SeLoger: A Tool for Graph-Based Reasoning in Separation Logic

Christoph Haase*[1], Samin Ishtiaq[2], Joël Ouaknine[3], and Matthew J. Parkinson[2]

[1] LSV – CNRS & ENS Cachan, France
[2] Microsoft Research Cambridge, UK
[3] Department of Computer Science, University of Oxford, UK

Abstract. This paper introduces the tool SeLoger, which is a reasoner for satisfiability and entailment in a fragment of separation logic with pointers and linked lists. SeLoger builds upon and extends graph-based algorithms that have recently been introduced in order to settle both decision problems in polynomial time. Running SeLoger on standard benchmarks shows that the tool outperforms current state-of-the-art tools by orders of magnitude.

1 Introduction

Tools based on separation logic [4,6] have shown tremendous promise when applied to the problem of formal verification in the presence of mutable data-structures. For example, shape analysis tools such as SpaceInvader, Thor, SLAyer or Infer are nowadays being applied to a range of low-level industrial systems code. Inside, these shape analysis tools mix traditional abstract interpretation techniques (*e.g.* custom abstract joins) combined with entailment procedures for restricted subsets of separation logic. Thus, one method for improving the scalability and applicability of these tools is to improve the underlying entailment or other decision procedures. This has been an active area of recent research, see *e.g.* [2,3,5].

Recently, we have shown in [3] that entailment in the fragment of separation logic with pointers and linked lists can be decided in polynomial time. This fragment was introduced in [1], and it forms the basis of a number of tools such as Smallfoot, SpaceInvader, and SLAyer. Traditionally, the separation logic reasoners integrated in those tools decide entailment via a syntactic proof search. In contrast, the decision procedure presented in [3] takes a different approach which is based on graph-theoretical methods.

In this paper, we introduce the tool SeLoger (SEparation LOgic GraphbasEd Reasoner) which implements an extension of the decision procedures presented in [3]. In Section 4, we compare SeLoger to the tool SLP by Navarro

* Parts of the research were carried out while the author was an intern at Microsoft Research Cambridge, UK, and while the author held an EPSRC PhD+ fellowship at the Department of Computer Science, University of Oxford, UK. The author is supported by the French Agence Nationale de la Recherche, ReacHard (grant ANR-11-BS02-001).

Peréz and Rybalchenko [5]. They show that SLP outperforms the reasoners in SMALLFOOT and JSTAR by several orders of magnitude, and we can show that SELOGER outperforms SLP by orders of magnitude.

Recently, in [2] Bouajjani *et al.* have introduced the tool SLAD which also builds upon some of the ideas presented in [3]. One difference to our tool is that it decides entailment under *intuitionistic* semantics, which is also the semantic model considered in [3]. In contrast, the semantic model dealt with in [1] is *non-intuitionistic*, and the decision procedure implemented in SELOGER extends the one presented in [3] in a non-trivial way in order to decide entailment under this semantic model. We also fixed in our implementation some subtle issues we discovered in the algorithm from [3]. Although SELOGER can decide entailment under intuitionistic semantics, since our target semantic model is the one presented in [1], we do not compare SELOGER to SLAD. We do not expect major differences to arise when comparing SLAD to SELOGER on the intersection of the logical languages supported by the tools.

2 Separation Logic

SELOGER decides satisfiability and entailment in the fragment of separation logic introduced in [1]. The syntax of the assertion language of this fragment is given by the following grammar, where x and y range over an infinite set of *variables*:

$$\begin{aligned}
\phi &::= \top \mid \bot \mid x = y \mid x \neq y \mid \phi \wedge \phi && (\textit{pure formulas})\\
\sigma &::= \mathsf{emp} \mid \mathsf{true} \mid \mathsf{pt}(x, y) \mid \mathsf{ls}(x, y) \mid \sigma * \sigma && (\textit{spatial formulas})\\
\alpha &::= (\phi; \sigma) && (\textit{assertions})
\end{aligned}$$

We call assertions of our assertion language *SL formulas*. For brevity, we only informally introduce the semantics of SL formulas. In [1], SL formulas are interpreted over memory models consisting of a *heap* and a *stack*. A heap is a function mapping a finite subset of an infinite domain of *heap cells* (usually the naturals) to heap cells. The elements of the domain of a heap are called *allocated heap cells*. A stack maps a finite subset of variables to heap cells, *i.e.*, it labels heap cells with variables. Pure formulas make Boolean judgements about stacks in the obvious way, *e.g.* a stack models $x = y$ if x and y are mapped to the same heap cell. Spatial formulas on the other hand make judgements about the shape of the heap. With emp a heap is required to have no allocated heap cells; true holds always; the *points-to relation* $\mathsf{pt}(x, y)$ requires that the heap consists of a single allocated cell labelled with x that maps to the heap cell labelled with y; and the *list relation* $\mathsf{ls}(x, y)$ requires that there be a chain of connected allocated heap cells starting in x and ending in y *with no repetitions*. Finally, $\sigma_1 * \sigma_2$ holds for a heap if the set of allocated heap cells can be partitioned into disjoint sets such that σ_1 holds on the first partition and σ_2 on the second. Last, given a memory model, an assertion $(\phi; \sigma)$ holds if the stack is a model of ϕ and the heap a model of σ.

Given SL formulas α, α', *satisfiability* asks whether there is a memory model in which α holds, and *entailment* is to decide whether α' holds in every memory

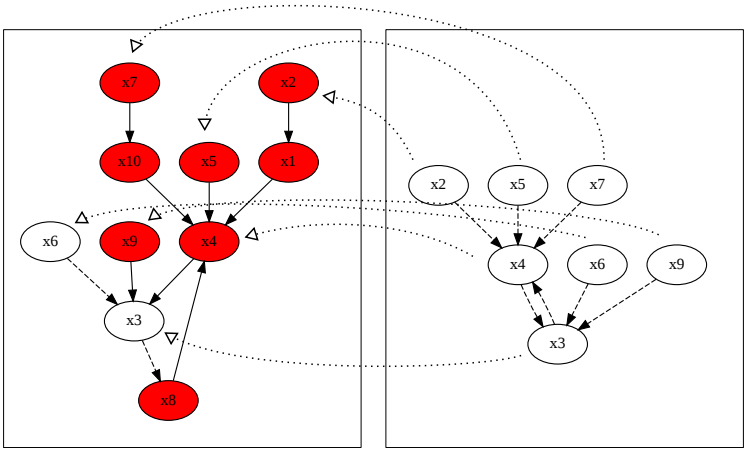

Fig. 1. Example of two SL graphs G_1 and G_2 and a graph homomorphism between them witnessing entailment between the corresponding SL formulas.

model in which α holds, written $\alpha \models \alpha'$. We call α the *assumption* and α' the *goal* of the entailment.

3 A Sketch of the Graph-Based Entailment Algorithm

The key idea of the algorithm presented in [3] is to represent SL formulas as directed labelled coloured graphs (SL graphs). Entailment can then be decided by checking whether a canonical mapping from the set of nodes of the graph representing the goal to the set of nodes representing the assumption fulfils certain *homomorphism conditions*.

For brevity, instead of providing formal definitions, let us illustrate the representation of SL formulas as SL graphs and a homomorphism witnessing entailment between the formulas with the help of an example which can be found in Figure 1. The graph G_1 in the left-hand side box represents the SL formula $\alpha_1 = (\top; \sigma_1)$ and the graph G_2 in the right-hand side box the SL formula $\alpha_2 = (\top; \sigma_2)$, where

$$\sigma_1 = \mathsf{pt}(x_7, x_{10}) * \mathsf{pt}(x_2, x_1) * \mathsf{pt}(x_{10}, x_4) * \mathsf{pt}(x_5, x_4) * \mathsf{pt}(x_1, x_4) * \mathsf{pt}(x_4, x_3) * \\ * \mathsf{pt}(x_9, x_3) * \mathsf{pt}(x_8, x_4) * \mathsf{ls}(x_6, x_3) * \mathsf{ls}(x_3, x_8); \text{ and}$$

$$\sigma_2 = \mathsf{ls}(x_2, x_4) * \mathsf{ls}(x_5, x_4) * \mathsf{ls}(x_7, x_4) * \mathsf{ls}(x_4, x_3) * \mathsf{ls}(x_3, x_4) * \mathsf{ls}(x_6, x_3) * \mathsf{ls}(x_9, x_3).$$

Both SL formulas belong to an actual entailment instance found in the benchmark suite used in this paper and have been generated by SeLoger using GraphViz. The points-to relation is represented by solid arrows and the list relations by dashed arrows, nodes of the graphs correspond to equivalence classes of variables (here, each equivalence class is a singleton). The canonical mapping

h is represented by dotted arrows and witnesses that each list edge in G_2 has a corresponding path in G_1. For example, there is a path from the node labelled with x_3 to the node labelled with x_4 in G_1, which is required by the list edge from x_3 to x_4 in G_2. Furthermore, no edge in G_1 occurs along two paths that are induced by two different list edges of G_2 under h, and all edges in G_1 occur along a path that is induced by a list edge of G_2 under h. Together with some further technical conditions, we can show that h is a homomorphism and that consequently α_1 entails α_2.

In general, the SL graphs corresponding to SL formulas do not exhibit such a nice structure as the ones presented in Figure 1. However, it is shown in [3] that any SL formula α is equivalent to an SL formula α' whose corresponding SL graph G enjoys some nice structural properties, *e.g.* that between any two nodes there is at most one loop-free path. In [3], a saturation procedure (there called reduction procedure) is presented that given α computes such a graph G if α is satisfiable, and indicates that α is unsatisfiable otherwise. In summary, the decision procedure for an entailment $\alpha_1 \models \alpha_2$ presented in [3] can be broken into three parts:

(i) Construction of the SL graphs G_1 and G_2 representing α_1 and α_2
(ii) Saturation of α_1 and α_2 (which gives a satisfiability test as a byproduct)
(iii) Checking whether the canonical mapping from the nodes of G_2 to the nodes of G_1 is a homomorphism

4 Experimental Evaluation

We have tested SeLoger against the tool SLP, rev. 13591, by Navarro Peréz and Rybalchenko [5] on the benchmark suite[4] used in the same paper and one class of benchmarks generated by us. In [5], the authors compare SLP to the reasoners used in jStar and Smallfoot. Since SLP significantly outperforms both jStar and Smallfoot on essentially all test cases, we decided to only benchmark SeLoger against SLP.

The benchmarks suite in [5] consists of three classes of benchmarks called "spaguetti", "bolognesa" and "clones". The class "bolognesa" consists of 11 files, each containing 1000 entailment checks of the form $\alpha \models \alpha'$. Both α and α' are SL formulas which are generated at random according to some rules specified in [5]. Initially, both SL formulas range over ten variables and this number is increased in each file by one such that the last "bolognesa" file contains 1000 entailment checks over formulas with 20 variables. Similarly, the "spaguetti" class contains 11 files with 1000 entailment checks of the form $\alpha \models \bot$, where α is generated at random and the number of variables used in α increases by one starting from 10. In both classes, the random instances are chosen such that roughly 50% of the entailments are valid. Finally, the "clones" class contains real-world

[4] The benchmark suite can be downloaded at `http://navarroj.com/research/tools/slp-benchmarks.tgz`

B.mark	SLP	SeLoger	B.mark.	SLP	SeLoger	B.mark	SLP	SeLoger	B.mark	SLP	SeLoger
bo-10	1410	291	sp-10	1240	255	cl-01	65	14	aw-01	23	1
bo-11	1781	341	sp-11	2214	297	cl-02	67	20	aw-02	25	2
bo-12	2421	439	sp-12	8181	348	cl-03	82	26	aw-03	28	2
bo-13	11.9k	442	sp-13	15.6k	391	cl-04	93	34	aw-04	33	3
bo-14	5862	467	sp-14	15.2k	408	cl-05	117	44	aw-05	43	3
bo-15	3937	495	sp-15	18.6k	438	cl-06	147	52	aw-06	64	4
bo-16	7156	546	sp-16	3503	442	cl-07	207	62	aw-07	127	5
bo-17	14.2k	571	sp-17	94.2k	517	cl-08	364	72	aw-08	345	6
bo-18	20.8k	642	sp-18	5129	525	cl-09	826	84	aw-09	1157	7
bo-19	40.7k	705	sp-19	27.2k	549	cl-10	2466	95	aw-10	4492	8
bo-20	27.0k	752	sp-20	70.7k	595	cl-11	8794	105	aw-11	18.4k	10
						cl-12	34.2k	118	aw-12	76.2k	11
						cl-13	139.8k	130			

Table 1. Comparison of SLP and SeLoger on the benchmark set used in [5] and an additional class ("awkward"). All times are in ms.

entailments. It consists of 13 files[5], each containing 209 entailments that were extracted from verification conditions generated by Smallfoot when run on the examples shipped with the tool. Some of the entailments require an enriched syntax since they include arbitrary data fields. The algorithm presented in [3] can, however, be straight-forwardly generalised to also allow for data fields as required by the benchmarks. Since the verification conditions are of a rather simple nature, in order to increase the complexity the "clones" class incrementally adds copies of the entailments to each entailment such that in the last benchmark file, each entailment consists of 13 copies of the original entailment. Last, we generated a benchmark class called "awkward", where the n-th instance consists of a *single* entailment of the form $*_{1\leq i\leq n}\mathsf{ls}(x_i, y_i) * \mathsf{ls}(x_i, z_i) * \mathsf{ls}(y_i, w_i) * \mathsf{ls}(z_i, w_i) \models *_{1\leq i\leq n}\mathsf{ls}(x_i, y_i) * \mathsf{ls}(y_i, x_i) * \mathsf{ls}(x_i, z_i) * \mathsf{ls}(z_i, x_i)$.

SeLoger is written in F# and, according to [5], SLP is implemented in Prolog and was provided to us as a binary file. We ran the SeLoger benchmarks on a Samsung Series 9 ultrabook with an Intel® Core™ i5-2467M 1.60 GHz processor with 4 GB DDR3 1066 MHz under Windows® 7 Home Premium (64-bit) and the SLP benchmarks on the same machine under Ubuntu Linux 12.04.1.

The results of the benchmarks are shown in Table 1 and illustrated in Figure 2. In Table 1, each column contains the average running time over ten runs. For SeLoger, the average coefficient of variation encountered was 0.05 with a standard deviation of 0.05, and for SLP the average coefficient of variation was 0.04 with a standard deviation of 0.07. We observe that SeLoger finishes on all benchmarks in less than 800ms, that it is up to 1075 times faster on the benchmarks from [5], and that the running time encountered in praxis appears almost linear, while it grows exponentially for SLP. We created the "awkward" benchmarks with the intention of exaggerating this difference. Also note that SeLoger behaves in general more robustly in the sense that the running times monotonically increase with the complexity of the benchmarks.

[5] In [5], the "clones" class only consists of eight files, however for better comparison we generated the additional five files using the benchmark generator used in [5]

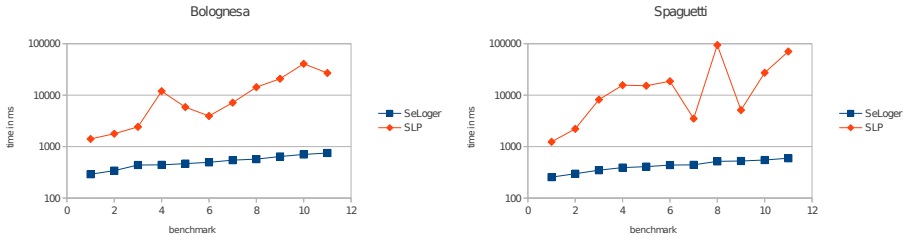

Fig. 2. Graphical illustration of some data from Table 1 on a logarithmic scale.

5 Conclusion

In this paper, we introduced the tool SeLoger which implements and extends the entailment algorithm for the fragment of separation logic with pointers and linked lists described in [3]. We compared our tool to the tool SLP by Navarro Peréz and Rybalchenko [5]. Our benchmarks show that SeLoger outperforms SLP on all benchmarks considered and is often orders of magnitudes faster.

Together with other tools such as SLAD [2] that are based on the graph-based approach to entailment checking from [3], this suggests that this approach not only yields new complexity results, but also delivers practically-usable and high-performance algorithms. We are confident that SeLoger can serve as a basis for future work on graph-based algorithms and decision procedures for even richer fragments of separation logic and will find its way into future program verifiers.

Acknowledgement. We would like to thank Juan Antonio Navarro Peréz for making SLP available to us.

References

1. Josh Berdine, Cristiano Calcagno, and Peter O'Hearn. A decidable fragment of separation logic. In *Proceedings of FSTTCS'04*, volume 3328 of *LNCS*, pages 110–117. Springer, 2005.
2. Ahmed Bouajjani, Cezara Drăgoi, Constantin Enea, and Mihaela Sighireanu. Accurate invariant checking for programs manipulating lists and arrays with infinite data. In *Proceedings of ATVA'12*, LNCS, pages 167–182. Springer, 2012.
3. Byron Cook, Christoph Haase, Joël Ouaknine, Matthew Parkinson, and James Worrell. Tractable reasoning in a fragment of separation logic. In *Proceedings of CONCUR'11*, volume 6901 of *LNCS*, pages 235–249. Springer, 2011.
4. Samin Ishtiaq and Peter O'Hearn. BI as an assertion language for mutable data structures. In *Proceedings of POPL'01*, pages 14–26. ACM, 2001.
5. Juan Antonio Navarro Peréz and Andrey Rybalchenko. Separation logic + Superposition calculus = Heap theorem prover. In *Proceedings of PLDI'11*, San Jose, CA, USA, 2011. ACM Press.
6. John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings of LICS'02*. IEEE Computer Society, 2002.