, quantile plots for every category, and also the raw data in XML format) is available on the competition web-site. [15]

Table 6 reports the top-three verifiers for each category. The run time (column 'CPU Time') refers to successfully solved verification tasks (column 'Solved Tasks'). The columns 'False Alarms' and 'Wrong Proofs' report the number of verification tasks for which the verifier reported wrong results: reporting an error path but the property holds (incorrect False) and claiming that the program fulfills the property although it actually contains a bug (incorrect True), respectively.

Discussion of Scoring Schema and Normalization. The verification community considers it more difficult to compute correctness proofs compared to computing error paths: according to Table 2, an answer True yields 2 points (confirmed witness) and 1 point (unconfirmed witness), while an answer False yields 1 point (confirmed witness). This can have consequences on the final ranking, as discussed in the report on the last SV-COMP edition [6].

Assigning a higher score value to results True (compared to results False) seems justified by the CPU time and energy that the verifiers need to compute the result. Table 7 shows actual numbers on this: the first column lists the three best verifiers of category *Overall*, the second and third columns report the average CPU time and average CPU energy for results True, and the forth and fifth columns for results False. The average is taken over all verification tasks; the CPU time is reported in seconds and the CPU energy in Joule (BenchExec reads and accumulates the energy measurements of Intel CPUs). Especially for the verifier SMACK, the effort to compute results True is significantly higher compared to the effort to compute results False: 210 s versus 51 s of average CPU time per verification task and 2 200 J versus 580 J of average CPU energy.

A similar consideration was made on the score normalization. The community considers the value of each category equal, which has the consequence that solving a verification task in a large category (many, often similar verification tasks) has less value than solving a verification task in a small category (only a few verification tasks) [3]. The values for category *Overall* in Table 6 illustrate

[14] `https://vcloud.sosy-lab.org/` [15] `https://sv-comp.sosy-lab.org/2017/results/`

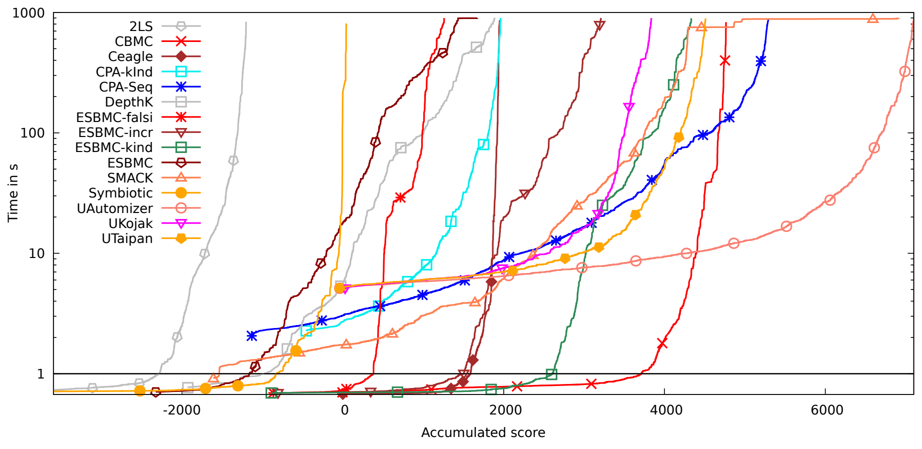


Fig. 4: Quantile functions for category *Overall*. Each quantile function illustrates the quantile (x-coordinate) of the scores obtained by correct verification runs below a certain run time (y-coordinate). More details were given previously [3]. A logarithmic scale is used for the time range from 1 s to 1000 s, and a linear scale is used for the time range between 0 s and 1 s.

the purpose of the score normalization: CPA-Seq solved 5 393 tasks, which is 791 solved tasks more than the winner UAutomizer could solve (4 602). So why did CPA-Seq not win the category? Because UAutomizer is better in the intuitive sense of 'overall': it solved tasks more diversely, the 'overall' value of the verification work is higher. Thus, UAutomizer received 7 099 points and CPA-Seq received 5 296 points. Similarly, in category *SoftwareSystems*, UAutomizer solved 177 more tasks than SMACK; the tasks that UAutomizer solved were considered of less value (i.e., from large categories). SMACK was able to solve considerably more verification tasks in the seemingly difficult BusyBox categories. In these cases, the score normalization correctly maps the community's intuition.

Score-Based Quantile Functions for Quality Assessment. We use score-based quantile functions [3] because these visualizations make it easier to understand the results of the comparative evaluation. The web-site [15] includes such a plot for each category; as example, we show the plot for category *Overall* (all verification tasks) in Fig. 4. A total of 15 verifiers participated in category *Overall*, for which the quantile plot shows the overall performance over all categories (scores for meta categories are normalized [3]). A more detailed discussion of score-based quantile plots, including examples of what interesting insights one can obtain from the plots, is provided in previous competition reports [3, 6].

Correctness of Results. Out of those verifiers that participated in all categories, UKojak is the only verifier that did not report any wrong result, CBMC did not report any false alarm, and Ceagle, CPA-kInd, CPA-Seq, and ESBMC-falsi did not report any wrong proof.

Table 8: Confirmation rate of witnesses

Result	True			False		
	Total	Confirmed	Unconfirmed	Total	Confirmed	Unconfirmed
UAutomizer	3 558	3 481	77	1 173	1 121	52
SMACK	2 947	2 695	252	1 929	1 768	161
CPA-Seq	3 357	3 078	279	2 342	2 315	27

Verifiable Witnesses. For SV-COMP, it is not sufficient to answer with just True or False: each answer must be accompanied by a verification witness. For correctness witnesses, an unconfirmed answer True was still accepted, but was assigned only 1 point instead of 2 (cf. Table 2). All verifiers in categories that required witness validation support the common exchange format for violation and correctness witnesses. We used the two independently developed witness validators that are integrated in CPAchecker and UAutomizer [7, 8].

It is interesting to see that the majority of witnesses that the top-three verifiers produced can be confirmed by the witness-validation process (more than 90 %). Table 8 shows the confirmed versus unconfirmed result: the first column lists the three best verifiers of category *Overall*, the three columns for result True reports the total, confirmed, and unconfirmed number of verification tasks for which the verifier answered with True, respectively, and the three columns for result False reports the total, confirmed, and unconfirmed number of verification tasks for which the verifier answered with False, respectively. More information (for all verifiers) is given in the detailed tables on the competition web-site [15], cf. also the report on the demo category for correctness witnesses from SV-COMP 2016 [6].

6 Conclusion

SV-COMP 2017, the 6th edition of the Competition on Software Verification, attracted 32 participating teams from 12 countries (number of teams 2012: 10, 2013: 11, 2014: 15, 2015: 22, 2016: 35). SV-COMP continues to be the broadest overview of the state of the art in automatic software verification. For the first time in verification history, proof hints (stored in an exchangeable witness) from verifiers were used on a large scale to help a different tool (validator) to validate whether it can, given the proof hints, reproduce a correctness proof. Given the results (cf. Table 8), this approach is successful. The two points for the results True were counted only if the correctness witness was confirmed; for unconfirmed results True, only 1 point was assigned. The number of verification tasks was increased from 6 661 to 8 908. The partitioning of the verification tasks into categories was considerably restructured; the categories *Overflows*, *MemSafety*, and *Termination* were extended and structured using sub-categories; many verification tasks from the software system BusyBox were added to the category *SoftwareSystems*. As before, the large jury and the organizer made sure that the competition follows the high quality standards of the TACAS conference, in particular with respect to the important principles of fairness, community support, and transparency.

References

1. P. Andrianov, V. Mutilin, K. Friedberger, M. Mandrykin, and A. Volkov. CPA-BAM-BnB: Block-abstraction memoization and region-based memory models for predicate abstractions (competition contribution). In *Proc. TACAS*. Springer, 2017.
2. D. Beyer. Competition on software verification (SV-COMP). In *Proc. TACAS*, LNCS 7214, pages 504–524. Springer, 2012.
3. D. Beyer. Second competition on software verification. In *Proc. TACAS*, LNCS 7795, pages 594–609. Springer, 2013.
4. D. Beyer. Status report on software verification. In *Proc. TACAS*, LNCS 8413, pages 373–388. Springer, 2014.
5. D. Beyer. Software verification and verifiable witnesses. In *Proc. TACAS*, LNCS 9035, pages 401–416. Springer, 2015.
6. D. Beyer. Reliable and reproducible competition results with BenchExec and witnesses. In *Proc. TACAS*, LNCS 9636, pages 887–904. Springer, 2016.
7. D. Beyer, M. Dangl, D. Dietsch, and M. Heizmann. Correctness witnesses: Exchanging verification results between verifiers. In *Proc. FSE*, pages 326–337. ACM, 2016.
8. D. Beyer, M. Dangl, D. Dietsch, M. Heizmann, and A. Stahlbauer. Witness validation and stepwise testification across software verifiers. In *Proc. FSE*, pages 721–733. ACM, 2015.
9. D. Beyer, M. Dangl, and P. Wendler. Boosting k-induction with continuously-refined invariants. In *Proc. CAV*, LNCS 9206, pages 622–640. Springer, 2015.
10. D. Beyer, S. Löwe, and P. Wendler. Benchmarking and resource measurement. In *Proc. SPIN*, LNCS 9232, pages 160–178. Springer, 2015.
11. F. Cassez, T. Sloane, M. Roberts, M. Pigram, P. G. D. Aledo, and P. Suvanpong. Skink 2.0: Static analysis of LLVM intermediate representation (competition contribution). In *Proc. TACAS*. Springer, 2017.
12. M. Chalupa, M. Vitovská, M. Jonáš, J. Slaby, and J. Strejček. Symbiotic 4: Beyond reachability (competition contribution). In *Proc. TACAS*. Springer, 2017.
13. B. Chimdyalwar, P. Darke, A. Chauhan, P. Shah, S. Kumar, and R. Venkatesh. VeriAbs: Verification by abstraction (competition contribution). In *Proc. TACAS*. Springer, 2017.
14. M. Dangl, S. Löwe, and P. Wendler. CPAchecker with support for recursive programs and floating-point arithmetic. In *Proc. TACAS*. Springer, 2015.
15. M. Y. R. Gadelha, H. I. Ismail, and L. C. Cordeiro. Handling loops in bounded model checking of C programs via k-induction. *STTT*, 19(1):97–114, 2017.
16. J. Giesl, F. Mesnard, A. Rubio, R. Thiemann, and J. Waldmann. Termination competition (termCOMP 2015). In *Proc. CADE*, LNCS 9195, pages 105–108. Springer, 2015.
17. M. Greitschus, D. Dietsch, M. Heizmann, A. Nutz, C. Schätzle, C. Schilling, F. Schüssele, and A. Podelski. Ultimate Taipan: Trace abstraction and abstract interpretation (competition contribution). In *Proc. TACAS*. Springer, 2017.
18. M. Heizmann, Y.-W. Chen, D. Dietsch, M. Greitschus, B. Musa, A. Nutz, C. Schätzle, C. Schilling, F. Schüssele, and A. Podelski. Ultimate Automizer with an on-demand construction of Floyd-Hoare automata (competition contribution). In *Proc. TACAS*. Springer, 2017.

19. J. Hensel, F. Emrich, F. Frohn, T. Stroeder, and J. Giesl. AProVE: Proving and disproving termination of memory-manipulating C programs (competition contribution). In *Proc. TACAS*. Springer, 2017.
20. F. Howar, M. Isberner, M. Merten, B. Steffen, and D. Beyer. The RERS grey-box challenge 2012: Analysis of event-condition-action systems. In *Proc. ISoLA*, LNCS 7609, pages 608–614. Springer, 2012.
21. M. Hruska, L. Holik, T. Vojnar, O. Lengal, A. Rogalewicz, and J. Simacek. Forester: From heap shapes to automata predicates (competition contribution). In *Proc. TACAS*. Springer, 2017.
22. M. Huisman, V. Klebanov, and R. Monahan. VerifyThis 2012 - A program verification competition. *STTT*, 17(6):647–657, 2015.
23. O. Inverso, T. L. Nguyen, B. Fischer, S. La Torre, and G. Parlato. Lazy-CSeq: A context-bounded model checking tool for multi-threaded C programs. In *Proc. ASE*, pages 807–812. IEEE, 2015.
24. M. Jonáš, J. Mrázek, V. Štill, J. Barnat, and H. Lauko. Optimizing and caching SMT queries in SymDIVINE (competition contribution). In *Proc. TACAS*. 2017.
25. M. Kotoun, P. Peringer, V. Šoková, and T. Vojnar. Optimized PredatorHP and the SV-COMP heap and memory-safety benchmark (competition contribution). In *Proc. TACAS*, LNCS 9636, pages 942–945. Springer, 2016.
26. D. Kröning and M. Tautschnig. CBMC: C bounded model checker (competition contribution). In *Proc. TACAS*, LNCS 8413, pages 389–391. Springer, 2014.
27. T. C. Le, Q.-T. Ta, and W.-N. Chin. HipTNT+: A termination and non-termination analyzer by second-order abduction (competition contribution). In *Proc. TACAS*. Springer, 2017.
28. J. Morse, M. Ramalho, L. Cordeiro, D. Nicole, and B. Fischer. ESBMC 1.22 (competition contribution). In *Proc. TACAS*, pages 405–407. Springer, 2014.
29. T. L. Nguyen, B. Fischer, S. La Torre, and G. Parlato. Lazy sequentialization for the safety verification of unbounded concurrent programs. In *Proc. ATVA*, LNCS 9938, pages 174–191. Springer, 2016.
30. T. L. Nguyen, O. Inverso, B. Fischer, S. La Torre, and G. Parlato. Lazy-CSeq 2.0: Combining lazy sequentialization with abstract interpretation (competition contribution). In *Proc. TACAS*. Springer, 2017.
31. A. Nutz, D. Dietsch, M. M. Mohamed, and A. Podelski. Ultimate Kojak with memory-safety checks (competition contribution). In *Proc. TACAS*. Springer, 2015.
32. Z. Rakamarić and M. Emmi. SMACK: Decoupling source language details from verifier implementations. In *Proc. CAV*, LNCS 8559, pages 106–113. Springer, 2014.
33. W. Rocha, H. O. Rocha, H. Ismail, L. Cordeiro, and B. Fischer. DepthK: A k-induction verifier based on invariant inference for C programs (competition contribution). In *Proc. TACAS*. Springer, 2017.
34. P. Schrammel and D. Kröning. 2LS for program analysis (competition contribution). In *Proc. TACAS*, LNCS 9636, pages 905–907. Springer, 2016.
35. P. Shved, M. Mandrykin, and V. Mutilin. Predicate analysis with Blast 2.7 (competition contribution). In *Proc. TACAS*, pages 525–527. Springer, 2012.
36. E. Tomasco, T. L. Nguyen, O. Inverso, B. Fischer, S. La Torre, and G. Parlato. MU-CSeq 0.4: Individual memory location unwindings (competition contribution). In *Proc. TACAS*, LNCS 9636, pages 938–941. Springer, 2016.
37. M. Zheng, J. G. Edenhofner, Z. Luo, M. J. Gerrard, M. B. Dwyer, and S. F. Siegel. CIVL: Applying a general concurrency verification framework to C/Pthreads programs (competition contribution). In *Proc. TACAS*. Springer, 2016.