Automating Separation Logic using SMT

Ruzica Piskac    Thomas Wies    Damien Zufferey

MPI-SWS    NYU    IST Austria

CAV, July 17 2013, Saint Petersburg
procedure concat(a: Node, b: Node) returns (res: Node)
  requires lseg(a, null) * lseg(b, null);
  ensures lseg(res, null);
{
  if (a == null)
    return b;

  Node curr := a;

  while (curr.next != null)
    invariant curr != null * lseg(a, curr) * lseg(curr, null);
    curr := curr.next;

  curr.next := b;

  return a;
}

pre / postconditions
loop invariants
procedure concat(a: Node, b: Node) returns (res: Node)
    requires lseg(a, null) * lseg(b, null);
    ensures lseg(res, null);
{
    if (a == null)
        return b;
    Node curr := a;

    while (curr.next != null)
        invariant curr != null * lseg(a, curr) * lseg(curr, null);
        curr := curr.next;

    curr.next := b;
    return a;
}
procedure concat(a: Node, b: Node) returns (res: Node)
  requires lseg(a, null) * lseg(b, null);
  ensures lseg(res, null);
{
  if (a == null)
    return b;
  Node curr := a;
  while (curr.next != null)
    invariant curr != null * lseg(a, curr) * lseg(curr, null);
    curr := curr.next;
  curr.next := b;
  return a;
}
procedure concat(a: Node, b: Node) returns (res: Node)
    requires lsleg(a, null, x) * uslseg(b, null, x);
    ensures slseg(res, null);
{
    if (a == null)
        return b;
    Node curr := a;
    while (curr.next != null)
        invariant curr != null;
        invariant ls1seg(a, curr, curr.data) * ls1seg(curr, null, x);
        curr := curr.next;
    curr.next := b;
    return a;
}
Our work

- Reduce a decidable fragment of SL to a decidable FO theory.
- Combining SL with other theories.
- Satisfiability, entailment, frame inference, and abduction problems for SL using SMT solvers.
- Implemented in the GRASSShopper tool.
Decidable SL fragment: $\text{SLL}\mathbb{B}$

SLL (separation logic formulas for linked lists) introduced in [Berdine et al., 2004].

SLL

$$\Sigma ::= x = y \mid x \neq y \mid x \mapsto y \mid \text{ls}(x, y) \mid \Sigma \ast \Sigma$$

With extend SLL to $\text{SLL}\mathbb{B}$ by adding boolean connective on top:

$$H ::= \Sigma \mid \neg H \mid H \land H$$
Semantics of SLLB (1)

\[ \Sigma ::= x = y \mid x \neq y \mid x \leftrightarrow y \mid \text{ls}(x, y) \mid H_1 \ast H_2 \]

- footprint = \( \emptyset \)
- footprint = \( \emptyset \)
- footprint = \{ x \}
Semantics of \(\text{SLLB} (2)\)

\[
\Sigma ::= x = y \mid x \neq y \mid x \leftrightarrow y \mid \text{ls}(x, y) \mid H_1 \ast H_2
\]

important: \(\exists U_1, U_2\)
Semantics of $\text{SLLB} (3)$

$$\Sigma ::= x = y \mid x \neq y \mid x \leftrightarrow y \mid \text{ls}(x, y) \mid H_1 \ast H_2$$

\[
\begin{align*}
x &\rightarrow w \rightarrow \cdots \rightarrow v \rightarrow y \\
\text{footprint} &\quad \emptyset \quad \text{ls}^0(y, y) \\
&\quad \text{ls}^1(v, y) \\
&\quad \cdots \\
&\quad \text{ls}^{n-1}(w, y) \\
&\quad \text{ls}^n(x, y)
\end{align*}
\]
Translate SLLB to a decidable FO theory.

Requirements:
- easy automation with SMT solvers
- well-behaved under theory combination
- no increase in complexity

GRASS: combination of two theories
- structure: functional graph reachability ($\mathcal{T}_G$) to encode the shape of the heap (pointers)
- footprint: stratified sets ($\mathcal{T}_S$) to encode the part of the heap used by a formula
GRASS: graph reachability and stratified sets

**Graph reachability**

\[ T ::= x \mid h(T) \]

\[ A ::= T = T \mid T \xrightarrow{h \setminus T} T \]

\[ R ::= A \mid \neg R \mid R \land R \mid R \lor R \]

**Stratified sets**

\[ S ::= X \mid \emptyset \mid S \setminus S \mid S \cap S \mid S \cup S \mid \{x. R\} \quad x \text{ not below } h \text{ in } R \]

\[ B ::= S = S \mid T \in S \]

**Top level boolean combination**

\[ F ::= A \mid B \mid \neg F \mid F \land F \mid F \lor F \]
\( T_G \): theory of function graphs

\( t_1 \xrightarrow{h \setminus t_3} t_2 \) is true if there exists a path in the graph of \( h \) that connects \( t_1 \) and \( t_2 \) without going through \( t_3 \).

\[
\begin{align*}
&v \\
&w \xrightarrow{h} w \text{ (reflexivity)} & \neg v \xrightarrow{h} w \text{ (no path)} \\
&x \xrightarrow{h} y \text{ (induced by } h) & \neg x \xrightarrow{h \setminus y} z \text{ (} y \text{ is before } z) \\
&B_{\text{tw}}(w, z) = \{y.w \xrightarrow{h \setminus z} y \land z \neq y\} = \{w, x, y\}
\end{align*}
\]
Usual way of translating SL to FO:

- **structure**: $T_G$ to encode the shape of the heap (pointers)
- **footprint**: $T_S$ to encode the part of the heap used by a formula

Negation (entailment check, frame) $\Rightarrow$ more complicated

- **structure**: uses $T_G$ and $T_S$ to encode the shape of the heap (pointers) and disjointness
- **set definition**: uses $T_S$ for keep track of the sets that will make the footprint
SLLB → GRASS: interesting cases

\[\text{Tr}_X(H) = \text{let } (F, G) = \text{tr}_X(H) \text{ in } F \land G\]

- \(F\) is the structure
- \(G\) is the set definitions.

\[\text{tr}_X(\text{ls}(x, y)) = (x \xrightarrow{h} y, \ X = Btwn(x, y))\]

\[\text{tr}_X(\Sigma_1 \ast \Sigma_2) = \text{let } Y_1, Y_2 \in \mathcal{X} \text{ fresh}
\]

\[\text{and } (F_1, G_1) = \text{tr}_{Y_1}(\Sigma_1)\]

\[\text{and } (F_2, G_2) = \text{tr}_{Y_2}(\Sigma_2)\]

\[\text{in } (F_1 \land F_2 \land Y_1 \cap Y_2 = \emptyset, \ X = Y_1 \cup Y_2 \land G_1 \land G_2)\]

\[\text{tr}_X(\neg H) = \text{let } (F, G) = \text{tr}_X(H) \text{ in } (\neg F, G)\]
Example: without negation

a non-empty acyclic list segment from $x$ to $z$

\[ x \neq z \land h(x) = y \land y \xrightarrow{h} z \land Y_2 \cap Y_3 = \emptyset \land Y_4 \cap Y_5 = \emptyset \land X = Y_1 \land Y_1 = Y_2 \cup Y_3 \land Y_2 = \emptyset \land Y_3 = Y_4 \cup Y_5 \land Y_4 = \{x\} \land Y_5 = Btwn(y, z) \]
Example: with negation

a non-empty acyclic list segment from $x$ to $z$

$$\neg((x \neq z \land x \rightarrow y \land \text{ls}(y, z)))$$

with negation

structure (negated)

$$x = z \lor h(x) \neq y \lor \neg y \rightarrow z \lor Y_2 \cap Y_3 \neq \emptyset \lor Y_4 \cap Y_5 \neq \emptyset \lor X \neq Y_1$$

set definitions (unchanged)

$$Y_1 = Y_2 \cup Y_3 \land Y_2 = \emptyset \land Y_3 = Y_4 \cup Y_5 \land Y_4 = \{x\} \land Y_5 = \text{Btw}n(y, z)$$
Why is that correct?

Translation: \( Tr_X(H) = \text{let} (F, G) = tr_X(H) \text{ in } F \land G \)

the auxiliary variables \( Y_i \) (in \( G \)) are existentially quantified

below negation, the existential quantifiers should become universal

the \( Y_i \) are defined as finite unions of set comprehensions

\( \rightarrow \text{satisfiable in any given heap interpretation } A \)

Due to the precise semantics of SLL

\( \rightarrow \text{exists exactly one assignment of the } Y_i \text{ that makes } G \text{ true in } A \)

\[ \exists Y_1, \ldots, Y_n. F \land G \quad \text{and} \]

\[ \forall Y_1, \ldots, Y_n. G \Rightarrow F \quad \text{are equivalent.} \]
Where are we now?

With the SLLB to GRASS translation we can

- Check for satisfiability
- Check entailment (reduces to satisfiability of $H_1 \land \neg H_2$)

We also have a translation from GRASS to SLLB:

- compute $F$ in $A \models_{\text{SL}} B \ast F$ (frame)
- compute $F$ in $A \ast F \models_{\text{SL}} B$ (antiframe)

The details are in the paper.
Combination with other theories and extensions

- The theories $\mathcal{T}_G$ and $\mathcal{T}_S$ are stably infinite. (Nelson-Oppen)
- Data: we can add data with constraints (see paper for details).
- More pointers: we can extend the signature with fields and use $\bullet \rightarrow \bullet$ with different fields (array theory).
- More complex data structures, e.g. doubly linked lists, ...
### Experimental results

Implementation: **GRASShopper** available at https://cs.nyu.edu/wies/software/grasshopper/

<table>
<thead>
<tr>
<th>program</th>
<th>sl</th>
<th>dl</th>
<th>rec sl</th>
<th>sls</th>
<th>program</th>
<th>sl</th>
<th>dl</th>
<th>rec sl</th>
<th>sls</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#</td>
<td>t</td>
<td>#</td>
<td>t</td>
<td></td>
<td>#</td>
<td>t</td>
<td>#</td>
<td>t</td>
</tr>
<tr>
<td>concat</td>
<td>4</td>
<td>0.1</td>
<td>5</td>
<td>1.3</td>
<td>6</td>
<td>0.6</td>
<td>5</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>copy</td>
<td>4</td>
<td>0.2</td>
<td>4</td>
<td>3.9</td>
<td>6</td>
<td>0.8</td>
<td>7</td>
<td>3.5</td>
<td></td>
</tr>
<tr>
<td>filter</td>
<td>7</td>
<td>0.6</td>
<td>5</td>
<td>1.1</td>
<td>8</td>
<td>0.4</td>
<td>5</td>
<td>1.1</td>
<td></td>
</tr>
<tr>
<td>free</td>
<td>5</td>
<td>0.1</td>
<td>5</td>
<td>0.3</td>
<td>4</td>
<td>0.1</td>
<td>5</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>insertion sort</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>merge sort</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>6.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td>6</td>
<td>0.2</td>
<td>5</td>
<td>1.5</td>
<td>5</td>
<td>0.2</td>
<td>6</td>
<td>0.2</td>
<td>6</td>
</tr>
<tr>
<td>reverse</td>
<td>4</td>
<td>0.1</td>
<td>4</td>
<td>0.5</td>
<td>6</td>
<td>0.2</td>
<td>4</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>remove</td>
<td>8</td>
<td>0.2</td>
<td>8</td>
<td>0.8</td>
<td>7</td>
<td>0.2</td>
<td>7</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>traverse</td>
<td>4</td>
<td>0.1</td>
<td>5</td>
<td>0.3</td>
<td>3</td>
<td>0.1</td>
<td>4</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>double all</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pairwise sum</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>2.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**sl** singly-linked list (loop or recursion)  
**dl** doubly-linked list  
**sls** sorted lists  

<table>
<thead>
<tr>
<th># number of VCs</th>
<th>t total time in s.</th>
</tr>
</thead>
</table>
Conclusion

- Reduce a decidable fragment of SL to a decidable FO theory.
- Combining SL with other theories.
- Satisfiability, entailment, frame inference, and abduction problems for SL using SMT solvers.
- Implemented in the GRASShopper tool.
Most prominent decidable fragments of SL: linked lists [Berdine et al., 2004], decidable in polynomial time [Cook et al., 2011] (graph-based).

SL → FO: [Calcagno and Hague, 2005] (no inductive predicate) and [Bobot and Filliâtre, 2012] (not a decidable fragment).

Alternatives to SL: (implicit) dynamic frames [Kassios, 2011] and region logic [Banerjee et al., 2008, Rosenberg et al., 2012].

The connection between SL and implicit dynamic frames has been studied in [Parkinson and Summers, 2012].

References II

Regional Logic for Local Reasoning about Global Invariants.
In ECOOP vol. 5142, of LNCS pp. 387–411.

A Decidable Fragment of Separation Logic.
In FSTTCS.

Separation Predicates: a Taste of Separation Logic in First-Order Logic.
In ICFEM.

From separation logic to first-order logic.
In FoSSaCs’05 pp. 395–409, Springer.

Tractable Reasoning in a Fragment of Separation Logic.
In CONCUR. Springer.

The dynamic frames theory.

Back to the future: revisiting precise program verification using SMT solvers.