Automating Proofs for Iterative Programs on Complex Data Structures

Constantin Enea[1], Mihaela Sighireanu[1] and Zhilin Wu[2,1]

[1] LIAFA, University Paris Diderot & CNRS
[2] IS of Chinese Academy of Science

February 27th, 2015

Outline

SL + ID = Good theory to work complex data structures

- Separation Logic (SL) supports local reasoning

  $\longrightarrow$ compact proofs

- Inductive Definitions (ID) naturally capture complex dynamic data structures

$$btree(\mathsf{E}) ::= (\mathsf{E} = \mathsf{nil} \wedge \mathtt{emp})$$

$$btree(\mathsf{E}) ::= (\mathsf{E} \neq \mathsf{nil} \wedge \exists \mathsf{L}, \mathsf{R}, \mathsf{d}.\quad \mathsf{E} \mapsto ((\mathtt{left}, \mathsf{L}), (\mathtt{right}, \mathsf{R}), (\mathtt{data}, \mathsf{d})) \star btree(\mathsf{L}) \star btree(\mathsf{R}))$$

- Decidability results for expressive fragments without data
  - sat [LICS'14]
  - entailment [CADE'13]

Reasoning about recursive programs

```
/* @pre:  btree(t) */
btree* insert(btree* t, int n) {
  if (t == NULL) {
    btree* r = btree_alloc(n, NULL, NULL);
    /* r ↦ ((left, nil), (right, nil), (data, n)) */
    return r;
  }
  if (*) {
    /* ∃L, R, d. t ↦ ((left, L), (right, R), (data, d)) ⋆ btree(L) ⋆ btree(R)
    t->left = insert (t->left, n);
    /* ∃L′, R, d. t ↦ ((left, L′), (right, R), (data, d)) ⋆ btree(L′) ⋆ btree(R)
  } else {
    /* ∃L, R, d. t ↦ ((left, L), (right, R), (data, d)) ⋆ btree(L) ⋆ btree(R)
    t->right = insert (t->right, n);
    /* ∃L, R′, d. t ↦ ((left, L), (right, R′), (data, d)) ⋆ btree(L) ⋆ btree(R′)
  }
  return t;
} /* @post:  btree(@ret) */
```

Reasoning about iterative programs

```
/* @pre:  btree(t) */
btree* deleteLeftMost(btree* t) {
  if (t == NULL) return t;
  if (t->left == NULL) {
    btree* r = t->right; /* t ↦ ((left, nil), (right, r), (data, d)) ⋆  btree(r) */
    ...
    return r;
  }
  btree *p = t, *tt = t->left;


  while (tt->left != NULL) {
    p = tt; tt = tt->left;
  }
  if (tt->right == NULL) p->left = NULL;
  else p->left = tt->right;
  ...
  return t;
} /* @post:  btree(@ret) */
```

Reasoning about iterative programs

```
/* @pre:  btree(t) */
btree* deleteLeftMost(btree* t) {
  if (t == NULL) return t;
  if (t->left == NULL) {
    btree* r = t->right; /* t ↦ ((left, nil), (right, r), (data, d)) ⋆  btree(r) */
    ...
    return r;
  }
  btree *p = t, *tt = t->left;
  /* @inv:p ≠ nil∧tt ≠ nil∧∃R, d.  btreeh(t, p)⋆  p ↦ ((left, tt), (right, R), (data, d))⋆
btree(tt) ⋆  btree(R) */
  while (tt->left != NULL) {
    p = tt; tt = tt->left;
  }
  if (tt->right == NULL) p->left = NULL;
  else p->left = tt->right;
  ...
  return t;
} /* @post:  btree(@ret) */
```

Reasoning about iterative programs

- ID for data structures fragments (with a hole)
- Proving invariants require complex *lemmas*, e.g.,

$$
\begin{aligned}
\big(\exists p', R, R', d, d'.\ \mathit{btreeh}(t, p') \star\ p' \mapsto ((\texttt{left}, p), (\texttt{right}, R), (\texttt{data}, d)) \star\ \mathit{btree}(R) \\
p \mapsto ((\texttt{left}, tt), (\texttt{right}, R'), (\texttt{data}, d')) \star\ \mathit{btree}(tt)\big) \\
\Longrightarrow \big(\exists R, d.\ \mathit{btreeh}(t, p) \star\ p \mapsto ((\texttt{left}, tt), (\texttt{right}, R), (\texttt{data}, d)) \\
\star\ \mathit{btree}(R) \star\ \mathit{btree}(tt)\big)
\end{aligned}
$$

Reasoning about iterative programs

- ID for data structures fragments (with a hole)
- Proving invariants require complex *lemmas*, e.g.,

$$\begin{aligned}(\exists p', R, R', d, d'.\ \mathit{btreeh}(t, p') \star\ p' \mapsto ((\texttt{left}, p), (\texttt{right}, R), (\texttt{data}, d)) \star\ \mathit{btree}(R) \\ p \mapsto ((\texttt{left}, tt), (\texttt{right}, R'), (\texttt{data}, d')) \star\ \mathit{btree}(tt)) \\ \Longrightarrow (\exists R, d.\ \mathit{btreeh}(t, p) \star\ p \mapsto ((\texttt{left}, tt), (\texttt{right}, R), (\texttt{data}, d)) \\ \star\ \mathit{btree}(R) \star\ \mathit{btree}(tt))\end{aligned}$$

- Worse when adding data, even for simple data structures, e.g., sorted lists:

$$\begin{aligned}&\mathit{lseg}(E, M, F) ::= E = F \wedge \texttt{emp} \wedge M = \emptyset \\ &\mathit{lseg}(E, M, F) ::= \exists X, v, M_1.\ E \mapsto \{(\texttt{next}, X), (\texttt{data}, v)\} \star\ \mathit{lseg}(X, M_1, F) \\ &\qquad\qquad \wedge\ v \leq M_1 \wedge M = M_1 \cup \{v\}\end{aligned}$$

Reasoning about iterative programs

- ID for data structures fragments (with a hole)
- Proving invariants require complex *lemmas*, e.g.,

$$\begin{aligned}(\exists p', R, R', d, d'.\ \mathit{btreeh}(t, p') \star\ p' \mapsto ((\texttt{left}, p), (\texttt{right}, R), (\texttt{data}, d)) \star\ \mathit{btree}(R)\\ p \mapsto ((\texttt{left}, tt), (\texttt{right}, R'), (\texttt{data}, d')) \star\ \mathit{btree}(tt))\\ \Longrightarrow (\exists R, d.\ \mathit{btreeh}(t, p) \star\ p \mapsto ((\texttt{left}, tt), (\texttt{right}, R), (\texttt{data}, d))\\ \star\ \mathit{btree}(R) \star\ \mathit{btree}(tt))\end{aligned}$$

- Worse when adding data, even for simple data structures, e.g., sorted lists:

$$\exists E_2.\ \mathit{lseg}(E_1, M_1, E_2) \star\ \mathit{lseg}(E_2, M_2, E_3) \wedge M_1 \leq M_2 \Longrightarrow \exists M.\ \mathit{lseg}(E_1, M, E_3)$$

Reasoning about iterative programs

- ID for data structures fragments (with a hole)
- Proving invariants require complex *lemmas*, e.g.,

$$\begin{aligned}(\exists p', R, R', d, d'.\ \mathit{btreeh}(t, p') \star\ p' \mapsto ((\texttt{left}, p), (\texttt{right}, R), (\texttt{data}, d)) \star\ \mathit{btree}(R)\\ p \mapsto ((\texttt{left}, tt), (\texttt{right}, R'), (\texttt{data}, d')) \star\ \mathit{btree}(tt))\\ \Longrightarrow (\exists R, d.\ \mathit{btreeh}(t, p) \star\ p \mapsto ((\texttt{left}, tt), (\texttt{right}, R), (\texttt{data}, d))\\ \star\ \mathit{btree}(R) \star\ \mathit{btree}(tt))\end{aligned}$$

- Worse when adding data, even for simple data structures, e.g., sorted lists:

  $\longrightarrow$ DRYAD [POPL'12]

Aim and contribution

Automate the invariant checking for iterative programs

$\longrightarrow$ automatically infer lemmas about shape and data

Aim and contribution

Automate the invariant checking for iterative programs
$\longrightarrow$ automatically infer lemmas about shape and data

Approach:

1. propose syntax for ID supporting shape + data specifications
2. ... and simple lemma, automatically generated
3. proof strategy using lemma

Outline

Separation Logic Fragment

Formulas:

$X, Y, E \in \mathrm{LVars}$ location variables

$\vec{F} \in (\mathrm{LVars} \cup \mathrm{DVars})^*$

$x \in \mathrm{Vars}$

$\rho \subseteq (\mathcal{F} \times \mathrm{LVars}) \cup (\mathcal{D} \times \mathrm{DVars})$

$P \in \mathcal{P}$ predicates

Δ formula over DVars

$$\Pi ::= X = Y \mid X \neq Y \mid \Delta \mid \Pi \wedge \Pi \quad \text{pure formulas}$$

$$\Sigma ::= \mathtt{emp} \mid E \mapsto \rho \mid P(E, \vec{F}) \mid \Sigma \star \Sigma \quad \text{spatial formulas}$$

$$\varphi ::= \Pi \wedge \Sigma \mid \varphi \vee \varphi \mid \exists x.\ \varphi \quad \text{formulas}$$

Separation Logic Fragment

Formulas:

$X, Y, E \in LVars$ location variables $\quad \rho \subseteq (\mathcal{F} \times LVars) \cup (\mathcal{D} \times DVars)$

$\vec{F} \in (LVars \cup DVars)^*$ $\quad P \in \mathcal{P}$ predicates

$x \in Vars$ $\quad \Delta$ formula over $DVars$

$$\Pi ::= X = Y \mid X \neq Y \mid \Delta \mid \Pi \wedge \Pi \qquad \text{pure formulas}$$

$$\Sigma ::= \mathtt{emp} \mid E \mapsto \rho \mid P(E, \vec{F}) \mid \Sigma \star \Sigma \qquad \text{spatial formulas}$$

$$\varphi ::= \Pi \wedge \Sigma \mid \varphi \vee \varphi \mid \exists x.\ \varphi \qquad \text{formulas}$$

General Inductive Definitions: set of rules $P(E, \vec{F}) ::= \exists \vec{Z}. \Pi \wedge \Sigma$

- $\vec{Z} \in Vars^*$
- if Σ is emp or $E \mapsto \rho$, i.e., base rule
- if Σ with predicate atoms, i.e., inductive rule, then
  - it has only one points-to from E
  - it is connected

Examples

Sorted Lists:

$$\mathit{list}(E, M) ::= E = \mathsf{nil} \wedge \mathtt{emp} \wedge M = \emptyset$$

$$\mathit{list}(E, M) ::= (\exists X, v, M_1.\ E \mapsto \{(\mathtt{next}, X), (\mathtt{data}, v)\} \star \mathit{list}(X, M_1) \wedge v \leq M_1 \wedge M = M_1 \cup \{v\}$$

Sorted Lists Segments:

$$\mathit{lseg}(E, M, F, M') ::= E = F \wedge \mathtt{emp} \wedge M = M'$$

$$\mathit{lseg}(E, M, F, M') ::= \exists X, v, M_1.\ E \mapsto \{(\mathtt{next}, X), (\mathtt{data}, v)\} \star \mathit{lseg}(X, M_1, F, M') \wedge v \leq M_1 \wedge M = M_1 \cup \{v\}$$

Examples

Binary Search Trees:

$$bst(E, M) ::= E = \mathsf{nil} \wedge \mathtt{emp} \wedge M = \emptyset$$

$$bst(E, M) ::= \exists X, Y, M_1, M_2, v.\, E \mapsto \{(\mathtt{left}, X), (\mathtt{right}, Y), (\mathtt{data}, v)\} \star bst(X, M_1) \star bst(Y, M_2) \wedge E \neq \mathsf{nil} \wedge M = \{v\} \cup M_1 \cup M_2 \wedge M_1 < v < M_2$$

Binary Search Trees Segments:

$$bsthole(E, M_1, F, M_2) ::= E = F \wedge \mathtt{emp} \wedge M_1 = M_2,$$

$$bsthole(E, M_1, F, M_2) ::= \exists X, Y, M_3, M_4, v.\, E \mapsto \{(\mathtt{left}, X), (\mathtt{right}, Y), (\mathtt{data}, v)\} \star bst(X, M_3) \star bsthole(Y, F, M_4, M_2) \wedge M_1 = \{v\} \cup M_3 \cup M_4 \wedge M_3 < v < M_4$$

$$bsthole(E, M_1, F, M_2) ::= \exists X, Y, M_3, M_4, v.\, E \mapsto \{(\mathtt{left}, X), (\mathtt{right}, Y), (\mathtt{data}, v)\} \star bsthole(X, F, M_3, M_2) \star bst(Y, M_4) \wedge M_1 = \{v\} \cup M_3 \cup M_4 \wedge M_3 < v < M_4$$

Syntactic Compositional P

$P(E, \vec{F})$ such that:

1. $(E, \vec{F}) \triangleq (\vec{\alpha}, \vec{\beta}, \vec{\xi})$ with $\vec{\alpha}$ are source, $\vec{\beta}$ are hole, and $\vec{\xi}$ are border.
2. size (resp. types) of $\vec{\alpha}$ and $\vec{\beta}$ are (resp. position-wise) the same
3. exactly one base rule of the form

$$P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) ::= \bigwedge_i \alpha_i = \beta_i \wedge \mathsf{emp}$$

4. at least one recursive rule of the form

$$P(\vec{\alpha}, \vec{\beta}, \vec{\xi}) ::= \exists \vec{Z}.\ \Pi \wedge E \mapsto \rho \ \star\ \Sigma_r \ \star\ P(\vec{\gamma}, \vec{\beta}, \vec{\xi})$$

with Σ_r contains only predicate atoms, $\gamma \subseteq \vec{Z}$, and the variables in $\vec{\beta}$ don't occur elsewhere.

Theorem (Composition Lemma)

If P *is syntactic compositional then*

$$\left(\exists \vec{\beta}.\ \mathrm{P}(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \star\ \mathrm{P}(\vec{\beta}, \vec{\gamma}, \vec{\xi})\right) \Longrightarrow \mathrm{P}(\vec{\alpha}, \vec{\gamma}, \vec{\xi})$$

is a valid lemma

Theorem (Composition Lemma)

If P *is syntactic compositional then*

$$\left(\exists \vec{\beta}.\ \mathrm{P}(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \star\ \mathrm{P}(\vec{\beta}, \vec{\gamma}, \vec{\xi})\right) \Longrightarrow \mathrm{P}(\vec{\alpha}, \vec{\gamma}, \vec{\xi})$$

is a valid lemma

Examples:

$$\left(\exists \mathrm{G}, \mathrm{M}_2.\mathit{lseg}(\mathrm{E}, \mathrm{M}, \mathrm{G}, \mathrm{M}_2) \star\ \mathit{lseg}(\mathrm{G}, \mathrm{M}_2, \mathrm{F}, \mathrm{M}_1)\right) \Longrightarrow \mathit{lseg}(\mathrm{E}, \mathrm{M}, \mathrm{F}, \mathrm{M}_1)$$

Theorem (Composition Lemma)

If P *is syntactic compositional then*

$$\left(\exists \vec{\beta}.\ \mathrm{P}(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \star\ \mathrm{P}(\vec{\beta}, \vec{\gamma}, \vec{\xi})\right) \Longrightarrow \mathrm{P}(\vec{\alpha}, \vec{\gamma}, \vec{\xi})$$

is a valid lemma

Examples:

$$\left(\exists \mathrm{G}, \mathrm{M}_2.\mathit{bsthole}(\mathrm{E}, \mathrm{M}, \mathrm{G}, \mathrm{M}_2) \star\ \mathit{bsthole}(\mathrm{G}, \mathrm{M}_2, \mathrm{F}, \mathrm{M}_1)\right) \Longrightarrow \mathit{bsthole}(\mathrm{E}, \mathrm{M}, \mathrm{F}, \mathrm{M}_1)$$

Syntactic Completive P and P′

Let $P(\vec{\alpha}, \vec{\beta}, \vec{\xi})$ and $P'(\vec{\alpha}, \vec{\xi})$ two inductive definitions with same parameters $\vec{\alpha}$ and $\vec{\xi}$.
P' is a completion of P with respect to a pair of constants $\vec{c}$ if the rules of P' are obtained from the rules of P by setting $\vec{\beta} = \vec{c}$.

Syntactic Completive P and P′

Let $P(\vec{\alpha}, \vec{\beta}, \vec{\xi})$ and $P'(\vec{\alpha}, \vec{\xi})$ two inductive definitions with same parameters $\vec{\alpha}$ and $\vec{\xi}$.
P' is a completion of P with respect to a pair of constants $\vec{c}$ if the rules of P' are obtained from the rules of P by setting $\vec{\beta} = \vec{c}$.

Examples:

- *list* a completion of *lseg* wrt nil, $\emptyset$,
- *bst* a completion of *bsthole* wrt nil, $\emptyset$

Theorem (Completion Lemmas)

If P' *is a completion of* P *wrt* $\vec{c}$ *and* P *is syntactically compositional then the following lemmas are valid:*

$$\mathrm{P}'(\vec{\alpha}, \vec{\xi}) \Leftrightarrow \mathrm{P}(\vec{\alpha}, \vec{c}, \vec{\xi})$$

and

$$\exists \vec{\beta}.\ \mathrm{P}(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \star\ \mathrm{P}'(\vec{\beta}, \vec{\xi}) \Longrightarrow \mathrm{P}'(\vec{\alpha}, \vec{\xi})$$

Theorem (Completion Lemmas)

If P' *is a completion of* P *wrt* $\vec{c}$ *and* P *is syntactically compositional then the following lemmas are valid:*

$$\mathsf{P}'(\vec{\alpha}, \vec{\xi}) \Leftrightarrow \mathsf{P}(\vec{\alpha}, \vec{c}, \vec{\xi})$$

and

$$\exists \vec{\beta}.\ \mathsf{P}(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \star\ \mathsf{P}'(\vec{\beta}, \vec{\xi}) \Longrightarrow \mathsf{P}'(\vec{\alpha}, \vec{\xi})$$

Examples:

$$\mathit{list}(\mathsf{E}, \mathsf{M}) \Leftrightarrow \mathit{lseg}(\mathsf{E}, \mathsf{M}, \mathsf{nil}, \emptyset)$$

Theorem (Completion Lemmas)

If P' *is a completion of* P *wrt* $\vec{c}$ *and* P *is syntactically compositional then the following lemmas are valid:*

$$\mathrm{P}'(\vec{\alpha}, \vec{\xi}) \Leftrightarrow \mathrm{P}(\vec{\alpha}, \vec{c}, \vec{\xi})$$

and

$$\exists \vec{\beta}.\ \mathrm{P}(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \star\ \mathrm{P}'(\vec{\beta}, \vec{\xi}) \Longrightarrow \mathrm{P}'(\vec{\alpha}, \vec{\xi})$$

Examples:

$$\exists \mathrm{F}, \mathrm{M}_2.\ \mathit{bsthole}(\mathrm{E}, \mathrm{M}_1, \mathrm{F}, \mathrm{M}_2) \star\ \mathit{bst}(\mathrm{F}, \mathrm{M}_2) \Longrightarrow \mathit{bst}(\mathrm{E}, \mathrm{M}_1).$$

Stronger Lemma

If P' is stronger than P and P compositional) then the following lemmas are valid:

$$P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \Longrightarrow P(\vec{\alpha}, \vec{\beta}, \vec{\xi})$$

$$\exists \vec{\beta}'.\ P'(\vec{\alpha}, \vec{\beta}', \vec{\xi}) \star P(\vec{\beta}', \vec{\beta}, \vec{\xi}) \Longrightarrow P(\vec{\alpha}, \vec{\beta}, \vec{\xi})$$

Stronger Lemma

If P' is stronger than P and P compositional) then the following lemmas are valid:

$$P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \Longrightarrow P(\vec{\alpha}, \vec{\beta}, \vec{\xi})$$

$$\exists \vec{\beta}'.\ P'(\vec{\alpha}, \vec{\beta}', \vec{\xi}) \star P(\vec{\beta}', \vec{\beta}, \vec{\xi}) \Longrightarrow P(\vec{\alpha}, \vec{\beta}, \vec{\xi})$$

Examples:

- $bsthole_{\geq 0}$ is stronger that $bsthole$

Stronger Lemma

If P' is stronger than P and P compositional) then the following lemmas are valid:

$$P'(\vec{\alpha}, \vec{\beta}, \vec{\xi}) \Longrightarrow P(\vec{\alpha}, \vec{\beta}, \vec{\xi})$$

$$\exists \vec{\beta}'.\ P'(\vec{\alpha}, \vec{\beta}', \vec{\xi}) \star P(\vec{\beta}', \vec{\beta}, \vec{\xi}) \Longrightarrow P(\vec{\alpha}, \vec{\beta}, \vec{\xi})$$

Examples:

- $bsthole_{\geq 0}$ is stronger that $bsthole$

... and other lemmas.

Outline

$$\varphi_1 \implies \exists \vec{X}.\ \varphi_2$$

- Slice φ_1 in parts entailing spatial atoms A of φ_2

$$(\varphi^A, \mathcal{C}_r^A, \mathcal{C}_g^A) \implies \exists \vec{X}.\ A$$

with $\mathcal{C}_r^A$ assertions on $\vec{X}$, $\mathcal{C}_g^A$ constraints on shared vars, such that

$$\mathrm{Pure}(\varphi_1) \wedge \mathcal{C}_r^A \implies \mathrm{Pure}(\varphi_2) \wedge \mathcal{C}_g^A$$

- check that slices are disjoint

Main procedure

1 **Procedure** slice $(\varphi_1, \exists \vec{X}.\ \varphi_2)$
2 | **if** $|\mathsf{Spatial}(\varphi_2)| > 1$ **then**
3 | | **foreach** *spatial atom* A *of* $\mathsf{Spatial}(\varphi_2)$ *different from* `emp` **do**
4 | | | $(\varphi^A, \mathcal{C}_r^A, \mathcal{C}_g^A) \leftarrow \mathsf{slice}(\varphi_1, \exists \vec{X}.\ A)$;
5 | | | **if** $\varphi^A = \bot$ **then return** $\bot$;
6 | | $\mathcal{C}_r \leftarrow \bigwedge_A \mathcal{C}_r^A$; $\mathcal{C}_g \leftarrow \bigwedge_A \mathcal{C}_g^A$;
7 | | **if** $\mathsf{Pure}(\varphi_1) \wedge \mathcal{C}_r \models \mathsf{Pure}(\varphi_2) \wedge \mathcal{C}_g$ *and* $\forall A, B.\ \varphi_A \bowtie \varphi_B$ **then**
8 | | | **return** $(\ \mathsf{Pure}(\varphi_1) \wedge \star_A \varphi_A,\ \mathcal{C}_r,\ \mathcal{C}_g\ )$
9 | **else**
10 | | $(A, \mathcal{C}_r, \mathcal{C}_g) \leftarrow \mathsf{matchAtom}(\varphi_1, \exists \vec{X}.\ \mathsf{Spatial}(\varphi_2))$;
11 | | **if** $A \neq \bot$ *and* $\mathsf{Pure}(\varphi_1) \wedge \mathcal{C}_r \models \mathsf{Pure}(\varphi_2) \wedge \mathcal{C}_g$ **then**
12 | | | **return** $(A, \mathcal{C}_r, \mathcal{C}_g)$
13 | | **else if** $A = \bot$ *and* $\mathsf{Spatial}(\varphi_2)$ *is a points-to atom* **then return** $\bot$;
14 | | **else if** $A = \bot$ *and* $\mathsf{Spatial}(\varphi_2)$ *is a predicate atom, say* $P(E, \vec{F})$ **then**
15 | | | **foreach** *lemma* $L \triangleq \exists \vec{Z}.\ \Pi \wedge \Sigma \Rightarrow P(E, \vec{F})$ **do**
16 | | | | $(A_1, \mathcal{C}_r, \mathcal{C}_g) \leftarrow \mathsf{matchAtom}(\varphi_1, \exists \vec{X} \exists \vec{Z}.\ root(L))$;
17 | | | | **if** $A_1 \neq \bot$ **then**
18 | | | | | $\exists \vec{Z}'.\Pi' \wedge \Sigma' \leftarrow \mathsf{quantElmt}(\exists \vec{X} \exists \vec{Z}.\Pi \wedge (\Sigma \setminus root(L)),\ \mathcal{C}_r\ )$;
19 | | | | | $(\varphi, \mathcal{C}_r', \mathcal{C}_g') \leftarrow \mathsf{slice}(\varphi_1 \setminus A_1, \exists \vec{Z}'.\ \Pi' \wedge \Sigma')$;
20 | | | | | **if** $\varphi \neq \bot$ *and* $\mathsf{Pure}(\varphi_1) \wedge \mathcal{C}_r \wedge \mathcal{C}_r' \wedge \Pi \models \mathsf{Pure}(\varphi_2) \wedge \mathcal{C}_g \wedge \mathcal{C}_g'$ **then**
21 | | | | | | **return** $(\ \mathsf{Pure}(\varphi_1) \wedge A_1 * \varphi,\ \mathcal{C}_r \wedge \mathcal{C}_r' \wedge \Pi,\ \mathcal{C}_g \wedge \mathcal{C}_g')$;
22 | **return** $\bot$

Main procedure – Example

$$\varphi_1 \implies \exists \vec{X}.\ \varphi_2$$

$$\varphi_1 ::= x_1 \neq \mathsf{nil} \wedge x_2 \neq \mathsf{nil} \wedge v_1 < v_2 \wedge x_1 \mapsto \{(\mathtt{next}, x_2), (\mathtt{data}, v_1)\}$$
$$\star\ x_2 \mapsto \{(\mathtt{next}, \mathsf{nil}), (\mathtt{data}, v_2)\}$$

$$\varphi_2 ::= \exists M.\ \mathit{lseg}(x_1, M, \mathsf{nil}, \emptyset) \wedge v_2 \in M$$

apply the inductive rule ...

Main procedure – Example

$$\varphi_1 \implies \exists \vec{X}.\ \varphi_2$$

$$\varphi_1' ::= x_1 \neq \mathsf{nil} \wedge x_2 \neq \mathsf{nil} \wedge v_1 < v_2 \wedge x_2 \mapsto \{(\mathtt{next}, \mathsf{nil}), (\mathtt{data}, v_2)\}$$

$$\varphi_2' ::= \exists M_1.\ \mathit{lseg}(x_2, M_1, \mathsf{nil}, \emptyset) \wedge v_1 \leq M_1$$

apply the inductive rule ...

Main procedure – Example

$$\varphi_1 \implies \exists \vec{X}.\ \varphi_2$$

$$\varphi_1'' ::= x_1 \neq \mathsf{nil} \wedge x_2 \neq \mathsf{nil} \wedge \mathtt{emp}$$
$$\varphi_2'' ::= \exists M_1''.\ \mathit{lseg}(\mathsf{nil}, M_1, \mathsf{nil}, \emptyset) \wedge v_2 \leq M_1''$$

apply the basic rule and check data constraints on the return path!

Outline

SPEN solver

- Extension of SMTLIB theory for SL with Int and BagInt
- Generate lemma depending on ID
- Saturate the two formulas $\longrightarrow$ data constraints
- Apply proof strategy
- Provide diagnosis
- Rely on Z3 for checking data constraints

Experiments

Data structure	Procedure	#VC	Lemma (#b, #r, #p, #c, #d)	$\Longrightarrow_{\mathbb{D}}$	Time (s)	
					Spen	Z3
sorted lists	`search`	4	(6, 8, 17, 3, 1)	6	0.10	0.05
	`insert`	8	(12, 18, 30, 12, 0)	20	0.33	0.10
	`delete`	4	(6, 10, 16, 6, 1)	10	0.15	0.05
BST	`search`	4	(9, 11, 27, 9, 1)	6	0.20	0.05
	`insert`	14	(18, 21, 33, 14, 0)	24	0.63	0.20
	`delete`	25	(30, 37, 106, 25, 0)	68	1.49	0.51
AVL	`search`	4	(10, 12, 17, 13, 1)	6	0.23	0.15
	`insert`	22	(20, 27, 66, 23, 0)	48	1.94	0.63
RBT	`search`	4	(9, 12, 27, 12, 1)	6	0.23	0.15
	`insert`	21	(32, 25, 126, 22, 10)	78	2.94	0.93

#b: base rules, #r: recursive rules, #p: exact matching,
#c: composition lemma, #d: other lemma
$\Longrightarrow_{\mathbb{D}}$: data entailments generated

Outline

Related Works

For shape only:

- CYCLIST [CADE'11]
- SLIDE [ATVA'14]
- SPEN [APLAS'14]

For singly linked lists:

- SLAD [ATVA'12] for SL + ArrayLogic
- Asterix [PLDI'11,APLAS'13] for SLL + length

For complex data structures:

- Sleek [NUS] $\longrightarrow$ ad-hoc?
- DRYAD [PLDI'12] $\longrightarrow$ no lemma generation