SL-COMP: Competition of Solvers for Separation Logic

Mihaela Sighireanu[2(✉)], Juan A. Navarro Pérez[11,12P], Andrey Rybalchenko[3], Nikos Gorogiannis[4], Radu Iosif[14], Andrew Reynolds[13], Cristina Serban[14], Jens Katelaan[10], Christoph Matheja[6], Thomas Noll[6], Florian Zuleger[10], Wei-Ngan Chin[5], Quang Loc Le[9], Quang-Trung Ta[5], Ton-Chanh Le[8], Thanh-Toan Nguyen[5], Siau-Cheng Khoo[5], Michal Cyprian[1], Adam Rogalewicz[1], Tomas Vojnar[1], Constantin Enea[2], Ondrej Lengal[1], Chong Gao[7], and Zhilin Wu[7]

[1] FIT, Brno University of Technology, Brno, Czechia
[2] IRIF, University Paris Diderot and CNRS, Paris, France
mihaela.sighireanu@irif.fr
[3] Microsoft Research, Cambridge, UK
[4] Middlesex University London, London, UK
[5] National University of Singapore, Singapore, Singapore
[6] RWTH Aachen University, Aachen, Germany
[7] State Key Laboratory of Computer Science, Chinese Academy of Sciences, Beijing, China
[8] Stevens Institute of Technology, Hoboken, USA
[9] Teesside University, Middlesbrough, UK
[10] TU Wien, Vienna, Austria
[11] University College London, London, UK
[12] Google, London, UK
[13] University of Iowa, Iowa City, USA
[14] VERIMAG, University Grenoble Alpes and CNRS, Saint-Martin-d'Hères, France

Abstract. SL-COMP aims at bringing together researchers interested on improving the state of the art of the automated deduction methods for Separation Logic (SL). The event took place twice until now and collected more than 1K problems for different fragments of SL. The input format of problems is based on the SMT-LIB format and therefore fully typed; only one new command is added to SMT-LIB's list, the command for the declaration of the heap's type. The SMT-LIB theory of SL comes with ten logics, some of them being combinations of SL with linear arithmetics. The competition's divisions are defined by the logic fragment, the kind of decision problem (satisfiability or entailment) and the presence of quantifiers. Until now, SL-COMP has been run on the StarExec platform, where the benchmark set and the binaries of participant solvers are freely available. The benchmark set is also available with the competition's documentation on a public repository in GitHub.

D. Beyer et al. (Eds.): TACAS 2019, Part III, LNCS 11429, pp. 116–132, 2019.
https://doi.org/10.1007/978-3-030-17502-3_8

1 Introduction

Separation Logic (SL) is an established and fairly popular Hoare logic for imperative, heap-manipulating programs, introduced nearly fifteen years ago by Reynolds [20,24,25]. Its high expressivity, its ability to generate compact proofs, and its support for local reasoning, and its support for local reasoning have motivated the development of tools for automatic reasoning about programs using SL. A rather exhaustive list of the past and present tools using SL may be found at [19].

These tools seek to establish memory safety properties and/or infer shape properties of the heap at a scale of millions of lines of code. They intensively use (semi-)decision procedures for checking satisfiability and entailment problems in SL. Therefore, the development of effective solvers for such problems became a challenge which led to both theoretical results on decidability and complexity of these problems for different fragments of SL and to publicly available tools. To understand the capabilities of these solvers and to motivate their improvement by comparison on a common benchmark, we initiated in 2014 the SL-COMP competition, inspired by the success of SMT-COMP for solvers on first order theories.

This paper presents the history of this competition and its organization for the round at TOOLympics 2019. Section 2 describes the main stages of the competition. Each stage is detailed in a separate section as follows: benchmark's definition in Sect. 3, the participants in Sect. 4 and the running infrastructure in Sect. 5. We conclude the paper in Sect. 6 by a discussion on the impact of the competition and its perspectives.

2 Competition's Stages

2.1 A Short History

The first edition of SL-COMP took place at FLoC 2014 Olympic Games, as an unofficial event associated with the SMT-COMP 2014 competition [31]. The organization details and the achievements of this edition are presented in details in [26]. This was an opportunity to collect from participants about 600 problems on different fragments of SL, to involve six solvers, to lay the foundations of a common input format and to set up a discussion list involving teams developing solvers or verification tools based on SL. Being attached to SMT-COMP allowed to benefit from the experience of SMT-COMP's organizer, David Cok, in setting competition's rules and the execution platform StarExec, as well as in running the competition and publishing the results.

The results of the first edition led to interesting discussions on the mailing list, mainly on the input format chosen, the layout of divisions and the frequency of running the competition. These discussions have converged in defining a working group on the input format and fixed a sparse rhythm of the competition, mainly aligned with FLoC venues.

Therefore, the second edition took place at FLoC 2018 and was associated with the first workshop on Automated Deduction for Separation Logics (ADSL). The organization of the competition followed the stages described in the next section and was disconnected from SMT-COMP. The organizer, Mihaela Sighireanu, exploited the experience acquired with the first edition in running the competition on StarExec. The competition involved ten solvers which ran on 1K problems split over ten newly defined divisions. More precisely, the benchmark set included the set of problems of the 2014 edition and new problems provided by the participants. The problems were specified in the new input format which is aligned with the latest version of SMT-LIB, as detailed in [15] and summarized in Sect. 3.2. The competition's results have been presented during a session of ADSL, which gave the opportunity of a live discussion on the different aspects of organization. The results are available on the competition web site [27].

The TOOLympics edition is a rerun of the second edition with two major changes: a new solver has been included and some benchmark instances have been fixed. The remainder of this paper will present the organization of this edition and the participants involved.

2.2 Organization Process

The competition has a short organization period, three months on average. This is possible due to the fact that material used in the competition (the benchmark set, the definition of the input format, the parsers for input and the pre-processing tools) are publicly available on StarExec and on a shared development repository [22] maintained by the participants and by the organizer.

The competition is launched by a call for benchmarks and participants which also fixes the competition timeline. The call is sent on the competition mailing list `sl-comp@googlegroups.com`.

New solvers are invited to send a short presentation (up to two pages) including the team, the sub-fragment of SL dealt, the main bibliography and the website. In addition, each solver has a corresponding person in the team, which is responsible of preparing the solver for running the competition. This preparation ensures that the input format is supported and that the solver is registered in the execution platform in the divisions of the competition it asked to compete. The organizer creates a subspace on the execution platform for each participant and assigns the permission to the solver's correspondent for this space. She may help the incomer to prepare the solver by providing insights on the use of the execution platform, the input format and the pre-processors from the competition's input format to the solver's format.

The benchmark problems are collected from the community and participants. Until now, we did not limit the number of benchmark instances proposed by participants in each category in order to improve our benchmark set. However, this may change in the future, as discussed on Sect. 3. The benchmark set may change during the competition due to reaction of competitors, but it is fixed starting with the pre-final run.

The competition is run in three steps. The first step is a training period where the solver's correspondent runs the solver on the execution platform and the existing benchmark set. During this step, the benchmark set may be changed as well as the solver's binary. The second step is a pre-final run, launched by the organizer using the binaries of solvers published on the execution platform. The results of this pre-final run are available for all solvers' representatives, which may allow to compare results and have a first view on competitors' achievements. The organizer contacts each correspondent to be sure that the results of this run are accepted. The last step is the final run, which determines the final result. The binaries of solvers submitted to the final run may be different from the ones used in the pre-final run.

The final run of the competition takes place one week before the event at which the competition's results are presented. However, the results are available as soon as possible on the competition's web site.

3 Benchmark Set

The current competition's benchmark set contains more than 1K problems, (precisely 1286 problems), which cover several fragments of Separation Logic. 25% of these problems are satisfiability checking problems. This section outlines the main features of this benchmark set, including the fragments covered, the input format, and the divisions established for this edition. A detailed description of the input theory and format is [15].

3.1 Separation Logic Theory

The input theory is a multi-sorted second order logic over a signature $\Sigma = (\Sigma^{\mathrm{s}}, \Sigma^{\mathrm{f}})$, where the set of sorts Σ^{s} includes two (non necessarily disjoint) subsets of sorts representing locations of the heap, $\Sigma^{\mathrm{s}}_{\mathsf{Loc}}$, respectively heap's data, $\Sigma^{\mathrm{s}}_{\mathsf{Data}}$. For each sort Loc in $\Sigma^{\mathrm{s}}_{\mathsf{Loc}}$, the set of operations includes a constant symbol $\mathsf{nil}^{\mathsf{Loc}}$ modeling the null location. The heap's type τ is an injection from location sorts in $\Sigma^{\mathrm{s}}_{\mathsf{Loc}}$ to data sorts in $\Sigma^{\mathrm{s}}_{\mathsf{Data}}$. We also assume that the signature Σ includes the Boolean signature and an equality function for each sort.

Let *Vars* be a countable set of first-order variables, each $x^{\sigma} \in \mathit{Vars}$ having an associated sort σ. The *Ground Separation Logic* SL^g is the set of formulae generated by the following syntax:

$$\varphi := \phi \mid \mathsf{emp} \mid \mathsf{t} \mapsto \mathsf{u} \mid \varphi_1 * \varphi_2 \mid \varphi_1 \mathrel{-\!\!*} \varphi_2 \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \exists x^{\sigma} \,.\, \varphi_1(x) \quad (1)$$

where ϕ is a Σ-formula, and t, u are Σ-terms of sorts in $\Sigma^{\mathrm{s}}_{\mathsf{Loc}}$ and $\Sigma^{\mathrm{s}}_{\mathsf{Data}}$ respectively, such that they are related by the heap's type τ. As usual, we write $\forall x^{\sigma} \,.\, \varphi(x)$ for $\neg\exists x^{\sigma} \,.\, \neg\varphi(x)$. We omit specifying the sorts of variables and functions when they are clear from the context.

The special atomic formulas of SL^g are the so-called *spatial atoms*: emp specifies an empty heap, $\mathsf{t} \mapsto \mathsf{u}$ specifies a heap consisting of one allocated cell whose

address is $\mathtt{t}$ and whose value is $\mathtt{u}$. The operator "$*$" is the separating conjunction denoting that the sub-heaps specified by its operands have disjoint locations. The operator "$-\!\!*$" is the separating implication operator, also called magic wand. A formula containing only spatial atoms combined using separating conjunction and implication is called *spatial*. Formulas without spatial atoms and separating operators are called *pure*.

The full separation logic SL contains formulas with spatial predicate atoms of the form $P^{\sigma_1 \cdots \sigma_n}(\mathtt{t}_1, \ldots, \mathtt{t}_n)$, where each $\mathtt{t}_i$ is a first-order term of sort σ_i, for $i = 1, \ldots, n$. The predicates $P^{\sigma_1 \cdots \sigma_n}$ belong to a finite set $\mathbb{P}$ of second-order variables and have associated a tuple of parameter sorts $\sigma_1, \ldots, \sigma_n \in \Sigma^{\mathrm{s}}$. Second-order variables $P^{\sigma_1 \cdots \sigma_n} \in \mathbb{P}$ are defined using a set of rules of the form:

$$P(x_1, \ldots, x_n) \leftarrow \phi_P(x_1, \ldots, x_n), \tag{2}$$

where ϕ_P is a formula possibly containing predicate atoms and having free variables in $x_1, \ldots, x_n$. The semantics of predicate atoms is defined by the least fixed point of the function defined by these rules.

An example of a formula specifying a heap with at least two singly linked list cells at locations x and y is:

$$x \mapsto \mathtt{node}(1, y) \ * \ y \mapsto \mathtt{node}(1, z) \ * \ \mathtt{ls}(z, \mathsf{nil}) \ \wedge \ z \neq x \tag{3}$$

where $\Sigma^{\mathrm{s}} = \{\mathtt{Int}, \mathsf{Loc}, \mathsf{Data}\}$ and the function $\mathtt{node}$ has parameters of sort $\mathtt{Int}$ and Loc and its type is Data. The predicate $\mathtt{ls}$ is defined by the following rules:

$$\mathtt{ls}(h, f) \leftarrow h = f \wedge \mathsf{emp} \tag{4}$$

$$\mathtt{ls}(h, f) \leftarrow \exists x, i \ . \ h \neq f \ \wedge \ x \mapsto \mathtt{node}(i, x) \ * \ \mathtt{ls}(x, f) \tag{5}$$

and specifies a possible empty heap storing a singly linked list of Data starting at the location denoted by h and whose last cell contains the location denoted by f. More complex examples of formulas and predicate definitions are provided in [15,26].

3.2 Input Format

The input format of the competition has been changed between the first and the second edition, but it was always based on the SMT-LIB format [2]. The syntax and semantics of this format were discussed and agreed in the public mailing group.

Signature encoding: Following this format, the new functions of SL theory are declared in a "theory" file `SepLogicTyped.smt2` as follows:

```
(theory SepLogicTyped

 :funs ((emp Bool)
        (sep Bool Bool Bool :left-assoc)
```

```
        (wand Bool Bool Bool :right-assoc)
        (par (L D) (pto L D Bool))
        (par (L) (nil L))
        )
)
```

Observe that `pto` and `nil` are polymorphic functions, with sort parameters `L` (for location sort) and `D` (for data sort). There is no restriction on the choice of location and data sorts. However, each problem shall fix them using a special command, not included in SMT-LIB, `declare-heap`. For example, to encode the example given in Eq. 3, we declare an uninterpreted sort `Loc` and a sort `Data` as a datatype as follows:

```
(declare-sort Loc 0)

(declare-datatype Data ((node (d Int) (next  U))))

(declare-heap (Loc Data))
```

The last declaration fixes the type of the heap model.

The predicate definitions are written into SMT-LIB format using the recursive function definition introduced in version 2.6. For instance, the definition of the list segment from Eqs. 4 and 5 is written into SMT-LIB as follows (based on the above declarations of `Loc` and `Data`):

```
(define-fun-rec ls ((h Loc) (f Loc)) Bool
    (or (and emp (= h f))
        (exists ((x Loc) (d Int))
                (and (distinct h f) (sep (pto h (node d x)) (ls x f))))
    )
)
```

Problem format: Each benchmark file is organized as follows:

- Preamble information required by the SMT-LIB format: the sub-logic of SL theory (see Sect. 3.3), the team which proposed the problem, the kind (crafted, application, etc.) and the status (sat or unsat) of the problem.
- A list of declarations for the sorts for locations and data, for the type of the heap (the `declare-heap` command), for the second order predicates, and for the free variables used in the problem's formulae. Notice that the input format is strongly typed. At the end of the declarations, a checking command `check-unsat` may appear to trigger for some solvers the checking for models of predicate declarations.
- One or two assertions (command `assert`) introducing the formulas used in the satisfiability respectively entailment problem.
- The file ends with a checking satisfiability command `check-unsat`. Notice that checking the validity of the entailment $A \Rightarrow B$ is encoded by satisfiability checking of its negation $A \wedge \neg B$.

3.3 Divisions

The main difficulty that faces automatic reasoning using SL is that the logic, due to its expressiveness, does not have very nice decidability properties [1]. For this reason, most program verification tools use incomplete heuristics to solve the satisfiability and entailment problems in SL or restrict the logic employed to decidable fragments. Overviews of decidable results for SL are available in [8,26].

Each benchmark instance of SL-COMP refers to one of the sub-logics of the multi-sorted Separation Logic. These sub-logics identify fragments which are handled by at least two participants or have been identified to be of interest during the discussion for the organization of the round.

The sub-logics are named using groups of letters, in a way similar to SMT-LIB format. These letters have been chosen to evoke the restrictions used by the sub-logics:

- `QF` for the restriction to quantifier free formulas;
- `SH` for the so-called "symbolic heap fragment" where formulas are restricted to (Boolean and separating) conjunctions of atoms and do not contain magic wand; moreover, pure atoms are only equality or dis-equality atoms;
- `LS` where the only predicate allowed is the acyclic list segment, `ls`, defined in Eqs. 4 and 5;
- `ID` for the fragment with user defined predicates;
- `LID` for the fragment of linear user defined predicates, i.e., only one recursive call for all rules of a predicate is allowed;
- `B` for the ground fragment allowing any Boolean combination of atoms.

Moreover, the existing fragments defined in SMT-LIB are used to further restrict the theory signature. For example, `LIA` denotes the signature for linear integer arithmetics.

Table 1. Divisions at SL-COMP and the participants enrolled

Division	size	Solvers enrolled
`qf_bsl_sat`	46	CVC4-SL
`qf_bsllia_sat`	24	CVC4-SL
`qf_shid_entl`	312	Cyclist-SL, Harrsh, S2S, Sleek, Slide, Songbird, Spen
`qf_shid_sat`	99	Harrsh, S2S, Sleek, SlSat
`qf_shidlia_entl`	75	ComSPEN, S2S
`qf_shidlia_sat`	33	ComSPEN, S2S
`qf_shlid_entl`	60	ComSPEN, Cyclist-SL, Harrsh, S2S, Spen
`qf_shls_entl`	296	Asterix, ComSPEN, Cyclist-SL, Harrsh, S2S, Spen
`qf_shls_sat`	110	Asterix, ComSPEN, Cyclist-SL, Harrsh, S2S, Spen
`shid_entl`	73	Cyclist-SL, S2S, Sleek, Songbird
`shidlia_entl`	181	S2S, Songbird

The current round of the competition has eleven divisions, named by concatenation of the name of the logic and the kind of problem solved (`sat` or `entl`). Table 1 provides the names of these divisions and the number of problems in each division:

- `qf_bsl_sat` and `qf_bsllia_sat` divisions include satisfiability problems for quantifier free formulas in the ground logic using respectively none or LIA logic for pure formulas.
- `qf_shid_entl` and `qf_shid_sat` divisions include entailment respectively satisfiability problems for the symbolic heap fragment with user defined predicates.
- `qf_shidlia_entl` and `qf_shidlia_sat` divisions include entailment respectively satisfiability problems for the quantifier free, symbolic heap fragment with user defined predicates and linear arithmetics included in pure formulas even in the predicate definitions.
- `qf_shlid_entl` division includes a subset of problems of division `qf_shid_entl` where the predicates are "linear" and compositional [10]. This fragment is of interest because the entailment problem has an efficient decision procedure.
- `qf_shls_entl` and `qf_shls_sat` divisions include entailment respectively satisfiability problems for the quantifier free symbolic heap fragment with only `ls` predicate atoms.
- `shid_entl` division contains entailment problems for quantified formulas in the symbolic heap fragment with general predicate definitions and no other logic theories than Boolean.
- `shidlia_entl` divisions extends the problems in `shid_entl` with constraints from linear integer arithmetics.

3.4 Selection Process

The benchmark set was built mainly from the contributions of participants. Some of these problems come from academic software analysis or verification tools based on SL (e.g., SmallFoot [30], Hip [5]). We did not received any problem issued from industrial tools. The problems were collected in the input format submitted by the participants and then translated into the input format of the competition. With the objective of increasing the size of the benchmark set, we did not limit the number of problems submitted by a participant. In this way, the edition 2018 has seen an increase of 100% in the size of the benchmark set. However in the future we could consider a change in the regulations to find a fair balance between teams. By using the meta-information in the preamble of each file, we are able to track the team which proposed the problem.

Notice that each problem has been examined by the organizer to ensure that the input format is respected and that it passed the parsing and type checking. However, the organizer accepts the status of the problem proposed until it is signaled incorrect by another team. In this case, a round of discussion is initiated to find an agreement on the status included in the file. Notice that the status

(sat or unsat) shall be known because it is important for the computation of the final result. The status of each problem was checked before the competition using at least two solvers and it did not change during the competition.

4 Participants

Eleven solvers are enrolled for this edition of the competition after its public announcement. Table 1 summarizes the enrollment of each solver in the divisions presented in the previous section.

4.1 Asterix

Asterix is presented in details in [21]. It was submitted by Juan Navarro Perez (at the time at University College London, UK, now at Google) and Andrey Rybalchenko (at the time at TU Munich, Germany, now at Microsoft Research Cambridge, UK). The solver deals with the satisfiability and entailment checking in the `QF_SHLS` fragment. For this, it implements a model-based approach. The procedure relies on SMT solving technology (Z3 solver is used) to untangle potential aliasing between program variables. It has at its core a matching function that checks whether a concrete valuation is a model of the input formula and, if so, generalizes it to a larger class of models where the formula is also valid.

Asterix was the winner of divisions `qf_shls_sat` and `qf_shls_entl` for both editions.

4.2 ComSPEN

The theoretical bases of ComSPEN have been presented in [11]. The development team is composed of Taolue Chen (University of London, UK), Chong Gao and Zhilin Wu (State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences).

The solver deals with both satisfiability and entailment problems in a fragment included in logic `QF_SHIDLIA` and which extends `QF_SHLID` with integer linear arithmetics in predicate definitions. The underlaying technique for satisfiability checking of a formula φ is to define an abstraction, $Abs(\varphi)$, where Boolean variables are introduced to encode the spatial part of φ, together with quantifier-free formulae to represent the transitive closure of the data constraints in the predicate atoms. Checking satisfiability of φ is then reduced to checking satisfiability of $Abs(\varphi)$, which can be solved by the state-of-the-art SMT solvers (e.g., Z3), with an NP upper-bound. For the entailment problem $\varphi \vdash \psi$, if φ and ψ are satisfiable, the procedure builds graphs for each formula and tries to build a graph isomorphism between them.

ComSPEN is implemented in C++. It uses the libraries Z3 and `boost`. The input format is the Spen's format, which requires a pre-processor for the competition's input format. Results are not available for ComSPEN because the 2019 edition is the first one for it.

4.3 Cyclist-SL

Cyclist-SL [4,7] was submitted by Nikos Gorogiannis (Middlesex University London, UK) in 2014. The solver deals with the entailment checking for the `QF_SLID` fragment. It is an instantiation of the theorem prover Cyclist-SL for the case of Separation Logic with inductive definitions. The solver builds derivation trees and uses induction to cut infinite paths in these trees that satisfy some soundness condition. For the Separation Logic, Cyclist-SL replaces the rule of weakening used in first-order theorem provers with the frame rule of SL.

Cyclist-SL won the division `qf_slid_entl` in 2014 and was at the second place in the same division in 2018.

4.4 CVC4-SL

CVC4 has a decision procedure described in [23] for the fragment `QF_BSL`. The solver CVC4-SL has been submitted by Andrew Reynolds (The University of Iowa, USA). Although this fragment is not supported by other solvers, two divisions were created for it because this fragment is the only one including the separating wand operator. CVC4-SL [6] participated in the 2018 edition and trivially won the two divisions.

4.5 Harrsh

Harrsh [17] was submitted by Jens Katelaan (TU Wien, Austria), the development team including Florian Zuleger from the same institute and Christoph Matheja and Thomas Noll (RWTH Aachen University, Germany). Harrsh deals with the fragment `QF_SHID` for both satisfiability and entailment checking. The decision procedures use a novel automaton model, so-called heap automata [16], which works directly on the structure of symbolic heaps. A heap automaton examines a SID bottom-up, starting from the non-recursive base case. At each stage of this analysis, a heap automaton remembers a fixed amount of information. Heap automata enjoy a variety of closure properties (intersection, union and complementation).

Harrsh is licensed under the MIT license and available on GitHub [12]. Harrsh was implemented in Scala and runs on the JVM. Harrsh has its own input format, but also supports both Cyclist-SL input format and the SL-COMP input format. Many SL-COMP entailment problems violate the syntactic restrictions of predicate definitions required by Harrsh. For this reason, the solver comes with a preprocessor that is able to transform many (but not all) benchmark's problems in the division `qf_shid_entl` into equivalent, Harrsh compatible specifications.

Harrsh entered SL-COMP in 2018 and competed in divisions `qf_shls_sat` and `qf_shid_sat` with encouraging results. Compared to all other participants, Harrsh has the disadvantage that it runs on the JVM: On simple problems, more than 99% of the runtime of Harrsh is spent starting and shutting down the JVM.

4.6 S2S

S2S is a solver submitted by Quang Loc Le (Teesside University, Middlesbrough, UK). It supports separation logic extended with string and arithmetic constraints, which correspond to all divisions of SL-COMP except ones based on QF_BSL. The solver is built around a generic framework to construct a forest of disjoint cyclic reduction trees for an input, either an entailment or a satisfiability problem. The implementation is done in Ocaml, from scratch. It contains three main components: front end with parsers, the proof systems and backend with SMT solvers (Z3). For the front end, the solver supports several formats, including the one of SL-COMP. The solver implements three concrete cyclic proof systems. The first system is a satisfiability solver in separation logic with general inductive predicates and arithmetic (fragment SLIDLIA). The second one is an entailment solver in the same fragment of separation logic above. Its implementation is the extension of a cyclic proof system with lemma synthesis [18]. The last system is a satisfiability solver for string logics. In all these three systems, any input of the leaf node evaluation method could be transformed into Presburger arithmetic and discharged efficiently by Z3.

In SL-COMP'2018, S2S won division qf_shlid_entl and qf_shidlia_sat.

4.7 Sleek

Sleek [5,28] participated in all editions of SL-COMP, the submitters at edition 2018 being Benjamin Lee and Wei-Ngan Chin (NUS, Singapore). The solver deals with the satisfiability and entailment checking for the QF_SHID fragment. It is an (incomplete but) automatic prover, that builds a proof tree for the input problem by using the classical inference rules and the frame rule of SL. It also uses a database of lemmas for the inductive definitions in order to discharge the proof obligations on the spatial formulas. The proof obligations on pure formulas are discharged by external provers like CVC4, Mona, or Z3.

Sleek was the winner of the division qf_shid_entl in edition 2014, and was in the third position in the same division in edition 2018.

4.8 Slide

Slide [14,29] was submitted by Adam Rogalewicz (FIT, Brno University of Technology, Czechia), the development team including Michal Cyprian and Tomas Vojnar from the same institute and Radu Iosif (Verimag, University Grenoble Alpes & CNRS, France). The solver deals with the entailment problem in the decidable sub-fragment of QF_SLID defined in [13]. The main principle of SLIDE is a reduction of entailment problems in SL into inclusion problems of tree automata. For the problems in the fragment identified in [13], the decision procedure implemented in Slide is EXPTIME-hard. More precisely, the proof method for checking $\varphi \Rightarrow \psi$ relies on converting φ and ψ into two tree automata A_φ resp. A_ψ, and checking the tree language inclusion of the automaton A_φ in the automaton A_ψ.

Slide takes an input in its own input format, which can be generated by the dedicated SL-COMP preprocessor. The reduction from the system of predicates into tree automata and the join operator is implemented in Python3. The result of the reduction are input files for the VATA tree automata library, which is used as a backend for the inclusion tests.

Slide participated in both past editions of SL-COMP. In 2018 edition, Slide solved 61 of 312 problems in division `qf_shid_entl`, 7 of 60 problems in division `qf_shlid_entl`, and 15 of 73 problems in division `shid_entl`. The number of solved problems is related to the fact that Slide is a prototype implementation, where our primary goal was to show the advantages of automata techniques. In order to cover more problems, one have to implement a new top-level parser, which would split the input entailment query into a set of subsequent queries, for which the automata-based technique can be used.

4.9 SlSat

SlSat [3] was submitted at SL-COMP'2014 by Nikos Gorogiannis (Middlesex University London, UK) and Juan Navarro Perez (at the time at UCL, UK, now at Google). The solver deals with the satisfiability problem for the `QF_SLID` fragment. The decision procedure is based on a fixed point computation of a constraint, called the "base" of an inductive predicate definition. This constraint is a conjunction of equalities and dis-equalities between a set of free variables built also by the fixed point computation from the set of inductive definitions.

SlSat was at the second position in division `qf_slid_sat` in edition 2014, and won this division at edition 2018.

4.10 Songbird

Songbird [32] was submitted by Quang-Trung Ta (National University of Singapore) and the development team includes Ton-Chanh Le (Stevens Institute of Technology, USA), Thanh-Toan Nguyen, Siau-Cheng Khoo, and Wei-Ngan Chin (National University of Singapore, Singapore). Songbird targets `SHIDLIA` fragment. It employs mathematical induction to prove entailments involving user-defined predicates. In addition, Songbird is also equipped with powerful proof techniques, which include a mutual induction proof system [35] and a lemma synthesis framework [36].

Songbird is implemented in OCaml and uses Z3 as the underlying SMT solver for the first-order logic formula which contains equality and linear arithmetic constraints. The input syntax of Songbird is described in [32].

Songbird integrated SL-COMP at the 2018 edition, and was the first in four divisions: `qf_shid_entl`, `qf_shidlia_entl`, `shid_entl`, `shidlia_entl`. It can also solve 100% of the problems in other two divisions `qf_shls_entl` and `qf_shls_sat`, but the runtime is slower than the best provers of these divisions.

4.11 Spen

Spen [9,33] was submitted by Mihaela Sighireanu (IRIF, University Paris Diderot & CNRS, France) and the development team includes Constantin Enea from the same institute, Ondrej Lengal and Tomas Vojnar (FIT, Brno University of Technology, Czechia). The solver deals with satisfiability and entailment problems for the fragments `QF_SHLID` and `QF_SHLS`. The decision procedures call the MiniSAT solver on a Boolean abstraction of the SL formulas to check their satisfiability and to "normalize" the formulas by inferring its implicit (dis)equalities. The core of the algorithm checking if $\varphi \Rightarrow \psi$ is valid searches a mapping from the atoms of ψ to sub-formulas of φ. This search uses the membership test in tree automata to recognize in sub-formulas of φ some unfolding of the inductive definitions used in ψ.

Spen is written in C and C++ and is open source [33]. It is based on the VATA library for tree automata. Spen won the division `qf_shlid_entl` at edition 2014 and was in the second position in divisions `qf_shls_entl` and `qf_shls_sat` in both editions.

5 Running the Competition

SL-COMP uses the StarExec platform [34] and requires several features provided by this platform. The pre-processing phase allows to translate each problem into the input format of the solver without time penalties. It is used by most of the solvers and some pre-processors are provided by SL-COMP's organizer, freely available on the competition GitHub repository [22]. The competition did not use the scrambling of benchmark's problems because the names used for inductive definitions defined in the files of some divisions are important for the solvers. Each benchmark file includes only one problem. The incremental feature was not used and is not supported by most of the competing solvers.

StarExec imposes a time and space limit on each attempt of a solver to solve a given problem. For the 2014 edition, the CPU time was limited to 2400 s and the memory (RAM) limit was 100 GB. To gain some time in running the competition, the 2018 edition used by default a timeout of 600 s and 4 GB of memory; if the time was exceeded, timeouts of 2400 then 3600 were tried. Even with these bigger timeouts, some jobs did have CPU timeout or reached the memory limit. To simplify the running, the new edition will use a memory limit of 100 GB and a timeout of 3600 s.

The participants trained their solvers on the platform and provided feedback where the expected result of a problem did not match their result. Several benchmark's problems and solvers were fixed during this period. One training run was executed before the official run to provide insights about the global results and to do a final check of the benchmark set.

The participants at each divisions are ordered according to the rules fixed for SMT-COMP'14 edition. The best solver is the one with, in order: (a) the least number of incorrect answers, (b) the largest number of correctly solved problems, and (c) the smallest time taken in solving the correctly solved problems.

Note that solvers are allowed to respond "unknown" or to time-out on a given benchmark's problem without penalty (other than not being able to count that problem as a success).

StarExec requires that a public version of a solver be made available on StarExec as a condition of participating in a competition. This allows users of StarExec to rerun a competition if so desired. More importantly, it allows users to upload a new set of problems of interest in their application domain and to try the various solvers against those problems. This feature was very useful for SL-COMP at edition 2018, because some solvers reused the binaries submitted in 2014. The results of the competition are provided on the competition web page with a link to the CSV files generated by StarExec. We are also archiving the results of previous editions in the GitHub.

6 Impact and Perspectives

The SL-COMP initiative fulfilled its goals: an interesting suite of SL problems is publicly available in a common format and the maturity of solvers submitted for this competition has been proven.

Moreover, we achieved to propose a common format for SL which is based on a mature and maintained format for first-order theories, SMT-LIB. This format reveals the features required by the existing solvers, e.g., the strong typing of formulas, the kind of inductive definitions handled, etc.

The participation at SL-COMP allowed to measure solvers against competitors and therefore to improve solvers during the competition and in meantime. Moreover, the existing benchmark set includes challenging problems for the competitors because about half (6 over 11) of the divisions are completely solved. Five divisions include problems not yet dealt: `qf_bsl_sat` has 2 problems (5%), `qf_shid_entl` has 11 problems (4%), `qf_shid_sat` has 26 problems (27%), `shid_entl` has 3 problems (5%) and `shid_sat` has 29 problems (17%).

A community interested in such tools has been identified and informed about the status of the existing solvers. This community could benefit from improving the tools built on the top of decision procedures for SL.

The SMT-COMP community discovered the status of the solvers for SL and became interested in this theory, as is demonstrated by the participation of CVC4, one of the most complete solver of SMT-COMP.

We expect that the 2019 edition of SL-COMP will enforce these results.

The perspectives mainly concern improvement of the organization process as the size of the competition (number of solvers and benchmark set) increases.

First of all, we are trying to reach a consensus for a good cadence of this competition. Yearly competitions could be very exciting for the first years, but may focus on engineering improvements rather than fundamental work. We feel that a good cadence is alternating a competition year with a year of benchmark set evaluation and improvement.

With the experience of the current competition, the benchmark set has to be improved also. As mentioned above, we have to balance the number of problems

coming from the same team in each division in order to reach a fair comparison criterium. For each problem, it would be interesting to attach a coefficient which is taken into account in the scoring system and thus obtain a better evaluation of each solver. A classic way to assign a difficulty level is to take into account the size of the formulas and of the inductive definitions used in the problem.

Finally, we should intensify the exchanges with related competitions in software verification and automated proving. Such competitions may benefit from SL-COMP results in terms of automation, and may provide interesting benchmark sets. For this, the results of SL-COMP should be made available in forms that allows to understood the state of the art of SL solvers and the contribution of each participating solver to this state of the art. We should also provide, in addition to the StarExec platform, other means to reproduce the results of each edition. For example, virtual machines may be archived with the sources and binaries of participants for each edition of the competition.

References

1. Antonopoulos, T., Gorogiannis, N., Haase, C., Kanovich, M., Ouaknine, J.: Foundations for decision problems in separation logic with general inductive predicates. In: Muscholl, A. (ed.) FoSSaCS 2014. LNCS, vol. 8412, pp. 411–425. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54830-7_27
2. Barrett, C., Stump, A., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB) (2018). www.SMT-LIB.org
3. Brotherston, J., Fuhs, C., Navarro Pérez, J.A., Gorogiannis, N.: A decision procedure for satisfiability in separation logic with inductive predicates. In: CSL-LICS, pp. 25:1–25:10. ACM (2014)
4. Brotherston, J., Gorogiannis, N., Petersen, R.L.: A generic cyclic theorem prover. In: Jhala, R., Igarashi, A. (eds.) APLAS 2012. LNCS, vol. 7705, pp. 350–367. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35182-2_25
5. Chin, W.-N., David, C., Nguyen, H.H., Qin, S.: Automated verification of shape, size and bag properties via user-defined predicates in separation logic. Sci. Comput. Program. **77**(9), 1006–1036 (2012)
6. CVC4-SL. http://cvc4.cs.stanford.edu/wiki/Separation_Logic
7. CYCLIST. https://github.com/ngorogiannis/cyclist
8. Demri, S., Deters, M.: Separation logics and modalities: a survey. J. Appl. Non-Classical Logics **25**(1), 50–99 (2015)
9. Enea, C., Lengál, O., Sighireanu, M., Vojnar, T.: Compositional entailment checking for a fragment of separation logic. In: Garrigue, J. (ed.) APLAS 2014. LNCS, vol. 8858, pp. 314–333. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12736-1_17
10. Enea, C., Sighireanu, M., Wu, Z.: On automated lemma generation for separation logic with inductive definitions. In: Finkbeiner, B., Pu, G., Zhang, L. (eds.) ATVA 2015. LNCS, vol. 9364, pp. 80–96. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24953-7_7
11. Gu, X., Chen, T., Wu, Z.: A complete decision procedure for linearly compositional separation logic with data constraints. In: Olivetti, N., Tiwari, A. (eds.) IJCAR 2016. LNCS (LNAI), vol. 9706, pp. 532–549. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40229-1_36

12. Harrsh. https://github.com/katelaan/harrsh
13. Iosif, R., Rogalewicz, A., Simacek, J.: The tree width of separation logic with recursive definitions. In: Bonacina, M.P. (ed.) CADE 2013. LNCS (LNAI), vol. 7898, pp. 21–38. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38574-2_2
14. Iosif, R., Rogalewicz, A., Vojnar, T.: Deciding entailments in inductive separation logic with tree automata. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 201–218. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11936-6_15
15. Iosif, R., Serban, C., Reynolds, A., Sighireanu, M.: Encoding separation logic in smt-lib v2.5. (2018). https://github.com/sl-comp/SL-COMP18/input/Docs
16. Jansen, C., Katelaan, J., Matheja, C., Noll, T., Zuleger, F.: Unified reasoning about robustness properties of symbolic-heap separation logic. In: Yang, H. (ed.) ESOP 2017. LNCS, vol. 10201, pp. 611–638. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54434-1_23
17. Katelaan, J., Matheja, C., Noll, T., Zuleger, F.: Harrsh: a tool for unied reasoning about symbolic-heap separation logic. In: Barthe, G., Korovin, K., Schulz, S., Suda, M., Sutcliffe, G., Veanes, M., (eds.) LPAR-22 Workshop and Short Paper Proceedings. Kalpa Publications in Computing, vol. 9, pp. 23–36. EasyChair (2018)
18. Le, Q.L., Sun, J., Qin, S.: Frame inference for inductive entailment proofs in separation logic. In: Beyer, D., Huisman, M. (eds.) TACAS 2018. LNCS, vol. 10805, pp. 41–60. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89960-2_3
19. O'Hearn, P.: Separation logic. http://www0.cs.ucl.ac.uk/staff/p.ohearn/SeparationLogic/Separation_Logic/SL_Home.html
20. O'Hearn, P., Reynolds, J., Yang, H.: Local reasoning about programs that alter data structures. In: Fribourg, L. (ed.) CSL 2001. LNCS, vol. 2142, pp. 1–19. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44802-0_1
21. Navarro Pérez, J.A., Rybalchenko, A.: Separation logic modulo theories. In: Shan, C. (ed.) APLAS 2013. LNCS, vol. 8301, pp. 90–106. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-03542-0_7
22. SL-COMP Repository. https://github.com/sl-comp
23. Reynolds, A., Iosif, R., Serban, C., King, T.: A decision procedure for separation logic in SMT. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 244–261. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46520-3_16
24. Reynolds, J.C.: Intuitionistic reasoning about shared mutable data structure. In: Oxford-Microsoft Symposium in Honour of Sir Tony Hoare. Palgrave Macmillan, Basingstoke (1999). Publication date November 2000
25. Reynolds, J.C.: Separation logic: a logic for shared mutable data structures. In: LICS, pp. 55–74. IEEE Computer Society (2002)
26. Sighireanu, M., Cok, D.: Report on SL-COMP 2014. JSAT **9**, 173–186 (2014)
27. SL-COMP 2018. https://www.irif.fr/∼sighirea/sl-comp/18/
28. SLEEK. http://loris-7.ddns.comp.nus.edu.sg/∼project/s2/beta/
29. SLIDE. http://www.fit.vutbr.cz/research/groups/verifit/tools/slide/
30. SmallFoot. http://www0.cs.ucl.ac.uk/staff/p.ohearn/smallfoot/
31. SMT-COMP. http://smtcomp.sourceforge.org
32. Songbird. https://songbird-prover.github.io/
33. SPEN. https://www.github.com/mihasighi/spen
34. StarExec. http://www.starexec.org

35. Ta, Q.-T., Le, T.C., Khoo, S.-C., Chin, W.-N.: Automated mutual explicit induction proof in separation logic. In: Fitzgerald, J., Heitmeyer, C., Gnesi, S., Philippou, A. (eds.) FM 2016. LNCS, vol. 9995, pp. 659–676. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48989-6_40
36. Ta, Q.-T., Le, T.C., Khoo, S.-C., Chin, W.-N.: Automated lemma synthesis in symbolic-heap separation logic. Proc. ACM Program. Lang. **2**(POPL), 9:1–9:29 (2017)