The Power of Compositional Inductive Definitions in Separation Logic

Mihaela Sighireanu

joint work with

Constantin Enea Zhillin Wu

PPS Days, November 9th, 2018

Verification is a Sinuous Way

Verification is a Sinuous Way

Summary

Separation Logic (SL) is a program logic specifying heap & stack properties.

Unbounded, heap-stored data structures are specified using inductively defined predicates (ID).

This work studies proof systems for checking validity of entailments encoding verification conditions (VC) in SL+ID.

Gentzen's cut elimination theorem does not hold for inductive theories; lemmas are required to complete proofs.

We identify a class of ID and prove lemmas allowing to complete proofs of iterative programs.

Summary

Separation Logic (SL) is a program logic
specifying heap & stack properties.

Unbounded, heap-stored data structures are specified using
inductively defined predicates (ID).

This work studies proof systems for checking validity of entailments
encoding verification conditions (VC) in SL+ID.

Gentzen's cut elimination theorem does not hold for inductive theories;
lemmas are required to complete proofs.

We identify a class of ID and prove lemmas allowing to complete proofs of
iterative programs.

Summary

Separation Logic (SL) is a program logic specifying heap & stack properties.

Unbounded, heap-stored data structures are specified using inductively defined predicates (ID).

This work studies proof systems for checking validity of entailments encoding verification conditions (VC) in SL+ID.

Gentzen's cut elimination theorem does not hold for inductive theories; lemmas are required to complete proofs.

We identify a class of ID and prove lemmas allowing to complete proofs of iterative programs.

Reasoning about heap manipulating programs

Heap manipulating programs: access memory using location variables.

Properties:

- *The program accesses only allocated locations*, i.e., $\neq$ `null`.
- *The location y is reachable from x using* `next` *offset* (i.e., field).
- *The program maintains a skip list.*

Specification logics:

- FO with reachability over labeled graphs

  [Yorsh *et al*'08, Drăgoi *et al*'09, Sagiv *et al*'14, . . .]
- Separation Logic with inductively defined predicates

  [Reynolds, O'Hearn *et al*'01]

Reasoning in Separation Logic

- scalability by frame rule due to separating conjunction $*$

$$\frac{\{P\}\ stmt\ \{Q\}}{\{P * R\}\ stmt\ \{Q * R\}}\ locs(R) \cap locs(stmt) = \emptyset$$

- entailment undecidable for records of 2 pointer fields
  $\longrightarrow$ search decidable fragments or sound procedures
- some fragments successfully used in software analyzers
  $\longrightarrow$CELIA, DRYAD, GRASShopper, SLEEK/HIP, Infer, Predator, Smallfoot, ...
- Gödel Prize in 2016

Separation Logic

Symbolic heaps fragment of SL

$$\varphi ::= \exists x_1, ..., x_n \,.\, \Sigma \wedge \Pi$$

- spatial part: $\Sigma ::= emp \mid x \overset{t}{\mapsto} \{f_i : x_i\}_i \mid \Sigma_1 * \Sigma_2 \mid P(x, \vec{y})$
- pure part: $\Pi ::= x_1 = x_2 \mid x_1 \neq x_2 \mid \Pi_1 \wedge \Pi_2$
- predicates: by sets of rules of the form

$$P(x, \vec{y}) \leftarrow \exists z_1, ..., z_m \,.\, \Pi \wedge \Sigma$$

Separation Logic

Symbolic heaps fragment of SL with data theories

$$\varphi ::= \exists x_1, ..., x_n \,.\, \Sigma \wedge \Pi$$

- spatial part: $\Sigma ::= emp \mid x \overset{t}{\mapsto} \{f_i : x_i\}_i \mid \Sigma_1 * \Sigma_2 \mid P(x, \vec{y})$
- pure part: $\Pi ::= x_1 = x_2 \mid x_1 \neq x_2 \mid \Pi_1 \wedge \Pi_2 \mid \Delta$
- predicates: by sets of rules of the form

$$P(x, \vec{y}) \leftarrow \exists z_1, ..., z_m \,.\, \Pi \wedge \Sigma$$

- data part: $\Delta ::= \Delta_{IA} \mid \Delta_{SET} \mid \Delta_{BAG} \mid \ldots$

Semantics

$$(S, H) \models \varphi$$

where $Locs \subset Vals$ and

- stack $S : Vars \cup LVars \to Vals$, ie, interpretation of program and logic variables
- heap $H : Locs \times Flds \rightharpoonup Vals$, ie, allocated memory,
  where $ldom(H) = fst(dom(H))$ are allocated locations in H, otherwise dangling

$(S, H) \models emp$ iff $dom(H) = \emptyset$

$(S, H) \models x \overset{t}{\mapsto} \{\rho\}$ iff $dom(H) = \{(S(x), f_i) \mid (f_i, x_i) \in \{\rho\}\}$ and
for every pair $(f_i, x_i) \in \{\rho\}$, $H(S(x), f_i) = S(x_i)$

$(S, H) \models \Sigma_1 * \Sigma_2$ iff $\exists H_1, H_2$ s.t. $ldom(H) = ldom(H_1) \uplus ldom(H_2)$,
$(S, H_1) \models \Sigma_1$, and $(S, H_2) \models \Sigma_2$

$(S, H) \models P(x, \vec{y})$ iff $\exists$ a rule $(P(X, \vec{Y}) \leftarrow \exists \vec{Z} : \Pi \wedge \Sigma) \in \mathcal{P}$ s.t.
$(S, H) \models \exists \vec{Z} : (\Pi \wedge \Sigma)[x/X, \vec{y}/\vec{Y}]$ and
$ldom(H) \cap \{S(y) \mid y \in \vec{y}\} = \emptyset$, i.e., $\vec{y}$ are dangling

Semantics

$$(S, H) \models \varphi$$

where $Locs \subset Vals$ and

- stack $S : Vars \cup LVars \to Vals$, ie, interpretation of program and logic variables
- heap $H : Locs \times Flds \rightharpoonup Vals$, ie, allocated memory,
  where $ldom(H) = fst(dom(H))$ are allocated locations in H, otherwise dangling

$(S, H) \models emp$ iff $dom(H) = \emptyset$

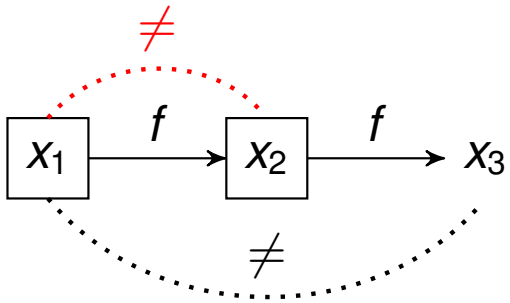
$(S, H) \models x \overset{t}{\mapsto} \{\rho\}$ iff $dom(H) = \{(S(x), f_i) \mid (f_i, x_i) \in \{\rho\}\}$ and
for every pair $(f_i, x_i) \in \{\rho\}$, $H(S(x), f_i) = S(x_i)$

$(S, H) \models \Sigma_1 * \Sigma_2$ iff $\exists H_1, H_2$ s.t. $ldom(H) = ldom(H_1) \uplus ldom(H_2)$,
$(S, H_1) \models \Sigma_1$, and $(S, H_2) \models \Sigma_2$

$(S, H) \models P(x, \vec{y})$ iff $\exists$ a rule $(P(X, \vec{Y}) \leftarrow \exists \vec{Z} : \Pi \wedge \Sigma) \in \mathcal{P}$ s.t.
$(S, H) \models \exists \vec{Z} : (\Pi \wedge \Sigma)[x/X, \vec{y}/\vec{Y}]$ and
$ldom(H) \cap \{S(y) \mid y \in \vec{y}\} = \emptyset$, i.e., $\vec{y}$ are dangling

Very Simple Examples

- Bounded heap



$$\exists x_1, x_2, x_3 \,.\, x_1 \mapsto \{f : x_2\} * x_2 \mapsto \{f : x_3\} \\ \wedge\, x_1 \neq x_3$$

- Singly linked lists

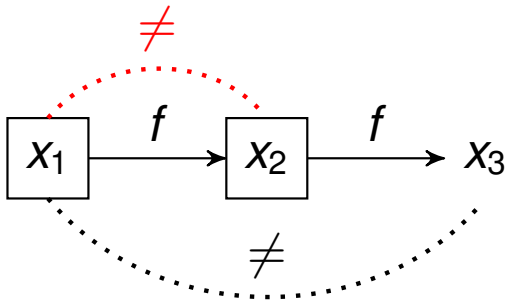
$$\exists x_1, x_2 \,.\, E \mapsto \{\texttt{next} : x_1\} \; * \mathit{sll}(x_1, x_2) \\ * x_2 \mapsto \{\texttt{next} : F\}$$

where rules of *sll*:

$$\mathit{sll}(E, F) \leftarrow \; E = F \wedge \mathit{emp}$$
$$\mathit{sll}(E, F) \leftarrow \; E \neq F \wedge \exists X_{tl} \,.\, E \mapsto \{\texttt{next} : X_{tl}\} * \mathit{sll}(X_{tl}, F)$$

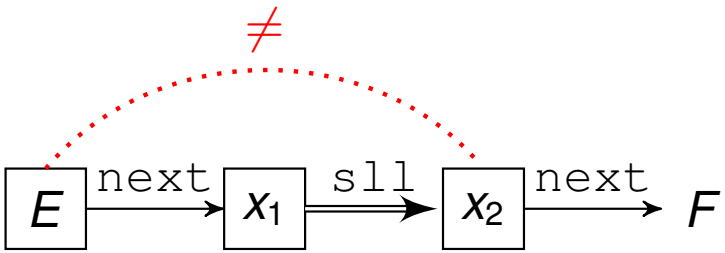
Very Simple Examples

- Bounded heap



$$\exists x_1, x_2, x_3 \,.\, x_1 \mapsto \{f : x_2\} * x_2 \mapsto \{f : x_3\} \\ \wedge\, x_1 \neq x_3$$

- Singly linked lists



$$\exists x_1, x_2 \,.\, E \mapsto \{\texttt{next} : x_1\} \; * \mathit{sll}(x_1, x_2) \\ * x_2 \mapsto \{\texttt{next} : F\}$$

where rules of *sll*:

$$\mathit{sll}(E, F) \leftarrow \; E = F \wedge \mathit{emp}$$
$$\mathit{sll}(E, F) \leftarrow \; E \neq F \wedge \exists X_{tl} \,.\, E \mapsto \{\texttt{next} : X_{tl}\} * \mathit{sll}(X_{tl}, F)$$

Program Specification

```
// Pre: slist(root, M0)
// Post: slist(root, M0) ∧ ret = 1 ↔ key ∈ M0
int search(struct sListNode * root, int key){
    struct sListNode *cur = root;
    while (cur != NULL && cur->data < key)
// Inv: ∃M1, M2. slseg(root, cur, M1) * slist(cur, M2) ∧
//      M0 = M1 ⊎ M2 ∧ M1 ≤ M2 ∧ key ∈ M0 ↔ key ∈ M2
        cur = cur->next;
    if (cur != NULL && cur->data == key)
        return 1;
    return 0;
}
```

where

$$slist(E, M) \leftarrow E = \texttt{null} \wedge M = \emptyset \wedge emp$$

$$slist(E, M) \leftarrow \exists X, d, M_1 \,.\, E \mapsto \{\texttt{data} : d, \texttt{next} : X\} * slist(X, M_1) \wedge M = M_1 \uplus \{d\} \wedge d \leq M_1$$

$$slseg(E, F, M) \leftarrow E = F \wedge M = \emptyset \wedge emp$$

$$slseg(E, F, M) \leftarrow \exists X, d, M_1 \,.\, E \mapsto \{\texttt{data} : d, \texttt{next} : X\} * slseg(X, F, M_1) \wedge M = M_1 \uplus \{d\} \wedge d \leq M_1$$

Verification Condition

```
    while (cur != NULL && cur->data < key)
// Inv: ∃M1, M2. slseg(root, cur, M1) * slist(cur, M2) ∧
//      M0 = M1 ⊎ M2 ∧ M1 ≤ M2 ∧ key ∈ M0 ↔ key ∈ M2
        cur = cur->next;
```

Inductiveness of Inv: $post(Inv \wedge b, S) \Rightarrow Inv'$

```
/* Inv:  ∃M1, M2. slseg(root, cur1, M1) * slist(cur1, M2) ∧
         M0 = M1 ⊎ M2 ∧ M1 ≤ M2 ∧ key ∈ M0 ↔ key ∈ M2
*/
        assume(cur1 != NULL && cur1->data < key);
        cur2 = cur1->next;
/* Inv': ∃M1', M2'. slseg(root, cur2, M1') * slist(cur2, M2') ∧
         M0 = M1' ⊎ M2' ∧ M1' ≤ M2' ∧ key ∈ M0 ↔ key ∈ M2'
*/
```

Verification Condition

$$\mathit{slseg}(\mathit{root},\mathit{cur1},M_1) \ * \ \mathit{cur1} \mapsto \{(\mathtt{data}, d), (\mathtt{next}, \mathit{cur2})\} * \mathit{slist}(\mathit{cur2},M_2')$$

$$\wedge\, M_0 = M_1 \uplus \{d\} \uplus M_2' \wedge M_1 \leq d \leq M_2' \wedge \ldots$$

$$\vdash \quad \mathit{slseg}(\mathit{root},\mathit{cur2},M_1') * \mathit{slist}(\mathit{cur2}, M_2')$$

$$\wedge\, M_0 = M_1' \uplus M_2' \wedge M_1' \leq M_2' \wedge \ldots$$

Proof System for $\varphi_1 \vdash_\delta \varphi_2$

$$\vdash_\Pi \frac{\Pi_1 \Rightarrow \Pi_2}{\Pi_1 \vdash_\delta \Pi_2} \qquad \bot_L \frac{}{\varphi_1 * u \overset{t}{\mapsto} \vec{v} * u \overset{t}{\mapsto} \vec{w} \vdash_\delta \varphi_2}$$

$$=_L \frac{\varphi_1[u/v] \vdash_{\delta \setminus \{v\}} \varphi_2[u/v]}{\varphi_1 \wedge u = v \vdash_\delta \varphi_2} \qquad P_R \frac{\varphi_1 \vdash_\delta \exists \vec{x} \cdot (\varphi_2 * \varphi_i^P(\vec{u}))}{\varphi_1 \vdash_\delta \exists \vec{x} \cdot (\varphi_2 * P(\vec{u}))} P(\vec{u}) \leftarrow \varphi_i^P$$

$$*P \frac{\varphi_1 \vdash_{\delta \cup \vec{u}} \exists \vec{x} \cdot (\varphi_2 \wedge \vec{u} = \vec{v})}{\varphi_1 * P(\vec{u}) \vdash_\delta \exists \vec{x} \cdot (\varphi_2 * P(\vec{v}))} \vec{u} \# \vec{x}$$

$$* \mapsto \frac{\varphi_1 \vdash_{\delta \cup \{u\}} \exists \vec{x} \cdot (\varphi_2 \wedge u = y \wedge \vec{v} = \vec{w})}{\varphi_1 * u \overset{t}{\mapsto} \vec{v} \vdash_\delta \exists \vec{x} \cdot (\varphi_2 * y \overset{t}{\mapsto} \vec{w})} u \notin \vec{x},\ \vec{v} \# \vec{x}$$

$$\mathrm{ID} \frac{\varphi_1 * \varphi_1^P(\vec{u}) \vdash_\delta \varphi_2 \quad \ldots \quad \varphi_1 * \varphi_m^P(\vec{u}) \vdash_\delta \varphi_2}{\varphi_1 * P(\vec{u}) \vdash_\delta \varphi_2}$$

but not enough...

Lemmas Needed

$$\text{Cut}\frac{\varphi_1 \vdash_{\delta \cup \vec{u}} \exists \vec{x} \cdot (\varphi_2 \wedge \vec{u} = \vec{v})}{\varphi_1 * \varphi_1'(\vec{u}) \vdash_{\delta} \exists \vec{x} \cdot (\varphi_2 * P(\vec{v}))} \vec{u} \# \vec{x}, \varphi_1'(\vec{u}) \vdash_{\delta'} P(\vec{u}), \delta' \subseteq \delta$$

Lemma 1: *slist* unfolded once

$$\exists d \,.\, E \mapsto \{\texttt{data} : d, \texttt{next} : F\} * \mathit{emp} \wedge M = \{d\} \vdash_{\{F\}} \mathit{slseg}(E, F, M)$$

Lemma 2: *slist* composition

$$\mathit{slseg}(X, Y, M_1) * \mathit{slseg}(Y, Z, M_2) \wedge M_1 \leq M_2 \vdash_{\{Z\}} \mathit{slseg}(X, Z, M_1 \uplus M_2)$$

How to Generate Lemmas?

Main idea: ID predicates used in practice, when well written, have simple lemmas.

Example: *slseg* composition lemmas

$$slseg(X, Y, M_1) * slseg(Y, Z, M_2) \wedge M_1 \leq M_2 \vdash slseg(Y, Z, M_1 \uplus M_2)$$

but for another definition of *slseg*:

$$slseg(E, M, F, M') \leftarrow E = F \wedge M = M' \wedge emp$$
$$slseg(E, M, F, M') \leftarrow \exists X, d, M_1 \,.\, E \mapsto \{\texttt{data} : d, \texttt{next} : X\} * slseg(X, M_1, F, M') \wedge M = M_1 \cup \{d\} \wedge d \leq M_1$$

Lemma: *slist* composition

$$slseg(X, M_1, Y, M_2) * slseg(Y, M_2, Z, M_3) \vdash slseg(X, M_1, Z, M_3)$$

How to Generate Lemmas?

Main idea: ID predicates used in practice, when well written, have simple lemmas.

Example: *slseg* composition lemmas

$$slseg(X, Y, M_1) * slseg(Y, Z, M_2) \wedge M_1 \leq M_2 \vdash slseg(Y, Z, M_1 \uplus M_2)$$

but for another definition of *slseg*:

$$\begin{aligned} slseg(E, M, F, M') \leftarrow\ & E = F \wedge M = M' \wedge emp \\ slseg(E, M, F, M') \leftarrow\ & \exists X, d, M_1 \,.\, E \mapsto \{\texttt{data} : d, \texttt{next} : X\} * slseg(X, M_1, F, M') \\ & \wedge\ M = M_1 \cup \{d\} \wedge\ d \leq M_1 \end{aligned}$$

Lemma: *slist* composition

$$slseg(X, M_1, Y, M_2) * slseg(Y, M_2, Z, M_3) \vdash slseg(X, M_1, Z, M_3)$$

Syntactically compositional predicates (1/2)

Restriction 1: Tail recursive ID

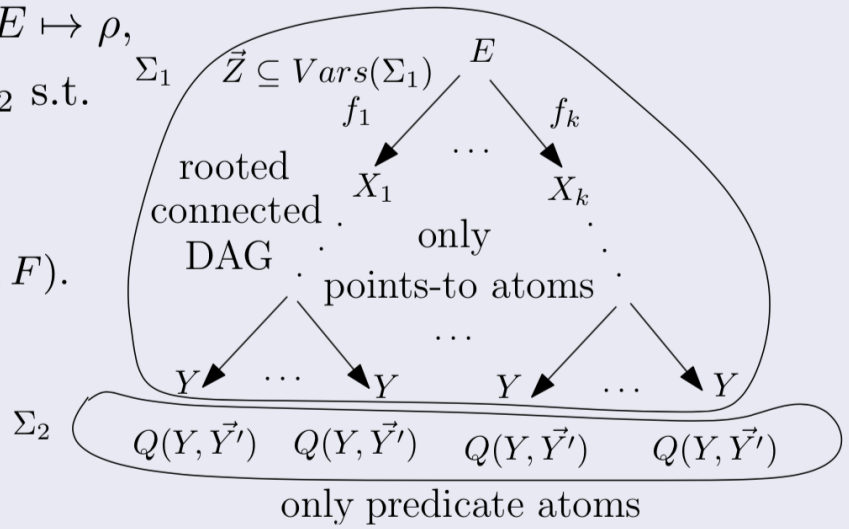
Inductive definitions

A predicate $P(E, \vec{F})$ is defined by rules $R : P(E, \vec{F}) ::= \exists \vec{Z}.\ \Pi \wedge \Sigma$ of the form

- a base rule: $\Sigma \triangleq emp$ or $\Sigma \triangleq E \mapsto \rho$,
- an inductive rule: $\Sigma \triangleq \Sigma_1 * \Sigma_2$ s.t.

The rule below is disallowed,

$lsegb(E, F) \quad ::=$
$\exists X.\ lsegb(E, X) * X \mapsto (\mathtt{next}, F).$



Restriction 2: Well formed set of predicates defined inductively $\mathcal{P}$:

i. no mutual recursion: "P calls Q" is a partial order over $\mathcal{P}$

ii. predicates not related by "calls" employ disjoint sets of fields

Syntactically compositional predicates (2/2)

Restriction 3: Syntactically compositional inductive definitions

$$P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \leftarrow \exists \vec{Y} \,.\, \Pi \wedge \Sigma$$

i. parameters are source $\vec{\alpha}$, destination $\vec{\beta}$, resp. static parameters $\vec{\xi}$

ii. exactly one base rule $P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \leftarrow \vec{\alpha} = \vec{\beta} * emp$

iii. each inductive rule $P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \leftarrow \exists \vec{Y} \,.\, \Pi \wedge \Sigma_1 * (\Sigma_2 * P(\vec{\gamma}, \vec{\beta}, \vec{\xi}))$ s.t.

   a. Σ_1 is only points-to atoms, Σ_2 only (possibly none) predicate atoms,
   b. $\vec{\gamma} \subseteq \vec{Y}$ linked by Σ_1 to $\vec{\alpha}$,
   c. variables $\vec{\beta}$ do not occur in Π, Σ_1, Σ_2, or $\vec{\gamma}$

Corollaries

Property: a predicate specifies a connected graph where

- all vertices are reachable from root location in $\vec{\alpha}$
- only destination $\vec{\beta}$ and static $\vec{\xi}$ locations are dangling
- only one model satisfies a predicate atom

Theorem

If P is syntactically compositional and $\vec{\pi}$ pending (i.e., in δ) then

$$P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) * P(\vec{\beta}, \vec{\pi}, \vec{\xi}) \vdash_{\delta} P(\vec{\alpha}, \vec{\pi}, \vec{\xi})$$

Examples

Example (Doubly linked lists)

$$\mathit{dllseg}(E, P, F, P') \leftarrow E = F \wedge P = P' \wedge \mathit{emp}$$
$$\mathit{dllseg}(E, P, F, P') \leftarrow \exists X \,.\, E \mapsto \{\mathtt{next} : X, \mathtt{prev} : P\} * \mathit{dllseg}(X, E, F, P')$$

Lemma: $\mathit{dllseg}(X, P_1, Y, P_2) * \mathit{dllseg}(Y, P_2, Z, P_3) \vdash_\delta \mathit{dllseg}(X, P_1, Z, P_3)$

Example (Binary Search Trees)

$$\mathit{bsthole}(E, M_1, F, M_2) \leftarrow E = F \wedge M_1 = M_2 \wedge \mathit{emp}$$
$$\mathit{bsthole}(E, M_1, F, M_2) \leftarrow \exists X, Y, d, M_3, M_4 \,.\, E \mapsto \{\mathtt{left} : X, \mathtt{right} : Y, \mathtt{data} : d\} * \mathit{bsthole}(X, M_3, F, M_2) * \mathit{bst}(Y, M_4) \wedge M = M_3 \cup \{d\} \cup M_4 \wedge M_3 < d < M_4$$
$$\mathit{bsthole}(E, M_1, F, M_2) \leftarrow \ldots \mathit{bsthole}(Y, M_4, F, M_2) \ldots$$

Lemma: $\mathit{bsthole}(X, M_1, Y, M_2) * \mathit{bsthole}(Y, M_2, Z, M_3) \vdash \mathit{bsthole}(X, M_1, Z, M_3)$

Examples

Example (Doubly linked lists)

$$\begin{aligned} \mathit{dllseg}(E, P, F, P') &\leftarrow E = F \wedge P = P' \wedge \mathit{emp} \\ \mathit{dllseg}(E, P, F, P') &\leftarrow \exists X \,.\, E \mapsto \{\mathtt{next} : X, \mathtt{prev} : P\} * \mathit{dllseg}(X, E, F, P') \end{aligned}$$

Lemma: $\mathit{dllseg}(X, P_1, Y, P_2) * \mathit{dllseg}(Y, P_2, Z, P_3) \vdash_\delta \mathit{dllseg}(X, P_1, Z, P_3)$

Example (Binary Search Trees)

$$\begin{aligned} \mathit{bsthole}(E, M_1, F, M_2) &\leftarrow E = F \wedge M_1 = M_2 \wedge \mathit{emp} \\ \mathit{bsthole}(E, M_1, F, M_2) &\leftarrow \exists X, Y, d, M_3, M_4 \,.\, E \mapsto \{\mathtt{left} : X, \mathtt{right} : Y, \mathtt{data} : d\} \\ &\quad * \mathit{bsthole}(X, M_3, F, M_2) * \mathit{bst}(Y, M_4) \\ &\quad \wedge M = M_3 \cup \{d\} \cup M_4 \wedge M_3 < d < M_4 \\ \mathit{bsthole}(E, M_1, F, M_2) &\leftarrow \ldots \mathit{bsthole}(Y, M_4, F, M_2) \ldots \end{aligned}$$

Lemma: $\mathit{bsthole}(X, M_1, Y, M_2) * \mathit{bsthole}(Y, M_2, Z, M_3) \vdash \mathit{bsthole}(X, M_1, Z, M_3)$

Derived Lemmas

Completion lemmas: eg,

$$bsthole(X, M_1, Y, M_2) * bst(Y, M_2) \vdash bst(X, M_1)$$

because *bst* rules are ones of *bsthole* when Y is `null` and M_2 is $\emptyset$.

Theorem (Completion lemmas soundness)

Let $P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \in \mathcal{P}$ *be a syntactically compositional predicate.*
If $Q(\vec{\alpha}, \vec{\xi}) \in \mathcal{P}$ *is a completion of* P *with respect to constants* $\vec{c}$*, then*

$$Q(\vec{\alpha}, \vec{\xi}) \Leftrightarrow P(\vec{\alpha}, \vec{c}, \vec{\xi}) \text{ and}$$

$$\exists \vec{\beta} \,.\, P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) * Q(\vec{\beta}, \vec{\xi}) \vdash_{\delta} P'(\vec{\alpha}, \vec{\xi})$$

Derived Lemmas

Stronger lemmas: eg,

$$\mathit{natslseg}(X, M_1, Y, M_2) \vdash_\delta \mathit{slseg}(X, M_1, Y, M_2)$$
$$\mathit{natslseg}(X, M_1, Y, M_2) * \mathit{slseg}(Y, M_2, Z, M_3) \vdash_\delta \mathit{slseg}(X, M_1, Z, M_3)$$

where

$$\mathit{natslseg}(E, M, F, M') \leftarrow E = F \wedge M = M' \wedge \mathit{emp}$$
$$\mathit{natslseg}(E, M, F, M') \leftarrow \exists X, d, M_1 \,.\, E \mapsto \{\mathtt{data} : d, \mathtt{next} : X\} * \mathit{slseg}(X, M_1, F, M') \wedge M = M_1 \cup \{d\} \wedge d \leq M_1 \wedge M \geq 0$$

So *natslseg* inductive rule pure part $\Rightarrow$ pure part of *slseg*.

Theorem (Stronger lemmas soundness)

Let $P(\vec{\alpha}, \vec{\beta}, \vec{\xi}), Q(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \in \mathcal{P}$ be syntactically compositional predicates.
If Q is stronger than P, then

$$Q(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \vdash_\delta P(\vec{\alpha}, \vec{\beta}, \vec{\xi}),$$
$$\exists \vec{\beta} \,.\, Q(\vec{\alpha}, \vec{\beta}, \vec{\xi}) * P(\vec{\beta}, \vec{\gamma}, \vec{\xi}) \vdash_\delta P(\vec{\alpha}, \vec{\gamma}, \vec{\xi}),$$
$$\textit{etc.}$$

Experiments

SPEN proof search heuristics:

- rewrite spatial formula by applying rules and lemmas, i.e.,
  $\psi \models \varphi$ (dis-)proved if $\psi \vdash \psi_1 \vdash \ldots \vdash \psi_k \equiv \varphi$
- pure and data constraints checked using SMT solvers

Experiments

SPEN proof search heuristics:

- rewrite spatial formula by applying rules and lemmas, i.e.,
  $\psi \models \varphi$ (dis-)proved if $\psi \vdash \psi_1 \vdash \ldots \vdash \psi_k \equiv \varphi$
- pure and data constraints checked using SMT solvers

Data struct.	Iterative Proc.	#VC	Lemma (#b, #r, #p, #c, #d)	$\Rightarrow_{\mathbb{D}}$	Time (s) SPEN	Time (s) SMT
sorted lists	`search`	4	(1, 3, 3, 1, 3)	5	1.108	0.10
	`insert`	8	(4, 6, 3, 1, 2)	7	2.902	0.15
	`delete`	4	(2, 2, 4, 1, 1)	6	1.108	0.10
BST	`search`	4	(2, 3, 6, 2, 2)	6	1.191	0.15
	`insert`	14	(15, 18, 27, 4, 6)	19	3.911	0.55
	`delete`	25	(13, 19, 82, 8, 5)	23	8.412	0.58
AVL	`search`	4	(2, 3, 6, 2, 2)	6	1.573	0.15
	`insert`	22	(18, 28, 74, 6, 8)	66	6.393	1.33
RBT	`search`	4	(2, 3, 6, 2, 2)	6	1.171	0.15
	`insert`	21	(27, 45, 101, 7, 10)	80	6.962	2.53

number of applications for basic (#*b*), matching (#*p*), inductive (#*r*), composition (#*c*), and derived (#*d*) lemmas.

Experiments

SPEN proof search heuristics:

- rewrite spatial formula by applying rules and lemmas, i.e.,
  $\psi \models \varphi$ (dis-)proved if $\psi \vdash \psi_1 \vdash \ldots \vdash \psi_k \equiv \varphi$
- pure and data constraints checked using SMT solvers

Data structure	#VC	Lemma (#b, #r, #p, #c, #d)	Time-SPEN(s) SPEN	SPEN-TA
Nested linked lists	16	(17,47,14,8,0)	4.428	4.382
Skip lists 2 levels	4	(11,16,1,1,0)	1.629	1.636
Skip lists 3 levels	10	(16,32,29,17,0)	3.858	3.485

number of applications for basic (#*b*), matching (#*p*), inductive (#*r*), composition (#*c*), and derived (#*d*) lemmas.

Related Work: Other Proof Systems for SL

- Cyclic proofs: CYCLIST-SL [Brotherston *et al*'12]
- Structural induction: SLEEK [Chin *et al*'15]
- Mutual induction: SongBird [Chin *et al*'16]
- Heuristic proof search with lemmas on ID
  $\longrightarrow$ user provided lemmas: VeriFast [Jacobs *et al*'08], DRYAD [Qiu *et al*'13]
  $\longrightarrow$ automatically generated lemmas, SPEN [ATVA'15]

Related Work: Decidable Fragments

Restrictions: on boolean structure and inductively defined predicates

- conjunctive formulas + singly linked lists
  $\longrightarrow$ reduction to PTIME graph homomorphism, SeLoger [Cook *et al*'11]
  $\longrightarrow$ encoding in Horn clauses, Asterix [Rybalchenko *et al*'11]
- conjunctive formulas + list like ID eg, nested lists, skip lists, etc.
  $\longrightarrow$ reduction to PTIME graph homomorphism, [ESOP'13]
  $\longrightarrow$ reduction to PTIME tree-automata $\in$, SPEN [APLAS'14]
- lists, trees and data $\longrightarrow$ reduction to SMT, GRASShopper [Piskac *et al*'14]
- conjunctive formulas + restricted BTW ID predicates
  $\longrightarrow$ reduction to EXPTIME tree-automata $\subseteq$, SLIDE [Iosif *et al*'14]
- conjunctive formulas + bounded tree width ID predicates
  $\longrightarrow$ (multiple-EXPTIME) reduction to MSO [Iosif *et al*'13]

Related Work: Decidable Fragments

Restrictions: on boolean structure and inductively defined predicates

- conjunctive formulas + singly linked lists
  $\longrightarrow$ reduction to PTIME graph homomorphism, SeLoger [Cook *et al*'11]
  $\longrightarrow$ encoding in Horn clauses, Asterix [Rybalchenko *et al*'11]
- conjunctive formulas + list like ID eg, nested lists, skip lists, etc.
  $\longrightarrow$ reduction to PTIME graph homomorphism, [ESOP'13]
  $\longrightarrow$ reduction to PTIME tree-automata $\in$, SPEN [APLAS'14]
- lists, trees and data $\longrightarrow$ reduction to SMT, GRASShopper [Piskac *et al*'14]
- conjunctive formulas + restricted BTW ID predicates
  $\longrightarrow$ reduction to EXPTIME tree-automata $\subseteq$, SLIDE [Iosif *et al*'14]
- conjunctive formulas + bounded tree width ID predicates
  $\longrightarrow$ (multiple-EXPTIME) reduction to MSO [Iosif *et al*'13]

Related Work: Decidable Fragments

Restrictions: on boolean structure and inductively defined predicates

- conjunctive formulas + singly linked lists
  $\longrightarrow$ reduction to PTIME graph homomorphism, SeLoger [Cook *et al*'11]
  $\longrightarrow$ encoding in Horn clauses, Asterix [Rybalchenko *et al*'11]
- conjunctive formulas + list like ID eg, nested lists, skip lists, etc.
  $\longrightarrow$ reduction to PTIME graph homomorphism, [ESOP'13]
  $\longrightarrow$ reduction to PTIME tree-automata $\in$, SPEN [APLAS'14]
- lists, trees and data $\longrightarrow$ reduction to SMT, GRASShopper [Piskac *et al*'14]

- conjunctive formulas + restricted BTW ID predicates
  $\longrightarrow$ reduction to EXPTIME tree-automata $\subseteq$, SLIDE [Iosif *et al*'14]
- conjunctive formulas + bounded tree width ID predicates
  $\longrightarrow$ (multiple-EXPTIME) reduction to MSO [Iosif *et al*'13]

Related Work: Decidable Fragments

Restrictions: on boolean structure and inductively defined predicates

- conjunctive formulas + singly linked lists
  $\longrightarrow$ reduction to PTIME graph homomorphism, SeLoger [Cook *et al*'11]
  $\longrightarrow$ encoding in Horn clauses, Asterix [Rybalchenko *et al*'11]
- conjunctive formulas + list like ID eg, nested lists, skip lists, etc.
  $\longrightarrow$ reduction to PTIME graph homomorphism, [ESOP'13]
  $\longrightarrow$ reduction to PTIME tree-automata $\in$, SPEN [APLAS'14]
- lists, trees and data $\longrightarrow$ reduction to SMT, GRASShopper [Piskac *et al*'14]

- conjunctive formulas + restricted BTW ID predicates
  $\longrightarrow$ reduction to EXPTIME tree-automata $\subseteq$, SLIDE [Iosif *et al*'14]
- conjunctive formulas + bounded tree width ID predicates
  $\longrightarrow$ (multiple-EXPTIME) reduction to MSO [Iosif *et al*'13]

Related Work: Decidable Fragments

Restrictions: on boolean structure and inductively defined predicates

- conjunctive formulas + singly linked lists
  $\longrightarrow$ reduction to PTIME graph homomorphism, SeLoger [Cook *et al*'11]
  $\longrightarrow$ encoding in Horn clauses, Asterix [Rybalchenko *et al*'11]
- conjunctive formulas + list like ID eg, nested lists, skip lists, etc.
  $\longrightarrow$ reduction to PTIME graph homomorphism, [ESOP'13]
  $\longrightarrow$ reduction to PTIME tree-automata $\in$, SPEN [APLAS'14]
- lists, trees and data $\longrightarrow$ reduction to SMT, GRASShopper [Piskac *et al*'14]

- conjunctive formulas + restricted BTW ID predicates
  $\longrightarrow$ reduction to EXPTIME tree-automata $\subseteq$, SLIDE [Iosif *et al*'14]
- conjunctive formulas + bounded tree width ID predicates
  $\longrightarrow$ (multiple-EXPTIME) reduction to MSO [Iosif *et al*'13]

Merci!