Automating Separation Logic using SMT

Ruzica Piskac Thomas Wies *Damien Zufferey*

Yale University NYU MIT CSAIL

PL lunch, October 29 2013

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

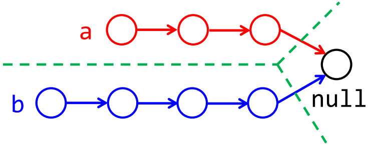
Separating Conjunction and Inductive Predicates

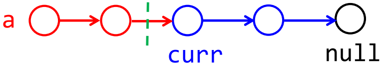
```
procedure concat(a: Node, b: Node) returns (res: Node)
  requires lseg(a, null) * lseg(b, null);
  ensures lseg(res, null);
{
  if (a == null)
    return b;

  Node curr := a;

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

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





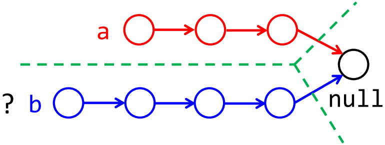
Frame Inference

```
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;
}
```



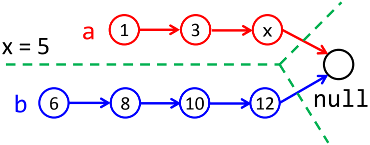
Adding Data

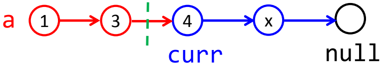
```
procedure concat(a: Node, b: Node) returns (res: Node)
  requires lsleg(a, null, x) * uslseg(b, null, x);
  ensures slseg(res, null);
{
  if (a == null)
    return b;

  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: $\mathsf{SLL}\mathbb{B}$

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

SLL

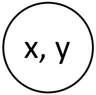
$$\Sigma ::= x = y \mid x \neq y \mid x \mapsto y \mid \mathsf{ls}(x, y) \mid \Sigma * \Sigma$$

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

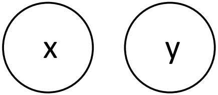
$$H ::= \Sigma \mid \neg H \mid H \wedge H$$

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

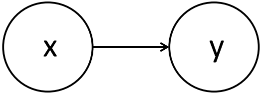
$$\Sigma ::= x = y \mid x \neq y \mid x \mapsto y \mid \mathsf{ls}(x, y) \mid H_1 * H_2$$



footprint = ∅



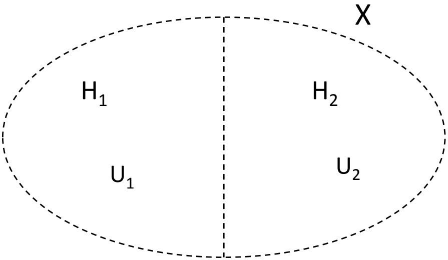
footprint = ∅



footprint = { x }

Semantics of SLL$\mathbb{B}$ (2)

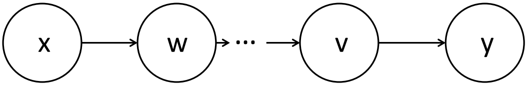
$$\Sigma ::= x = y \mid x \neq y \mid x \mapsto y \mid \mathsf{ls}(x, y) \mid H_1 * H_2$$



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

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

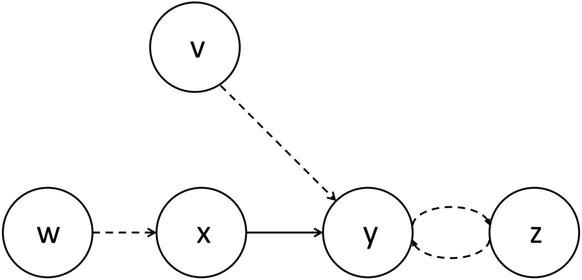
- structure: *functional graph reachability* ($\mathcal{T}_\mathsf{G}$)
  to encode the shape of the heap (pointers)
- footprint: *stratified sets* ($\mathcal{T}_\mathsf{S}$)
  to encode the part of the heap used by a formula

GRASS: graph reachability and stratified sets

$$
\begin{array}{rcll}
 & & \text{graph reachability} & \\
T & ::= & x \mid h(T) & \\
A & ::= & T = T \mid T \xrightarrow{h\setminus T} T & \\
R & ::= & A \mid \neg R \mid R \wedge R \mid R \vee R & \\
 & & & \\
 & & \text{stratified sets} & \\
S & ::= & X \mid \emptyset \mid S \setminus S \mid S \cap S \mid S \cup S \mid \{x.\, R\} & x \text{ not below } h \text{ in } R \\
B & ::= & S = S \mid T \in S & \\
 & & & \\
 & & \text{top level boolean combination} & \\
F & ::= & A \mid B \mid \neg F \mid F \wedge F \mid F \vee F &
\end{array}
$$

$\mathcal{T}_G$: theory of function graphs

$t_1 \xrightarrow{h \backslash t_3} t_2$ 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)

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

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

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

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

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

Usual way of translating SL to FO:

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

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

- structure: uses $\mathcal{T}_G$ and $\mathcal{T}_S$ to encode the shape of the heap (pointers) and disjointness
- set definition: uses $\mathcal{T}_S$ 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.

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

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

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

Example: without negation

a non-empty acyclic list segment from x to z

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

translate to

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

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)

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

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)

Done by model ennumeration $\rightarrow$ not practical
Therefore, we will see another way of doing compositional reasoning.

Implicit frame inference

Idea: let the solver do the frame inference:

$$\forall x.\, x \in Frame \Rightarrow h'(x) = h(x)$$

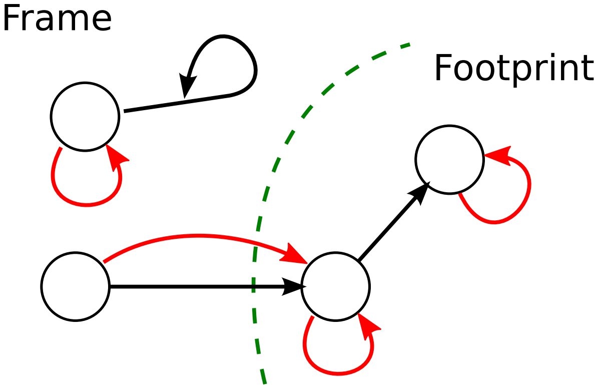
Not that easy: our decision procedure works on partial model.

We need to tell the solver how to update $x \xrightarrow{h} y$:

All *paths* in the frame are preserved.

We add entry points to interface the frame and the footprint.

$ep(FP, x)$ in picture



$ep(FP, x)$

Updating the paths (roughly):

$$\forall x, y, z \in Frame.\, x \xrightarrow{h \setminus ep(FP,x)} y \Rightarrow (Btwn(x, z, y) \Leftrightarrow Btwn'(x, z, y))$$
$$\forall x, y, z.\, x \in Frame \wedge x = ep(FP, x) \Rightarrow (Btwn(x, y, z) \Leftrightarrow Btwn'(x, y, z))$$

Axioms defining the entry point function:

$$\begin{aligned}
&\forall x.\, Btwn(x, ep(FP, x), ep(FP, x)) \\
&\forall x.\, ep(FP, x) \in FP \vee ep(FP, x) = x \\
&\forall x, y.\, Btwn(x, y, y) \wedge y \in FP \Rightarrow \\
&\qquad\qquad ep(FP, x) \in FP \wedge Btwn(x, ep(FP, x), y)
\end{aligned}$$

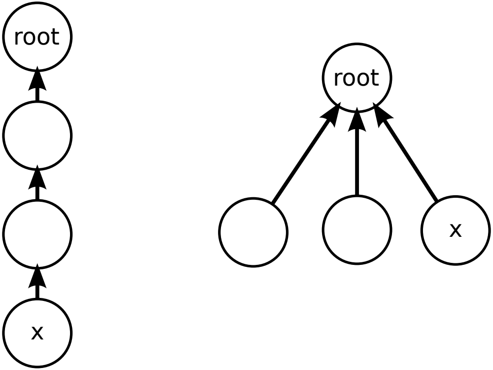
$ep(FP, \_)$ is idempotent (still decidable).

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, ...

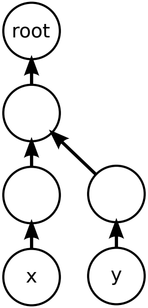
Mixed specifications: union-find

procedure find(x: Node, **ghost** root_x: Node, **implicit ghost** X: set<Node>)
  returns (res: Node)
  requires lseg_set(x, root_x, X) $*$ root_x.next $\mapsto$ null;
  ensures res $=$ root_x $*$ acc(X) $*$ ($\forall$ z $\in$ X :: z.next $=$ res) $*$ res.next $\mapsto$ null;



Mixed specifications: union-find

procedure union(x: Node, y: Node, **ghost** root_x: Node, **ghost** root_y: Node,
implicit ghost X: set<Node>, **implicit ghost** Y: set<Node>)
requires lseg_set(x, root_x, X) + lseg_set(y, root_y, Y);
requires root_x.next $\mapsto$ null + root_y.next $\mapsto$ null;
ensures (acc(X) + acc(Y)) $*$ (root_y.next $\mapsto$ null + acc(root_x));
ensures ($\forall$ z $\in$ X :: z.next = root_x) $*$ ($\forall$ z $\in$ Y :: z.next = root_y);
ensures root_x = root_y $\vee$ root_x.next = root_y;



Experimental results

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

Benchmarks	# VCs	time in s
SLL (loop)	56	1.9
SLL (rec.)	70	3.1
sorted SLL	55	6.6
DLL	59	11
sorting algorithms	98	15
union-find	8	4.8
SLL.filter (deref. null pointer)	7	0.4
DLL.insert (missing update)	8	3.1
quicksort (underspec. split)	12	0.9
union-find (bug in postcond.)	4	12.8

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

- 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 reachability in graphs [Lahiri and Qadeer, 2008, Wies et al., 2011, Totla and Wies, 2013] , decision procedures for theories of stratified sets [Zarba, 2004].
- Entry points for modular reasoning [Itzhaky et al., 2014]

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.

Itzhaky, S., Lahav, O., Banerjee, A., Immerman, N., Nanevski, A. and Sagiv, M. (2014).
Modular Reasoning on Unique Heap Paths via Effectively Propositional Formulas.
In POPL.

Kassios, I. T. (2011).
The dynamic frames theory.