SPEN

Constantin Enea[1], Ondřej Lengál[2], Mihaela Sighireanu[1], and Tomáš Vojnar[2]

[1] Univ. Paris Diderot, LIAFA CNRS UMR 7089
[2] FIT, Brno University of Technology, IT4Innovations Centre of Excellence, Czech Republic

SPEN is an open source solver[3] for checking validity of entailments between formulas in a fragment of Separation Logic with recursive definitions.

1 Separation Logic Fragment

The logic fragment mainly contains existentially quantified Separation Logic formulas with spatial atoms representing complex list segments, for example doubly linked lists, nested linked lists, fixed depth skip lists, introduced by the recursive definitions.

More precisely, in an entailment $A \Rightarrow B$ dealt by SPEN, A and B are formulas in the "symbolic heaps" fragment of Separation Logic whose syntax is given by the following grammar:

$x, y \in \mathit{Vars}$ program variables $\quad X, Y \in \mathit{LVars}$ logical variables $\quad E, F ::= x \mid X$

$f \in \mathbb{F}$ fields $\quad \rho ::= (f, E) \mid \rho, \rho \quad P \in \mathbb{P}$ predicates

$\overrightarrow{B} \in (\mathit{Vars} \cup \mathit{LVars})^*$ vectors of variables

$\Pi ::= E = F \mid E \neq F \mid \Pi \wedge \Pi$ pure formulas

$\Sigma ::= \mathit{emp} \mid E \mapsto \{\rho\} \mid P(E, F, \overrightarrow{B}) \mid \Sigma * \Sigma$ spatial formulas

$A, B \triangleq \exists \overrightarrow{X}. \Pi \wedge \Sigma$ formulas

The fragment is parameterized by a set $\mathbb{P}$ of *predicates* defined using the following abstract syntax:

$$P(E, F, \overrightarrow{B}) \triangleq (E = F \wedge \mathit{emp}) \vee \big(E \neq \{F\} \cup \overrightarrow{B} \wedge \exists X_{\mathtt{t1}}. \Sigma(E, X_{\mathtt{t1}}, \overrightarrow{B}) * P(X_{\mathtt{t1}}, F, \overrightarrow{B})\big)$$

where Σ is an existentially-quantified formula, called *the matrix of* P, which has a special form defined in [1]. For this edition of the competition, SPEN is applied to formulas built using the fixed set of predicates defined in Figure 1.

The semantics used for the Separation Logic formulas is the strict semantics.

2 Principle

The solver is based on a method defined in [1] which reduces the checking of validity of $A \Rightarrow B$ to checking a number of simpler problems $A' \Rightarrow P(...)$ where A' is a sub-formula of A and $P(...)$ is a predicate atom of B.

[3] SPEN is a public project in GitHub.

singly linked lists:

$$\texttt{ls}(E,F) \triangleq lemp(E,F) \vee (E \neq F \wedge \exists X_{\texttt{t1}}.\, E \mapsto \{(f, X_{\texttt{t1}})\} * \texttt{ls}(X_{\texttt{t1}}, F))$$

lists of acyclic lists:

$$\texttt{nll}(E,F,B) \triangleq lemp(E,F) \vee (E \neq \{F,B\} \wedge \exists X_{\texttt{t1}}, Z.\, E \mapsto \{(s, X_{\texttt{t1}}), (h, Z)\} * \texttt{ls}(Z, B) * \texttt{nll}(X_{\texttt{t1}}, F, B))$$

lists of cyclic lists:

$$\texttt{nlcl}(E,F) \triangleq lemp(E,F) \vee (E \neq F \wedge \exists X_{\texttt{t1}}, Z.\, E \mapsto \{(s, X_{\texttt{t1}}), (h, Z)\} * \circlearrowleft^{1+} \texttt{ls}[Z] * \texttt{nlcl}(X_{\texttt{t1}}, F))$$

skip lists with three levels:

$$\texttt{skl}_3(E,F) \triangleq lemp(E,F) \vee (E \neq F \wedge \exists X_{\texttt{t1}}, Z_1, Z_2.\, E \mapsto \{(f_3, X_{\texttt{t1}}), (f_2, Z_2), (f_1, Z_1)\} * \texttt{skl}_1(Z_1, Z_2) * \texttt{skl}_2(Z_2, X_{\texttt{t1}}) * \texttt{skl}_3(X_{\texttt{t1}}, F))$$

$$\texttt{skl}_2(E,F) \triangleq lemp(E,F) \vee (E \neq F \wedge \exists X_{\texttt{t1}}, Z_1.\, E \mapsto \{(f_3, \texttt{NULL}), (f_2, X_{\texttt{t1}}), (f_1, Z_1)\} * \texttt{skl}_1(Z_1, X_{\texttt{t1}}) * \texttt{skl}_2(X_{\texttt{t1}}, F))$$

$$\texttt{skl}_1(E,F) \triangleq lemp(E,F) \vee (E \neq F \wedge \exists X_{\texttt{t1}}.\, E \mapsto \{(f_3, \texttt{NULL}), (f_2, \texttt{NULL}), (f_1, X_{\texttt{t1}})\} * \texttt{skl}_1(X_{\texttt{t1}}, F))$$

Fig. 1. Inductive definitions in SPEN($lemp(E,F) \triangleq E = F \wedge emp$)

The reduction is based on a boolean abstraction of the initial formulas and checking satisfiability of boolean queries on these abstraction using a SAT-solver (at the present time, the incremental version of MINISAT).

The validity of simpler entailments $A' \Rightarrow P(...)$ is checked by reduction to membership testing between a tree $T[A']$ built from A' and the language of a tree automaton $\mathcal{T}[P(...)]$ built from the recursive definition of P. The membership test relies on VATA [2], an efficient library for the manipulation of tree automata.

3 Installation and use

For installing SPEN from sources you need: Bison $\geq$ 2.7, Flex $\geq$ 2.4, Cmake $\geq$ 2.8.2, GCC$\geq$ 4.8.0, MINISAT (incremental version), VATA library, and SMTLIB2 parser of Alberto Griggio.

The input of SPENis a file in the SMTLIB2 format using a theory which is an extension of the theory proposed for this SL-SMTCOMP.

The output of SPEN is UNSAT when the entailment is valid, SAT (and a diagnosis) when the entailment is not valid, and UNKNOWN (and an explanation) when the set of predicates used is not supported.

References

1. Constantin Enea, Ondřej Lengál, Mihaela Sighireanu, and Tomáš Vojnar. Compositional entailment checking for a fragment of separation logic. Technical Report FIT-TR-2014-01, FIT BUT, 2014. `http://www.fit.vutbr.cz/~ilengal/pub/FIT-TR-2014-01.pdf`.
2. Ondřej Lengál, Jiří Šimáček, and Tomáš Vojnar. VATA: A library for efficient manipulation of non-deterministic tree automata. In *Proc. of TACAS'12*, volume 7214 of *LNCS*, pages 79–94. Springer, 2012.