SLIDE: Separation Logic with Inductive DEfinitions

Michal Cyprian[1], Radu Iosif[2], Adam Rogalewicz[1], and Tomáš Vojnar[1]

[1] FIT, Brno University of Technology, IT4Innovations Centre of Excellence, Czech Republic
[2] University Grenoble Alpes, CNRS, VERIMAG, Grenoble, France

1 Introduction

SLIDE (Separation Logic with Inductive DEfinitions)[3] is an entailment prover for a fragment of Separation logic (SL) [2]. The main principle of SLIDE is a reduction of entailment problem in SL into inclusion problem of tree automata [1]. We identify a fragment of SL, for which our technique is sound and complete and prove the entailment problem on this fragment to be EXPTIME-hard.

2 Fragment of Separation logic

SLIDE can solve the entailment problem of the form $P_i(\mathbf{x}) \models_{\mathcal{P}} P_j(\mathbf{x})$ where P_i and P_j are predicates of the inductive system $\mathcal{P}$ defined bellow. We consider the following syntax of *basic formulae* of Separation Logic (SL):

$$\begin{array}{l} \alpha \ \in \ Var \setminus \{\mathbf{nil}\};\ x \ \in \ Var; \\ \Pi ::= \alpha = x \mid \Pi_1 \wedge \Pi_2 \\ \Sigma ::= \mathbf{emp} \mid \alpha \mapsto (x_1, \ldots, x_n) \mid \Sigma_1 * \Sigma_2 \text{ , for some } n > 0 \\ \varphi \ ::= \Sigma \wedge \Pi \mid \exists x \ . \ \varphi \end{array}$$

A system of *inductive definitions* (inductive system) $\mathcal{P}$ is a set of rules of the form

$$\left\{ P_i(x_{i,1}, \ldots, x_{i,n_i}) \equiv |_{j=1}^{m_i} \ R_{i,j}(x_{i,1}, \ldots, x_{i,n_i}) \right\}_{i=1}^{k} \tag{1}$$

where $\{P_1, \ldots, P_k\}$ is a set of *predicates*, $x_{i,1}, \ldots, x_{i,n_i}$ are called *formal parameters*, and the formulae $R_{i,j}$ are called the *rules* of P_i. Each rule is of the form $R_{i,j}(\mathbf{x}) \equiv \exists \mathbf{z} \ . \ \Sigma * P_{i_1}(\mathbf{y}_1) * \ldots * P_{i_m}(\mathbf{y}_m) \wedge \Pi$, where $\mathbf{x} \cap \mathbf{z} = \emptyset$, and the following holds:

1. $\Sigma \not\equiv \mathbf{emp}$ is a non-empty spatial formula, called the *head* of $R_{i,j}$. In practice, we allow frontier or root rules to have **emp**ty heads.
2. $P_{i_1}(\mathbf{y}_1), \ldots, P_{i_m}(\mathbf{y}_m)$ is a tuple of *predicate occurrences*, called the *tail* of $R_{i,j}$, where $|\mathbf{y}_j| = n_{i_j}$, for all $1 \leq j \leq m$.
3. Π is a pure formula, restricted such that, for all formal parameters $\beta \in \mathbf{x}$, we allow only equalities of the form $\alpha =_\Pi \beta$, where α is allocated in Σ.
4. for all $1 \leq r, s \leq m$, if $x_{i,k} \in \mathbf{y}_r$, $x_{i,l} \in \mathbf{y}_s$, and $x_{i,k} =_\Pi x_{i,l}$, for some $1 \leq k, l \leq n_i$, then $r = s$; a formal parameter of a rule cannot be passed to two or more subsequent occurrences of predicates in that rule.

[3] `http://www.fit.vutbr.cz/research/groups/verifit/tools/slide/`

3 Brief technical information

Underlying principle: The predicate calls $P(\mathbf{x})$ (resp. $R(\mathbf{x})$) in the entailment query $P(\mathbf{x}) \models_{\mathcal{P}} R(\mathbf{x})$ are translated to tree automata A_P (resp. A_R) such that $\mathcal{L}(A_P) \subseteq \mathcal{L}(A_R) \Rightarrow P(\mathbf{x}) \models_{\mathcal{P}} R(\mathbf{x})$. The translation is possible in theory for the whole fragment of SL defined in Sec. 2. The SLIDE implements a smaller fragment, where Σ in a rule can contain only a single points-to. When we restrict the fragment even more, we get a complete result—i.e. $\mathcal{L}(A_P) \subseteq \mathcal{L}(A_R) \Leftrightarrow P(\mathbf{x}) \models_{\mathcal{P}} R(\mathbf{x})$. The detailed description can be found in [1].
Join operation: In order to allow entailment queries of the form

$$\mathcal{Q} \equiv P_1(\mathbf{x_1}) * \cdots * P_n(\mathbf{x_n}) \models_{\mathcal{P}} R_1(\mathbf{y_1}) * \ldots * R_m(\mathbf{y_m}),$$

we have implemented a join operation, which allows us to translate a formula $P_1(\mathbf{x}) * P_2(\mathbf{y})$ into and an equivalent formula $P(\mathbf{x} \cup \mathbf{y})$. Using the join operator, we can translate entailment query of the form $\mathcal{Q}$ into the considered form $P(\mathbf{x}) \models_{\mathcal{P}} R(\mathbf{x})$. The translation is based on syntactic manipulations with the system of predicates $\mathcal{P}$ and it is an incomplete heuristics.
Implementation: SLIDE takes an input in its own simple format, which can be generated by SL-COMP preprocessor *SL-SLIDE*. 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[4], which is used as a backend for the inclusion tests.

4 Participation in SL-COMP'18

In SL-COMP'18[5], SLIDE participated in three divisions.

- **qf_shid_entl**: solved 61 of 312 benchmarks.
- **qf_shlid_entl**: solved 7 of 60 benchmarks.
- **shid_entl**: solved 15 of 73 benchmarks.

The number of solved benchmarks 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 benchmarks, 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.

References

1. R. Iosif, A. Rogalewicz, and T. Vojnar. Deciding entailments in inductive separation logic with tree automata. In *Proc. of ATVA'14*, volume 8837 of *LNCS*. Springer, 2014.
2. J. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *Proc. of LICS'02*. IEEE CS Press, 2002.

[4] `https://github.com/ondrik/libvata`
[5] `https://www.irif.fr/ sighirea/sl-comp/18/`