Automating Separation Logic using SMT

Ruzica Piskac   Thomas Wies   *Damien Zufferey*

MPI-SWS   NYU   IST Austria

CAV, July 17 2013, Saint Petersburg

Motivation: Program with SL Specification

```
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

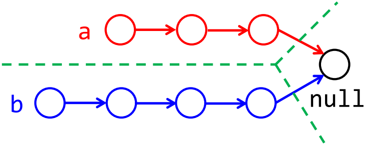
Separating Conjunction and Inductive Predicates

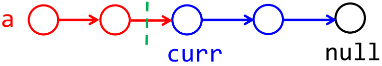
```
procedure concat(a: Node, b: Node) returns (res: Node)
  requires lseg(a, null) * lseg(b, null);
  ensures lseg(res, null);
{
  if (a == null)
    return b;

  Node curr := a;

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

  curr.next := b;
  return a;
}
```





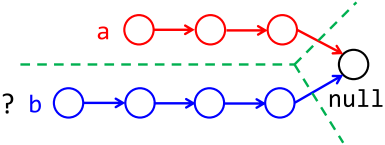
Frame Inference

```
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;
}
```



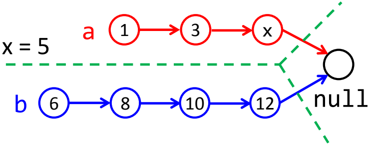
Adding Data

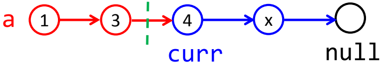
```
procedure concat(a: Node, b: Node) returns (res: Node)
  requires lsleg(a, null, x) * uslseg(b, null, x);
  ensures slseg(res, null);
{
  if (a == null)
    return b;

  Node curr := a;

  while (curr.next != null)
    invariant curr != null;
    invariant lslseg(a, curr, curr.data) * lslseg(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 GRASShopper tool.

Decidable SL fragment: SLL$\mathbb{B}$

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

SLL

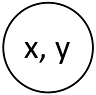
$$\Sigma ::= x = y \mid x \neq y \mid x \mapsto y \mid \mathsf{ls}(x, y) \mid \Sigma * \Sigma$$

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

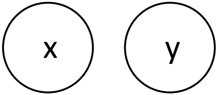
$$H ::= \Sigma \mid \neg H \mid H \wedge H$$

Semantics of SLL$\mathbb{B}$ (1)

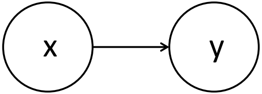
$$\Sigma ::= x = y \mid x \neq y \mid x \mapsto y \mid \mathsf{ls}(x, y) \mid H_1 * H_2$$



footprint = ∅



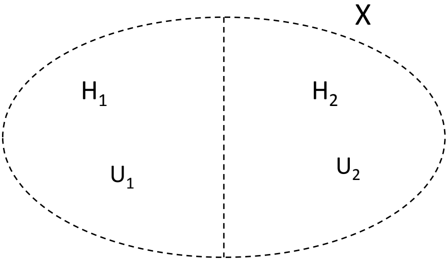
footprint = ∅



footprint = { x }

Semantics of SLL𝔹 (2)

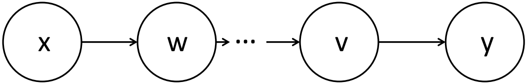
$$\Sigma ::= x = y \mid x \neq y \mid x \mapsto y \mid \mathsf{ls}(x, y) \mid H_1 * H_2$$



important: $\exists U_1, U_2$

Semantics of SLL$\mathbb{B}$ (3)

$$\Sigma ::= x = y \mid x \neq y \mid x \mapsto y \mid \mathsf{ls}(x, y) \mid H_1 * H_2$$



	footprint
$\emptyset$	$\mathsf{ls}^0(y,y)$
	$\mathsf{ls}^1(v,y)$
...	...
	$\mathsf{ls}^{n-1}(w,y)$
	$\mathsf{ls}^n(x,y)$

SLL$\mathbb{B}$ → GRASS

Translate SLL$\mathbb{B}$ 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

$$
\begin{array}{lcl}
T & ::= & x \mid h(T) \\
A & ::= & T = T \mid T \xrightarrow{h \setminus T} T \\
R & ::= & A \mid \neg R \mid R \wedge R \mid R \vee R
\end{array}
$$

stratified sets

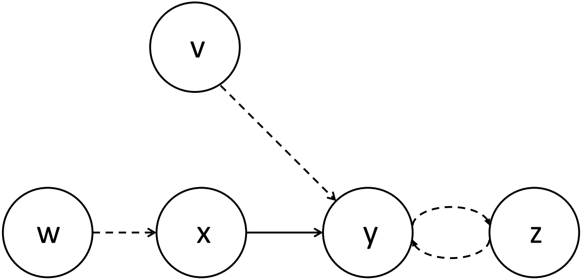
$$
\begin{array}{lcl}
S & ::= & X \mid \emptyset \mid S \setminus S \mid S \cap S \mid S \cup S \mid \{x.\, R\} \quad x \text{ not below } h \text{ in } R \\
B & ::= & S = S \mid T \in S
\end{array}
$$

top level boolean combination

$$
F ::= A \mid B \mid \neg F \mid F \wedge F \mid F \vee F
$$

$\mathcal{T}_G$: 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.



$w \xrightarrow{h} w$ (reflexivity)

$\neg v \xrightarrow{h} w$ (no path)

$x \xrightarrow{h} y$ (induced by h)

$\neg x \xrightarrow{h \setminus y} z$ (y is before z)

$$Btwn(w, z) = \{y . w \xrightarrow{h \setminus z} y \wedge z \neq y\} = \{w, x, y\}$$

SLL$\mathbb{B}$ $\rightarrow$ GRASS (1)

Usual way of translating SL to FO:

- structure: $\mathcal{T}_{\mathsf{G}}$ to encode the shape of the heap (pointers)
- footprint: $\mathcal{T}_{\mathsf{S}}$ to encode the part of the heap used by a formula

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

- structure: uses $\mathcal{T}_{\mathsf{G}}$ and $\mathcal{T}_{\mathsf{S}}$ to encode the shape of the heap (pointers) and disjointness
- set definition: uses $\mathcal{T}_{\mathsf{S}}$ for keep track of the sets that will make the footprint

SLL$\mathbb{B}$ $\rightarrow$ GRASS: interesting cases

$$Tr_X(H) = \text{let } (F, G) = tr_X(H) \text{ in } F \wedge G$$

F is the structure

G is the set definitions.

$$tr_X(\mathsf{ls}(x, y)) = (x \xrightarrow{h} y,\ X = Btwn(x, y))$$

$$\begin{aligned} tr_X(\Sigma_1 * \Sigma_2) = \text{let } & Y_1, Y_2 \in \mathcal{X} \text{ fresh} \\ & \text{and } (F_1, G_1) = tr_{Y_1}(\Sigma_1) \\ & \text{and } (F_2, G_2) = tr_{Y_2}(\Sigma_2) \\ & \text{in } (F_1 \wedge F_2 \wedge Y_1 \cap Y_2 = \emptyset,\ X = Y_1 \cup Y_2 \wedge G_1 \wedge G_2) \end{aligned}$$

$$tr_X(\neg H) = \text{let } (F, G) = tr_X(H) \text{ in } (\neg F,\ G)$$

Example: without negation

a non-empty acyclic list segment from x to z

$$x \neq z * x \mapsto y * \mathrm{ls}(y, z)$$

translate to

$$x \neq z \land h(x) = y \land y \xrightarrow{h} z \land Y_2 \cap Y_3 = \emptyset \land Y_4 \cap Y_5 = \emptyset \land X = Y_1 \land Y_1 = Y_2 \cup Y_3 \land Y_2 = \emptyset \land Y_3 = Y_4 \cup Y_5 \land Y_4 = \{x\} \land Y_5 = Btwn(y, z)$$

Example: with negation

a non-empty acyclic list segment from x to z

$$\neg(x \neq z * x \mapsto y * \mathsf{ls}(y, z))$$

with negation

structure (negated)

$$x = z \vee h(x) \neq y \vee \neg y \xrightarrow{h} z \vee Y_2 \cap Y_3 \neq \emptyset \vee Y_4 \cap Y_5 \neq \emptyset \vee X \neq Y_1$$

set definitions (unchanged)

$$Y_1 = Y_2 \cup Y_3 \wedge Y_2 = \emptyset \wedge Y_3 = Y_4 \cup Y_5 \wedge Y_4 = \{x\} \wedge Y_5 = Btwn(y, z)$$

Why is that correct ?

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

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
→ satisfiable in any given heap interpretation $\mathcal{A}$

Due to the precise semantics of SLL$\mathbb{B}$
→ exists exactly one assignment of the Y_i that makes G true in $\mathcal{A}$

$\exists Y_1, \ldots, Y_n.\, F \wedge G$ and
$\forall Y_1, \ldots, Y_n.\, G \Rightarrow F$ are equivalent.

Where are we now ?

With the SLL$\mathbb{B}$ to GRASS translation we can

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

We also have a translation from GRASS to SLL$\mathbb{B}$:

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

The details are in the paper.

Combination with other theories and extensions

- The theories $\mathcal{T}_\mathsf{G}$ and $\mathcal{T}_\mathsf{S}$ 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 $\bullet \xrightarrow{\bullet \setminus \bullet} \bullet$ 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/`

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

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

- # number of VCs
- t total time in s.

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.

Related work

- Most prominent decidable fragments of SL: linked lists [Berdine et al., 2004], decidable in polynomial time [Cook et al., 2011] (graph-based).
- SL $\rightarrow$ 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].
- SMT-based decision procedures for theories of reachability in graphs [Lahiri and Qadeer, 2008, Wies et al., 2011, Totla and Wies, 2013] , decision procedures for theories of stratified sets [Zarba, 2004].

Banerjee, A., Naumann, D. A. and Rosenberg, S. (2008).
Regional Logic for Local Reasoning about Global Invariants.
In ECOOP vol. 5142, of LNCS pp. 387–411,.

Berdine, J., Calcagno, C. and O'Hearn, P. (2004).
A Decidable Fragment of Separation Logic.
In FSTTCS.

Bobot, F. and Filliâtre, J.-C. (2012).
Separation Predicates: a Taste of Separation Logic in First-Order Logic.
In ICFEM.

Calcagno, C. and Hague, M. (2005).
From separation logic to first-order logic.
In FoSSaCs'05 pp. 395–409, Springer.

Cook, B., Haase, C., Ouaknine, J., Parkinson, M. and Worrell, J. (2011).
Tractable Reasoning in a Fragment of Separation Logic.
In CONCUR. Springer.

Kassios, I. T. (2011).
The dynamic frames theory.
Formal Asp. Comput. *23*, 267–288.

Lahiri, S. K. and Qadeer, S. (2008).
Back to the future: revisiting precise program verification using SMT solvers.