SPEN

Constantin Enea\textsuperscript{1}, Ondřej Lengál\textsuperscript{2}, Mihaela Sighireanu\textsuperscript{1}, and Tomáš Vojnar\textsuperscript{2}

\textsuperscript{1} Univ. Paris Diderot, LIAFA CNRS UMR 7089
\textsuperscript{2} FIT, Brno University of Technology, IT4Innovations Centre of Excellence, Czech Republic

SPEN is an open source solver\textsuperscript{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:

$$x, y \in Vars \text{ program variables } \quad X, Y \in LVars \text{ logical variables } \quad E, F ::= x \mid X \quad f \in F \text{ fields } \quad \rho ::= (f, E) \mid \rho, \rho \quad P \in P \text{ predicates } \quad \vec{B} \in (Vars \cup LVars)^* \text{ vectors of variables }$$

$$\Pi ::= E = F \mid E \neq F \mid \Pi \land \Pi \quad \text{pure formulas } \quad \Sigma ::= emp \mid E \mapsto \{\rho\} \mid P(E, F, \vec{B}) \mid \Sigma \ast \Sigma \quad \text{spatial formulas } \quad A, B \triangleq \exists \neg X. \Pi \land \Sigma$$

The fragment is parameterized by a set $P$ of predicates defined using the following abstract syntax:

$$P(E, F, \vec{B}) \triangleq (E = F \land emp) \lor (E \neq \{F\} \cup \vec{B} \land \exists X_{t1.} \Sigma(E, X_{t1}, \vec{B}) \ast P(X_{t1}, F, \vec{B}))$$

where $\Sigma$ is an existentially-quantified formula, called the matrix of $P$, which has a special form defined in \cite{1}. For this edition of the competition, SPEN is applied to formulas built using the fixed set of predicates defined in Figure\textsuperscript{1}.

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

2 Principle

The solver is based on a method defined in \cite{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 $B$.

\textsuperscript{3} SPEN is a public project in GitHub.
singly linked lists:
\[1s(E, F) \triangleq \text{lemp}(E, F) \lor (E \neq F \land \exists X_{c1}. E \mapsto \{(f, X_{c1})\} * 1s(X_{c1}, F))\]

lists of acyclic lists:
\[nll(E, F, B) \triangleq \text{lemp}(E, F) \lor (E \neq \{F, B\} \land \exists X_{c1}, Z. E \mapsto \{(s, X_{c1}), (h, Z)\} * 1s(Z, B) * nll(X_{c1}, F, B))\]

lists of cyclic lists:
\[ncll(E, F) \triangleq \text{lemp}(E, F) \lor (E \neq \{F, B\} \land \exists X_{c1}, Z. E \mapsto \{(s, X_{c1}), (h, Z)\} * 1s(Z, B) * ncll(X_{c1}, F))\]

skip lists with three levels:
\[skl_3(E, F) \triangleq \text{lemp}(E, F) \lor (E \neq F \land \exists X_{c1}, Z_1, Z_2. E \mapsto \{(f_3, X_{c1}), (f_2, Z_2), (f_1, Z_1)\} * \text{skl}_1(Z_1, Z_2) * \text{skl}_2(Z_2, X_{c1}) * \text{skl}_3(X_{c1}, F))\]
\[skl_2(E, F) \triangleq \text{lemp}(E, F) \lor (E \neq F \land \exists X_{c1}, Z_1. E \mapsto \{(f_3, \text{NULL}), (f_2, X_{c1}), (f_1, Z_1)\} * \text{skl}_1(Z_1, X_{c1}) * \text{skl}_2(X_{c1}, F))\]
\[skl_1(E, F) \triangleq \text{lemp}(E, F) \lor (E \neq F \land \exists X_{c1}. E \mapsto \{(f_3, \text{NULL}), (f_2, \text{NULL}), (f_1, X_{c1})\} * \text{skl}_1(X_{c1}, F))\]

Fig. 1. Inductive definitions in $\text{SPEN}($lemp$(E, F) \triangleq E = F \land \text{emp}$)

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 $T[P(...)]$ 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 SPEN is a file in the SMTLIB2 format using a theory which is an extension of the theory proposed for this SL-SMTCOMP.

The output of $\text{SPEN}$ 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
