A Cooperative Approach to Loop Invariant Discovery for Pointer Programs

Andrew Ireland

Dependable Systems Group
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh

Motivations & Overview

Combining the strengths of individual techniques, while compensating for each other's weaknesses.

- **Cooperation** between program analysis and deductive reasoning – *beyond generic properties.*
- **Separation logic** facilitates modular reasoning for pointer programs – *program analysis tools emerging.*
- **Proof planning** facilitates cooperative reasoning and proof patching – *automates the discovery of generalizations, lemmas, induction revisions, case splits and loop invariants.*

Proposed Approach

Loop invariant = shape + content

1. Discover the **shape** via **symbolic evaluation**.
2. Specify the loop invariant by combining **shape** and **content** (schematic).
3. Instantiate and verify the loop invariant via **proof planning**.
4. Patch symbolic evaluation failures via **proof patching**.

Note: shape invariant discovery mechanism is treated as an oracle.

Separation Logic

Peter O'Hearn (QMU) & John Reynolds (CMU)

- An extension to Hoare logic developed with the aim of simplifying pointer program proofs.
- Also supports proof where concurrent processes share resources.
- Focuses the proof effort on the parts of the heap that are relevant to a program, so called **local reasoning**.
- Tools: SMALLFOOT, SPACE INVADER, SLAYER, ...

Modelling the Heap

Empty heap: the assertion *emp* holds for a heap that contains no cells.

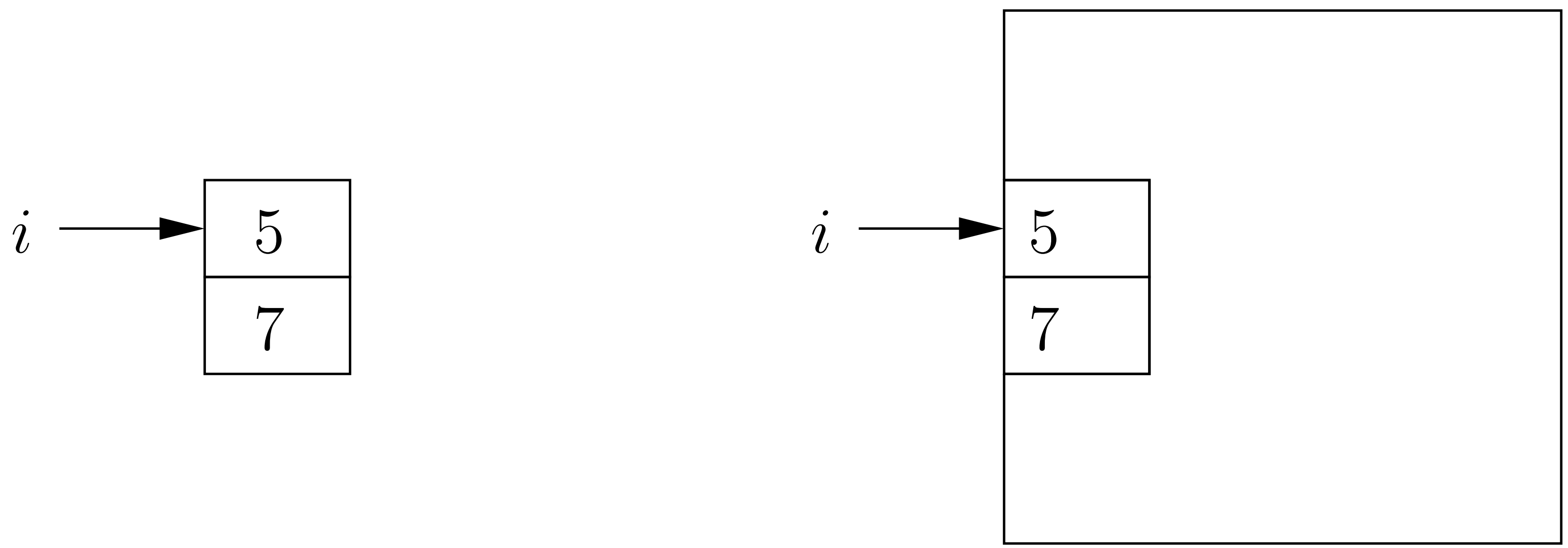
Singleton heap: the assertion $X \mapsto E$ holds for a heap that contains a single cell (maps-to relation), *e.g.*

$$i \longrightarrow \boxed{5}$$

$$(i \mapsto 5)$$

Separating Conjunction

$P * Q$ holds for a heap if the heap can be divided into two disjoint heaps H_1 and H_2, such that P holds in H_1 and Q holds in H_2, *e.g.*

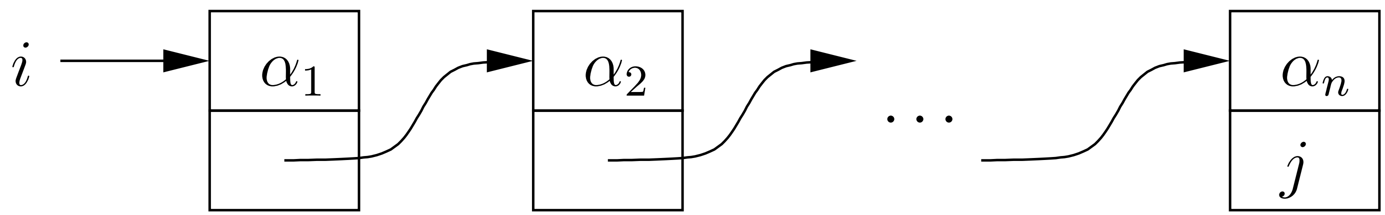


$(i \mapsto 5) * (i + 1 \mapsto 7)$ $\qquad$ $(i \mapsto 5) * (i + 1 \mapsto 7) * true$

$(i \mapsto 5, 7)$ $\qquad$ $(i \hookrightarrow 5, 7)$

Singly-linked Lists

$$\begin{aligned} \mathit{list}([], Y, Z) &\leftrightarrow \mathit{emp} \wedge Y = Z \\ \mathit{list}([W|X], Y, Z) &\leftrightarrow (\exists p.\ (Y \mapsto W, p) * \mathit{list}(X, p, Z)) \end{aligned}$$



$$\mathit{list}(\alpha, i, j)$$

where $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_n]$

Sequence Concatenation & Reversal

$$
\begin{aligned}
app([], Z) &= Z \\
app([X|Y], Z) &= [X|app(Y, Z)]
\end{aligned}
$$

$$
\begin{aligned}
rev([]) &= [] \\
rev([X|Y]) &= app(rev(Y), [X])
\end{aligned}
$$

List Reversal: Program & Specification

$\{P\}$
```
j := nil;
```
$\{R\}$
```
while not(i = nil) do
  k := [i+1]; [i+1] := j; j := i; i := k
od
```
$\{Q\}$

$$P: \quad (\exists \alpha.\, list(\alpha, i, nil) \;\wedge \alpha_{init} = \alpha)$$

$$Q: \quad (\exists \beta.\, list(\beta, j, nil) \;\wedge rev(\alpha_{init}) = \beta)$$

$$R: \quad (\exists \alpha, \beta.\; \underbrace{list(\alpha, i, nil) * list(\beta, j, nil)}_{\textbf{shape}} \wedge \underbrace{rev(\alpha_{init}) = app(rev(\alpha), \beta)}_{\textbf{content}})$$

Shape via Symbolic Evaluation

Stephen Magill, Ed Clarke, Peter Lee (CMU) &
Aleksander Nanevski (Harvard)

- Shape invariants generated via symbolic evaluation within separation logic.
- Loop code is evaluated repeatedly, after each iteration a weakening step is applied if possible, *i.e.* a **fold**.
- Termination occurs if symbolic evaluation converges to a fixed point – *convergence not guaranteed.*
- Note: content dealt with via predicate abstraction, while we adopt a theorem proving approach.

Annotated Programs

$$\{H \wedge P\}\ C$$

- H describes the shape of the heap, *i.e.*

$$(x \mapsto y, z)$$

$$ls(p_1, p_2) \equiv (\exists x, k.\ p_1 \mapsto (x, k) * ls(k, p_2)) \vee (p_1 = p_2 \wedge \mathit{emp})$$

$$ls^+(p_1, p_2) \equiv (\exists x, k.\ p_1 \mapsto (x, k) * ls(k, p_2))$$

- P denotes facts about the stack variables.
- C denotes the program code.

Memory Descriptions

$$(H'; S; P')$$

- S denotes a list of equalities, *i.e.* $x = v$, where v is a new symbolic variable, corresponding to the program variable x.
- H' and P' are generated from H and P by replacing all occurrences of x by the corresponding v respectively.

Symbolic Evaluation

$$(ls(v_1, nil); i = v_1, \underline{j = \_}, k = \_;)$$

`j:=nil`

$$(ls(v_1, nil); i = v_1, j = nil, k = \_;)$$

enter loop

$$(\underline{ls(v_1, nil)}; i = v_1, j = nil, k = \_; \neg(v_1 = nil))$$

unfold *ls*

$$(v_1 \mapsto (v_2, v_3) * ls(v_3, nil); i = v_1, j = nil, \underline{k = \_}; \neg(v_1 = nil))$$

`k:=[i+1]`

$$(v_1 \mapsto (v_2, v_3) * ls(v_3, nil); i = v_1, j = nil, k = v_3; \neg(v_1 = nil))$$

$$\vdots$$

Symbolic Evaluation

$$\vdots$$

$$(\underline{v_3 \mapsto (v_4, v_1) * v_1 \mapsto (v_2, nil)} * ls(v_5, nil); i = v_5, j = v_3, k = v_5;$$
$$\neg(v_3 = nil) \wedge \neg(v_1 = nil))$$

fold (1)

$$(ls^+(v_3, nil) * ls(v_5, nil); i = v_5, j = v_3, k = v_5;$$
$$\neg(v_3 = nil) \wedge \neg(v_1 = nil))$$

$$\begin{array}{rcl} P \mapsto (I, K) * ls(K, Q) & \Rightarrow & ls^+(P, Q) \\ ls(P, K) * K \mapsto (I, Q) & \Rightarrow & ls^+(P, Q) \\ P \mapsto (I, K) * K \mapsto (J, Q) & \Rightarrow & ls^+(P, Q) \qquad (1) \end{array}$$

where K is not associated with a program variable.

Convergence

- First iteration:

$$(v_1 \mapsto (v_2, nil) * ls(v_3, nil); i = v_3, j = v_1, k = v_3; \\ \neg(v_1 = nil))$$

- Second iteration:

$$(ls^+(v3, nil) * ls(v_5, nil); i = v_5, j = v_3, k = v_5; \\ \neg(v_3 = nil) \wedge \neg(v_1 = nil))$$

- Third iteration:

$$(ls^+(v5, nil) * ls(v_7, nil); i = v_7, j = v_5, k = v_7; \\ \neg(v_5 = nil) \wedge \neg(v_3 = nil) \wedge \neg(v_1 = nil))$$

Note: loop invariant is denoted by the disjunction of memory descriptions leading up to convergence.

Shape Invariant

- Symbolic evaluation:

$$
\begin{gathered}
(ls(i, nil) \wedge j = nil) \ \vee \\
(ls(i, nil) * (j \mapsto (\_, nil))) \ \vee \\
(ls(i, nil) * ls^{+}(j, nil))
\end{gathered}
$$

- Hand-crafted:

$$list(\_, i, nil) * list(\_, j, nil)$$

Proof Planning

- **Proof plans:** automation via high-level proof outlines.
- **Middle-out reasoning:** use of meta-variables in delaying choice during proof planning – *similar to Lazy Thinking*.
- **Proof critics:** automatic proof patching via proof-failure analysis, *e.g.* invariant & lemma discovery.
- **Cooperative reasoning:**
  - Clam/HOL: proof by mathematical induction.
  - SPADEase: program analysis/proof planning for SPARK.

Content via Middle-Out Proof Planning

- Postcondition:

$$(\exists \beta .\ list(\beta, j, nil)\ \wedge rev(\alpha_{init}) = \beta)$$

- Shape invariant:

$$list(\_, i, nil) * list(\_, j, nil)$$

- Schematic loop invariant:

$$(\exists \alpha, \beta .\ \underbrace{list(\alpha, i, nil) * list(\beta, j, nil)}_{\textbf{shape}} \wedge \underbrace{rev(\alpha_{init}) = F_1(\beta, \alpha)}_{\textbf{content}})$$

Schematic Verification Condition

Given:

$$\ldots\ (\exists \alpha', \beta'.\ \ldots \wedge\ rev(\alpha_{init}) = F_1(\beta', \alpha'))\ \ldots$$

Goal:

$$\boxed{(\ \ldots\ (\exists\, b.\ \ldots\ \boxed{\exists \alpha, \boxed{\beta_{tl}}.\ \ldots \wedge\ rev(\alpha_{init}) = F_1(\ \boxed{[b|\boxed{\beta_{tl}}]}^{\uparrow}, \lfloor \alpha \rfloor)}\ )\ \ldots\ )}^{\uparrow}$$

Wave-rules:

$$app(X, \boxed{[Y|\underline{Z}]}^{\uparrow}) \ \Rightarrow\ app(\boxed{app(\underline{X}, [Y])}^{\downarrow}, Z)$$

$$\boxed{app(\underline{rev(Y)}, [X])}^{\downarrow} \ \Rightarrow\ rev(\boxed{[X|\underline{Y}]}^{\downarrow})$$

Middle-Out Rippling

$$\ldots \wedge\ rev(\alpha_{init}) \;=\; F_1(\boxed{[b|\boxed{\beta_{tl}}]}^{\uparrow}, \lfloor\alpha\rfloor)\ \ldots$$

$$\ldots \wedge\ rev(\alpha_{init}) \;=\; app(\boxed{app(\boxed{F_2(\boxed{[b|\boxed{\beta_{tl}}]}^{\uparrow}, \lfloor\alpha\rfloor)}, [b])}^{\downarrow}, \beta_{tl})\ \ldots$$

$$\ldots \wedge\ rev(\alpha_{init}) \;=\; app(rev(\boxed{[b|\boxed{F_3(\boxed{[b|\boxed{\beta_{tl}}]}^{\uparrow}, \lfloor\alpha\rfloor)}]}^{\downarrow}), \beta_{tl})\ \ldots$$

$$\ldots \wedge\ rev(\alpha_{init}) \;=\; app(rev(\left\lfloor \boxed{[b|\boxed{\alpha}]}^{\downarrow} \right\rfloor), \beta_{tl})\ \ldots$$

$$F_1 \;=\; \lambda x.\lambda y.\ app(rev(y), x)$$

Cooperation: Strengths and Weaknesses?

- Discovery of the shape invariant is achieved via **symbolic evaluation**.
- Verification of the shape invariant is achieved via **proof planning**.
- Discovery and verification of the content invariant is achieved via **middle-out reasoning**.
- **Patching** divergence via discovery of missing rules for folding (lemmas).

Future Work

- Extend current proof plans to deal with pointer references and existential sinks.
- Bridge the gap between machine generated and hand-crafted shape invariants.
- Build upon:
  - CMU method, SMALLFOOT or SPACE INVADER.
  - VERISOFT verification environment for sequential imperative programs (ISABELLE based).
  - ISAPLANNER proof planner (ISABELLE based).

Conclusion

Proposal: A cooperative approach to automatic loop invariant discovery for separation logic.

Hypothesis: Adopting a cooperative approach will deliver significant benefits in terms of search control.

Evidence: prototyping underway ...