Functional Correctness of Pointer Programs

Ewen Maclean

eahm2@macs.hw.ac.uk

Dependable Systems Group

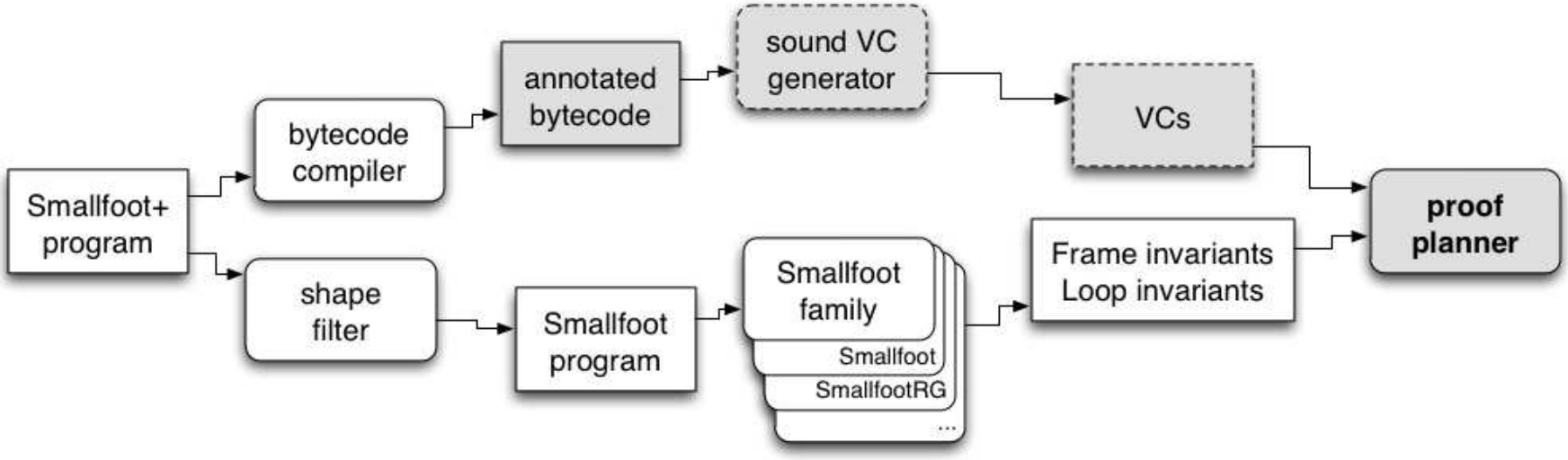
Mathematical And Computer Sciences

Heriot-Watt University

Motivation

- Pointer Programs
- Shape and Functional Properties
- Inductive Data Structures on the Heap
- Exploit Separation Logic
- Combine “Proof-planning” and Program Analysis

Cooperative Reasoning



- Animation of pointer programs
- Exploitation of existing Theorem Proving Systems

Example Pointer Program

```
{P}
j := nil
{R}
while !(i == nil) do
  k := i.2;
  i.2 := j;
  j := i;
  i := k;
done
{Q}
```

$P: \; list(\alpha_0, i, nil)$

$Q: \; list(rev(\alpha_0), j, nil)$

$R: \; \exists \alpha, \beta. list(\alpha, i, nil) * list(\beta, j, nil) \wedge \alpha_0 = rev(\alpha) <> \beta$

Functional Correctness

- Interested in data
- Loop-invariant now hard
- Existential elimination needed
- Can extract a *pure* part
- Can exploit Smallfoot to guide the proof
- Meta-variable instantiation via Isabelle

Proof

- Pre-loop verification

$$list(\alpha_0, i, nil) \vdash \exists \alpha, \beta.\ list(\alpha, i, nil) * list(\beta, nil, nil) \wedge \alpha_0 = rev(\beta) <> \alpha$$

- Loop-invariant verification

$$\exists \alpha', \beta'. list(\alpha', i, nil) * list(\beta', j, nil) \ \wedge \ \alpha_0 = rev(\beta') <> \alpha' \ \wedge \ \neg(i = nil) \ \vdash$$

$$\exists \alpha, \beta, x_2. (\exists x, y.\ (i \mapsto x, y) * ((i \mapsto x, j) \mathrel{-\!\!*} list(\alpha, x_2, nil) * list(\beta, i, nil))) \wedge$$

$$\alpha_0 = rev(\beta) <> \alpha \ \wedge \ \exists x_1.\ i \hookrightarrow x_1, x_2$$

- Post-loop verification

$$\exists \alpha', \beta'.\ list(\alpha', i, nil) * list(\beta', j, nil) \ \wedge \ \alpha_0 = rev(\beta') <> \alpha' \ \wedge \ i = nil \ \vdash$$

$$list(rev(\alpha_0), j, nil)$$

Methods

$$A \mathrel{-\!\!*} B$$

- Split up B using lseg method

$$lseg(h :: t, i, j) \Rightarrow \exists k.i \rightarrow h, k * lseg(t, k, j)$$

- Use rules

$$\begin{array}{lll} pure(A) \rightarrow & A \wedge (B * C) & \Rightarrow \quad B * (A \wedge C) \\ pure(A) \rightarrow & A \rightarrow (B * C) & \Rightarrow \quad B * (A \rightarrow C) \end{array}$$

- Record positions
- Generate tactic using AC rules:

$$\begin{array}{lll} A * B & \Longleftrightarrow & B * A \\ A * (B * C) & \Longleftrightarrow & (A * B) * C \end{array}$$

Mutation with $x \overset{next}{\mapsto} y \mathrel{-\!\!*} \ldots$

score= 10 $x \overset{next}{\mapsto} y$

This case represents an exact ground match.

score= 6 $x \overset{next}{\mapsto} \mathcal{F}_1$

This case represents an inexact match, and instantiates the $\mathcal{F}_1$ with y.

score= 3 $data\_lseg(\alpha, x, \mathcal{F}_1)$

This case represents a match with the first cons of the linked list.

score= 3 $data\_lseg(\alpha, \mathcal{F}_1, y)$

This case represents a match with the last cons of the linked list.

score= 1 $data\_lseg(\alpha, \mathcal{F}_1, \mathcal{F}_2)$

This case represents a match with the a cons somewhere embedded within the linked list.

Functional Residue

$$\frac{(l1 \neq null \wedge ((([l1 \overset{data}{\mapsto} \mathcal{X}_4] * ([l1 \overset{next}{\mapsto} \mathcal{X}_5] * lseg(\mathcal{X}_3, \mathcal{X}_5, null)) \wedge cons(\mathcal{X}_4, \mathcal{X}_3) = \mathcal{X}_1 * (lseg(\mathcal{X}_2, l0, null) \wedge \mathcal{X}_a = append(reverse(\mathcal{X}_1), \mathcal{X}_2))}{([l1 \overset{next}{\mapsto} \mathcal{F}_1] * (lseg(\mathcal{F}_5, l0, null) * ([l1 \overset{data}{\mapsto} \mathcal{F}_6] * (cons(\mathcal{F}_6, \mathcal{F}_5) = \mathcal{F}_3 \wedge (\mathcal{X}_a = append(reverse(\mathcal{F}_2), \mathcal{F}_3) \wedge lseg(\mathcal{F}_2, \mathcal{F}_1, null)}$$

Instantations Calculated:

$$\{\mathcal{X}_4/\mathcal{F}_6, \mathcal{X}_5/\mathcal{F}_1, \mathcal{X}_2/\mathcal{F}_5, \mathcal{X}_3/\mathcal{F}_2, \mathcal{X}_5/\mathcal{F}_1, \mathcal{X}_2/\mathcal{F}_5\}$$

Functional Residue:

$$\frac{l1 \neq null \wedge cons(\mathcal{X}_4, \mathcal{X}_3) = \mathcal{X}_1 \wedge \mathcal{X}_a = append(reverse(\mathcal{X}_1), \mathcal{X}_2)}{cons(\mathcal{X}_4, \mathcal{X}_3) = \mathcal{F}_3 \wedge \mathcal{X}_a = append(reverse(\mathcal{X}_3), \mathcal{F}_3)}$$

Simplified Version:

$$l1 \neq null \vdash append(reverse(cons(\mathcal{X}_4, \mathcal{X}_3)), \mathcal{X}_2) = append(reverse(\mathcal{X}_3), cons(\mathcal{X}_4, \mathcal{X}_2))$$

Example Pointer Program Revisited

```
{P}
j := nil
{R}
while !(i == nil) do
  k := i.2;
  i.2 := j;
  j := i;
  i := k;
done
{Q}
```

$P: \; list(\alpha_0, i, nil)$

$Q: \; list(rev(\alpha_0), j, nil)$

$R: \; \exists \alpha, \beta. list(\alpha, i, nil) * list(\beta, j, nil) \wedge P(\alpha_0, \beta, \alpha)$

Invariant Generation

Functional Residue:

$$\frac{l1 \neq null \wedge P[\mathcal{X}_a, cons(\mathcal{X}_4, \mathcal{X}_3), \mathcal{X}_2]}{P[\mathcal{X}_a, \mathcal{X}_3, cons(\mathcal{X}_4, \mathcal{X}_2)]}$$

Term Synthesis:

- Parse Functional Residue
- Guess equality
- Synthesise Terms
  - Theory File – e.g. lists
  - Depth Limit
- With equality
  - Counter-example checker
  - Many branches in which P is shared

Example Term Synthesis

$$P \equiv \lambda x, y, z.\ x = append(rev(y), z)$$

We choose equality so that we can use a counter example checker to rule out incorrect conjectures such as

$$P \equiv \lambda x, y, z.\ x = append(z, y)$$

By populating the $\mathcal{X}_i$ by

$$\mathcal{X}_4 = 0 \qquad \mathcal{X}_3 = [1, 2, 3] \qquad \mathcal{X}_2 = [4, 5, 6]$$

we see that the latter suggestion is evaluated as

$$[4, 5, 6, 0, 1, 2, 3] = [0, 4, 5, 6, 1, 2, 3]$$

which can be ruled out.