Tutorial: How to Cook a Static Analyzer

or, The Surprising Effectiveness of Substructural Proof Theory

Peter O'Hearn

Queen Mary, University of London

HCSS Conference, 21 May, 2009

People

This is about work by the

SpaceInvader Team (Ldn): Cristiano Calcagno, Dino Distefano, Peter O'Hearn, Hongseok Yang

benefitting from lots of collaboration with our
SLAyer Colleagues (MSR): Josh Berdine, Byron Cook

A Substructural Logic

$$A \;\not\vdash\; A * A$$

$$10 \mapsto 3 \;\not\vdash\; 10 \mapsto 3 * 10 \mapsto 3$$

$$A * B \;\not\vdash\; A$$

$$10 \mapsto 3 * 42 \mapsto 5 \;\not\vdash\; 10 \mapsto 3$$

Program Verification Extremes

- **Extreme 1: Interactive Proof.** HOL, Coq, Isabelle... More human effort, more expressive.
- **Extreme 2: Static Analysis.** Less expressive, more automatic.
- And there is population in between (SPARK-Ada, Spec-#...).

Program Verification Extremes

- **Extreme 1: Interactive Proof.** HOL, Coq, Isabelle... More human effort, more expressive.
  - *Several embeddings of separation logic in proof assistants.*
- **Extreme 2: Static Analysis.** Less expressive, more automatic.
  - *Several prototype program analysis tools: SpaceInvader, SLAyer, THOR, Xisa...*
- And there is population in between (SPARK-Ada, Spec-#...).
  - *Smallfoot, SmallfootRG, jStar*

Verification by Static Analysis, Current Context

- 2000's: impressive practical advances in automatic program verification E.g.
  - SLAM: Protocol properties of procedure calls in device drivers, e.g. any call to `ReleaseSpinLock` is preceded by a call to `AquireSpinLock`
  - ASTRÉE: no run-time errors in Airbus code

Verification by Static Analysis, Current Context

- 2000's: impressive practical advances in automatic program verification E.g.
  - SLAM: Protocol properties of procedure calls in device drivers, e.g. any call to `ReleaseSpinLock` is preceded by a call to `AquireSpinLock`
  - ASTRÉE: no run-time errors in Airbus code
- The Missing Link
  - ASTRÉE assumes: no dynamic pointer allocation
  - SLAM assumes: memory safety
  - Wither automatic heap verification? (for substantial programs)

Verification by Static Analysis, Current Context

- 2000's: impressive practical advances in automatic program verification E.g.
  - SLAM: Protocol properties of procedure calls in device drivers, e.g. any call to `ReleaseSpinLock` is preceded by a call to `AquireSpinLock`
  - ASTRÉE: no run-time errors in Airbus code
- The Missing Link
  - ASTRÉE assumes: no dynamic pointer allocation
  - SLAM assumes: memory safety
  - Wither automatic heap verification? (for substantial programs)
- Many important programs make serious use of heap: Linux, Apache, TCP/IP, IOS... but heap verification is hard.

Part I

Symbolic Execution and Verification

Symbolic Heaps (say less, to do more..)

A special form[1]

$$(B_1 \wedge \cdots \wedge B_n) \wedge (H_1 * \cdots * H_m)$$

where

$$\begin{array}{lcl} H & ::= & E \mapsto \rho \mid \mathsf{tree}(E) \mid \mathsf{lseg}(E, E) \\ B & ::= & E{=}E \mid E{\neq}E \end{array}$$

$$E ::= x \mid \mathsf{nil}$$

$$\rho ::= f_1 : E_1, \ldots, f_n : E_n$$

$$B ::= E{=}E \mid E{\neq}E$$

[1] assertional if-then-else as well

Verification = Symbolic Execution + Entailment Checking

- Inductive Definitions unrolled only on demand (on heap access) during execution.
- Rolled up only after execution, during entailment checking

- The tree definition

$$\mathsf{tree}(E) \iff \textit{if } E{=}\mathsf{nil} \textit{ then } \mathsf{emp}$$
$$\textit{else } \exists x, y.\, (E{\mapsto}l : x, r : y) * \mathsf{tree}(x) * \mathsf{tree}(y)$$

Copytree Verification

Just inside the if (where $p \neq \texttt{nil}$)...

$\{p \neq \texttt{nil} \wedge \texttt{tree}(p)\}$

$$\mathsf{tree}(E) \iff \textit{if } E = \mathsf{nil} \textit{ then } \mathsf{emp}$$
$$\textit{else } \exists x, y.\, (E \mapsto l: x, r: y) * \mathsf{tree}(x) * \mathsf{tree}(y)$$

Copytree Verification

Just inside the if (where $p \neq \texttt{nil}$)...

$\{p \neq \texttt{nil} \wedge \texttt{tree}(p)\}$ unroll it...

$$\mathsf{tree}(E) \iff \textit{if } E{=}\mathsf{nil} \textit{ then } \mathsf{emp}$$
$$\textit{else } \exists x, y.\, (E \mapsto l : x, r : y) * \mathsf{tree}(x) * \mathsf{tree}(y)$$

Copytree Verification

Just inside the if (where $p \neq \texttt{nil}$)...

$\{p \neq \texttt{nil} \wedge \texttt{tree}(p)\}$ unroll it...

$\{p \mapsto [l : x, r : y] * \texttt{tree}(x) * \texttt{tree}(y)\}$

$$\mathsf{tree}(E) \iff \textit{if } E = \mathsf{nil} \textit{ then } \mathsf{emp}$$
$$\textit{else } \exists x, y.\, (E \mapsto l : x, r : y) * \mathsf{tree}(x) * \mathsf{tree}(y)$$

Copytree Verification

Just inside the if (where $p \neq \texttt{nil}$)...

$\{p \neq \texttt{nil} \land \texttt{tree}(p)\}$ unroll it...

$\{p \mapsto [l : x, r : y] * \texttt{tree}(x) * \texttt{tree}(y)\}$

$i := p \to l\ ;\ j := p \to r;$

$\{p \mapsto [l : i, r : j] * \texttt{tree}(i) * \texttt{tree}(j)\}$

$\mathsf{tree}(E) \iff$ *if* $E = \mathsf{nil}$ *then* emp

else $\exists x, y.\ (E \mapsto l : x, r : y) * \mathsf{tree}(x) * \mathsf{tree}(y)$

Copytree Verification

Just inside the if (where $p \neq \texttt{nil}$)...

$\{p \neq \texttt{nil} \wedge \texttt{tree}(p)\}$ unroll it...

$\{p \mapsto [l : x, r : y] * \texttt{tree}(x) * \texttt{tree}(y)\}$

$i := p \to l \; ; \; j := p \to r;$

$\{p \mapsto [l : i, r : j] * \texttt{tree}(i) * \texttt{tree}(j)\}$

$\texttt{tree\_copy}(ii \; ; \; i) \; ; \; \texttt{tree\_copy}(jj \; ; \; j)$

$s := \texttt{new}() \; ; \; s \to l := ii \; ; \; s \to r := jj;$

$\text{tree}(E) \iff$ *if* $E = \text{nil}$ *then* emp

else $\exists x, y. \, (E \mapsto l : x, r : y) * \text{tree}(x) * \text{tree}(y)$

Copytree Verification

Just inside the if (where $p \neq \texttt{nil}$)...

$\{p \neq \texttt{nil} \wedge \texttt{tree}(p)\}$ unroll it...

$\{p \mapsto [l : x, r : y] * \texttt{tree}(x) * \texttt{tree}(y)\}$

$i := p \rightarrow l\ ;\ j := p \rightarrow r;$

$\{p \mapsto [l : i, r : j] * \texttt{tree}(i) * \texttt{tree}(j)\}$

$\texttt{tree\_copy}(ii\ ;\ i)\ ;\ \texttt{tree\_copy}(jj\ ;\ j)$

$s := \texttt{new}()\ ;\ s \rightarrow l := ii\ ;\ s \rightarrow r := jj;$

$\{p \mapsto [l : i, r : j] * \texttt{tree}(i) * \texttt{tree}(j) * s \mapsto [l : ii, r : jj] * \texttt{tree}(ii) * \texttt{tree}(jj)\}$

$$\text{tree}(E) \iff \textit{if } E = \text{nil } \textit{then } \text{emp}$$
$$\textit{else } \exists x, y.\ (E \mapsto l : x, r : y) * \text{tree}(x) * \text{tree}(y)$$

Copytree Verification

We are left with an entailment

$$p \mapsto [l : i, r : j] * \texttt{tree}(i) * \texttt{tree}(j) * s \mapsto [l : ii, r : jj] * \texttt{tree}(ii) * \texttt{tree}(jj)$$

$$\vdash \qquad \texttt{tree}(p) * \texttt{tree}(s)$$

$$\text{tree}(E) \iff \textit{if } E = \text{nil } \textit{then } \text{emp}$$
$$\textit{else } \exists x, y.\, (E \mapsto l : x, r : y) * \text{tree}(x) * \text{tree}(y)$$

Copytree Verification

We are left with an entailment

$$p \mapsto [l : i, r : j] * \texttt{tree}(i) * \texttt{tree}(j) * s \mapsto [l : ii, r : jj] * \texttt{tree}(ii) * \texttt{tree}(jj)$$

$$\vdash \qquad \texttt{tree}(p) * \texttt{tree}(s)$$

let me roll it...

$$\text{tree}(E) \iff \textit{if } E = \text{nil } \textit{then } \text{emp}$$
$$\textit{else } \exists x, y.\, (E \mapsto l : x, r : y) * \text{tree}(x) * \text{tree}(y)$$

Flawed Copytree Failed Verification

When we mistakenly point back into the source tree

we are left with an entailment

$$p \mapsto [l : i, r : j] * \mathtt{tree}(i) * \mathtt{tree}(j) * s \mapsto [l : i, r : j] * \mathtt{tree}(ii) * \mathtt{tree}(jj)$$

$$\vdash \qquad \mathtt{tree}(p) * \mathtt{tree}(s)$$

that we can't roll up...

Part II

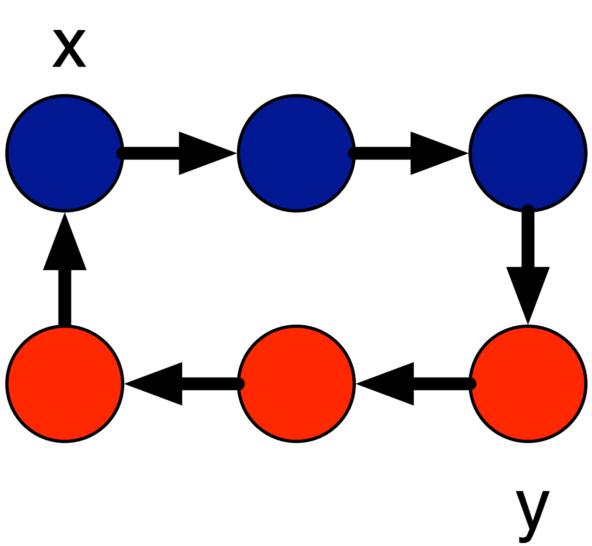
Proving Entailments

Induction and Linked Lists

List segments   ($\mathsf{list}(E)$ is shorthand for $\mathsf{lseg}(E, \mathsf{nil})$)

$$\mathsf{lseg}(E, F) \iff \texttt{if } E = F \texttt{ then } \mathsf{emp}$$
$$\texttt{else } \exists y. E \mapsto tl : y * \mathsf{lseg}(y, F)$$

$$\mathsf{lseg}(x, y) * \mathsf{lseg}(y, x)$$

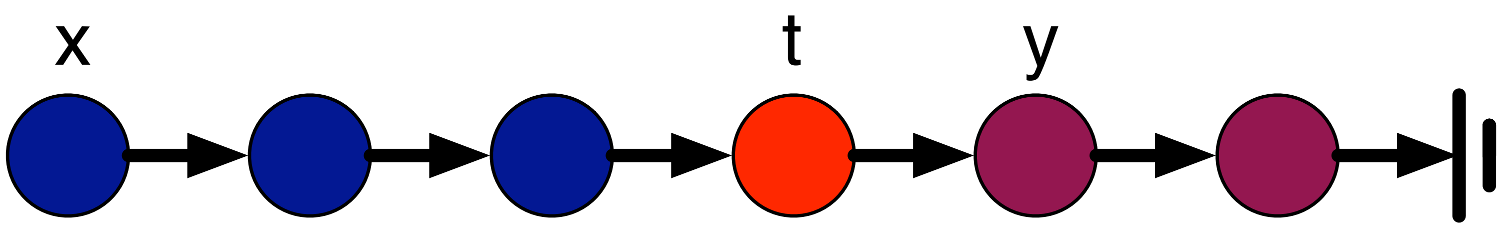


Induction and Linked Lists

List segments ($\mathsf{list}(E)$ is shorthand for $\mathsf{lseg}(E, \mathsf{nil})$)

$$\mathsf{lseg}(E, F) \iff \texttt{if } E = F \texttt{ then } \mathsf{emp}$$
$$\texttt{else } \exists y. E \mapsto tl : y * \mathsf{lseg}(y, F)$$

$$\mathsf{lseg}(x, t) * t \mapsto [tl : y] * \mathsf{list}(y)$$

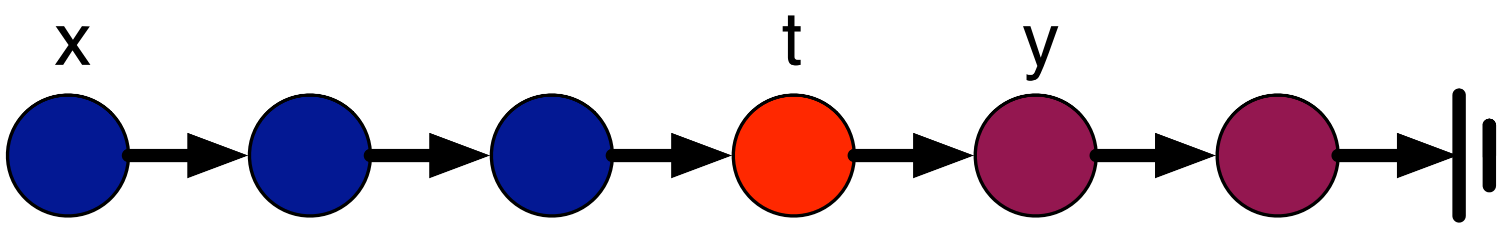


Induction and Linked Lists

List segments (list(E) is shorthand for lseg(E, nil))

$$\text{lseg}(E,F) \iff \texttt{if } E = F \texttt{ then emp}$$
$$\texttt{else } \exists y. E \mapsto tl : y * \text{lseg}(y, F)$$

Entailment $\text{lseg}(x,t) * t \mapsto [tl : y] * \text{list}(y) \vdash \text{list}(x)$

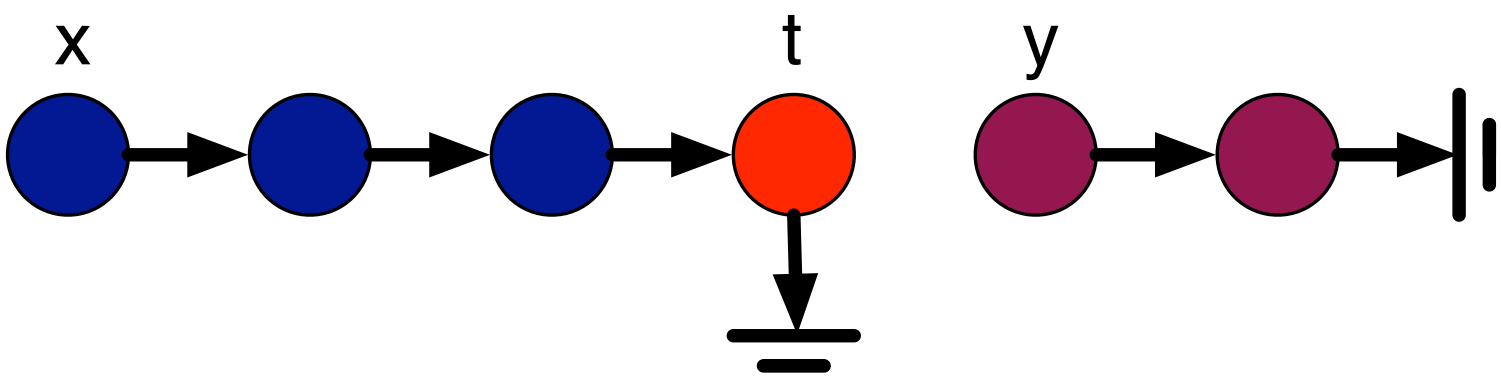


Induction and Linked Lists

List segments ($\mathsf{list}(E)$ is shorthand for $\mathsf{lseg}(E, \mathsf{nil})$)

$$\mathsf{lseg}(E, F) \iff \texttt{if } E = F \texttt{ then } \mathsf{emp}$$
$$\texttt{else } \exists y. E \mapsto tl : y * \mathsf{lseg}(y, F)$$

Non-Entailment $\mathsf{lseg}(x, t) * t \mapsto \texttt{nil} * \mathsf{list}(y) \not\vdash \mathsf{list}(x)$



Solution (Berdine and Calcagno)

- A proof theory oriented around Abstraction and Subtraction.

Solution (Berdine and Calcagno)

- A proof theory oriented around Abstraction and Subtraction.
- Sample Abstraction Rule

$$\mathsf{lseg}(x, t) * \mathsf{list}(t) \vdash \mathsf{list}(x)$$

Solution (Berdine and Calcagno)

- A proof theory oriented around Abstraction and Subtraction.
- Sample Abstraction Rule

$$\mathsf{lseg}(x, t) * \mathsf{list}(t) \ \vdash \ \mathsf{list}(x)$$

- Subtraction Rule

$$\frac{Q_1 \vdash Q_2}{Q_1 * S \vdash Q_2 * S}$$

Solution (Berdine and Calcagno)

- A proof theory oriented around Abstraction and Subtraction.
- Sample Abstraction Rule

$$\mathsf{lseg}(x, t) * \mathsf{list}(t) \vdash \mathsf{list}(x)$$

- Subtraction Rule

$$\frac{Q_1 \vdash Q_2}{Q_1 * S \vdash Q_2 * S}$$

- Try to reduce an entailment to the axiom

$$\frac{}{B \wedge \mathsf{emp} \vdash \mathsf{true} \wedge \mathsf{emp}}$$

Works great!

$$\mathsf{lseg}(x, t) * t \mapsto [tl : y] * \mathsf{list}(y) \;\vdash\; \mathsf{list}(x) \qquad\qquad \text{Abstract (Roll)}$$

Works great!

$$\mathsf{lseg}(x, t) * \mathsf{list}(t) \vdash \mathsf{list}(x) \qquad \text{Abstract (Inductive)}$$

$$\mathsf{lseg}(x, t) * t \mapsto [tl : y] * \mathsf{list}(y) \vdash \mathsf{list}(x) \qquad \text{Abstract (Roll)}$$

Works great!

$\mathsf{list}(x) \;\vdash\; \mathsf{list}(x)$	Subtract
$\mathsf{lseg}(x, t) * \mathsf{list}(t) \;\vdash\; \mathsf{list}(x)$	Abstract (Inductive)
$\mathsf{lseg}(x, t) * t \mapsto [tl : y] * \mathsf{list}(y) \;\vdash\; \mathsf{list}(x)$	Abstract (Roll)

Works great!

$\ddot{\smile}$

$\mathsf{emp} \vdash \mathsf{emp}$	Axiom!
$\mathsf{list}(x) \vdash \mathsf{list}(x)$	Subtract
$\mathsf{lseg}(x, t) * \mathsf{list}(t) \vdash \mathsf{list}(x)$	Abstract (Inductive)
$\mathsf{lseg}(x, t) * t \mapsto [tl : y] * \mathsf{list}(y) \vdash \mathsf{list}(x)$	Abstract (Roll)

Works great!

$\ddot{\smile}$

$\mathsf{emp} \vdash \mathsf{emp}$	Axiom!
$\mathsf{list}(x) \vdash \mathsf{list}(x)$	Subtract
$\mathsf{lseg}(x, t) * \mathsf{list}(t) \vdash \mathsf{list}(x)$	Abstract (Inductive)
$\mathsf{lseg}(x, t) * t \mapsto [tl : y] * \mathsf{list}(y) \vdash \mathsf{list}(x)$	Abstract (Roll)

$\mathsf{lseg}(x, t) * t \mapsto \mathtt{nil} * \mathsf{list}(y) \vdash \mathsf{list}(x)$	Abstract (Inductive)

Works great!

$\ddot\smile$

$\mathsf{emp} \vdash \mathsf{emp}$	Axiom!
$\mathsf{list}(x) \vdash \mathsf{list}(x)$	Subtract
$\mathsf{lseg}(x,t) * \mathsf{list}(t) \vdash \mathsf{list}(x)$	Abstract (Inductive)
$\mathsf{lseg}(x,t) * t \mapsto [tl : y] * \mathsf{list}(y) \vdash \mathsf{list}(x)$	Abstract (Roll)

$\mathsf{list}(x) * \mathsf{list}(y) \vdash \mathsf{list}(x)$	Subtract
$\mathsf{lseg}(x,t) * t \mapsto \mathtt{nil} * \mathsf{list}(y) \vdash \mathsf{list}(x)$	Abstract (Inductive)

Works great!

$\ddot\smile$

$\mathsf{emp} \vdash \mathsf{emp}$	Axiom!
$\mathsf{list}(x) \vdash \mathsf{list}(x)$	Subtract
$\mathsf{lseg}(x, t) * \mathsf{list}(t) \vdash \mathsf{list}(x)$	Abstract (Inductive)
$\mathsf{lseg}(x, t) * t \mapsto [tl : y] * \mathsf{list}(y) \vdash \mathsf{list}(x)$	Abstract (Roll)

$\ddot\frown$

$\mathsf{list}(y) \vdash \mathsf{emp}$	Junk: Not Axiom!
$\mathsf{list}(x) * \mathsf{list}(y) \vdash \mathsf{list}(x)$	Subtract
$\mathsf{lseg}(x, t) * t \mapsto \mathtt{nil} * \mathsf{list}(y) \vdash \mathsf{list}(x)$	Abstract (Inductive)

List of abstraction rules for lseg

Rolling

$$\mathsf{emp} \quad \longrightarrow \quad \mathsf{lseg}(E, E)$$

$$E_1 \neq E_3 \wedge E_1 \mapsto [tl : E_2, \rho] * \mathsf{lseg}(E_2, E_3) \quad \longrightarrow \quad \mathsf{lseg}(E_1, E_3)$$

Induction Avoidance

$$\mathsf{lseg}(E_1, E_2) * \mathsf{lseg}(E_2, \mathsf{nil}) \quad \longrightarrow \quad \mathsf{lseg}(E_1, \mathsf{nil})$$

$$\mathsf{lseg}(E_1, E_2) * E_2 \mapsto [t : \mathsf{nil}] \quad \longrightarrow \quad \mathsf{lseg}(E_1, \mathsf{nil})$$

$$\mathsf{lseg}(E_1, E_2) * \mathsf{lseg}(E_2, E_3) * E_3 \mapsto [\rho] \quad \longrightarrow \quad \mathsf{lseg}(E_1, E_3) * E_3 \mapsto [\rho]$$

$$E_3 \neq E_4 \wedge \mathsf{lseg}(E_1, E_2) * \mathsf{lseg}(E_2, E_3) * \mathsf{lseg}(E_3, E_4)$$

$$\longrightarrow \quad \mathsf{lseg}(E_1, E_3) * \mathsf{lseg}(E_3, E_4)$$

Proof Procedure for $Q_1 \vdash Q_2$, Normalization Phase

- Substitute out all equalities

$$\frac{Q_1[E/x] \vdash Q_2[E/x]}{x = E \land Q_1 \vdash Q_2}$$

- Generate disequalities. E.g., using

$$x \mapsto [\rho] * y \mapsto [\rho'] \;\rightarrow\; x \neq y$$

- Remove empty lists and trees: $\mathsf{lseg}(x, x)$, $\mathsf{tree}(\mathtt{nil})$
- Check antecedent for inconsistency, if so, return "valid". Inconsistencies:
  $x \mapsto [\rho] * x \mapsto [\rho'] \qquad \mathtt{nil} \mapsto - \qquad x \neq x \qquad \cdots$
- Check pure consequences (easy inequational logic), if failed then "invalid"

Proof Procedure for $Q_1 \vdash Q_2$, Abstract/Subtract Phase

Trying to prove $B_1 \wedge H_1 \vdash H_2$

- For each spatial predicate in H_2, try to apply abstraction rules to match it with things in H_1.
- Then, apply subtraction rule.

$$\frac{Q_1 \vdash Q_2}{Q_1 * S \vdash Q_2 * S}$$

- If you are left with

$$B \wedge \mathsf{emp} \vdash \mathsf{true} \wedge \mathsf{emp}$$

report "valid", else "invalid"

Completeness

- Proof procedure is complete (and quadratic) when we know all the listsegs are nonempty (when $x \neq y$ is there for each $\mathsf{lseg}(x, y)$).

Completeness

- Proof procedure is complete (and quadratic) when we know all the listsegs are nonempty (when $x \neq y$ is there for each $\mathsf{lseg}(x, y)$).
- Complete procedure for general case, using excluded middle

$$\frac{x = y \wedge Q_1 \vdash Q_2 \qquad\qquad x{\neq}y \wedge Q_1 \vdash Q_2}{Q_1 \vdash Q_2}$$

Completeness

- Proof procedure is complete (and quadratic) when we know all the listsegs are nonempty (when $x \neq y$ is there for each $\mathsf{lseg}(x, y)$).
- Complete procedure for general case, using excluded middle

$$\frac{x = y \wedge Q_1 \vdash Q_2 \qquad x \neq y \wedge Q_1 \vdash Q_2}{Q_1 \vdash Q_2}$$

- The resulting proof procedure is exponential. Never implemented.

Part III

Automatically Inferring Frame Axioms

A Small Spec, and a Small Proof

- Spec
  $[\mathtt{tree}(p)]$ $\mathtt{DispTree}(p)$ $[\mathtt{emp}]$

- Proof of body of recursive procedure

  $[\mathtt{tree}(i) * \mathtt{tree}(j)]$
  $\mathtt{DispTree}(i);$
  $[\mathtt{emp} * \mathtt{tree}(j)]$
  $\mathtt{DispTree}(j);$
  $[\mathtt{emp}]$

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}} \text{ Frame Rule}$$

A Small Spec, and a Small Proof

- Spec
  $[\texttt{tree}(p)]\ \texttt{DispTree}(p)\ [\texttt{emp}]$

- Proof of body of recursive procedure

  $[\texttt{tree}(i) * \texttt{tree}(j)]$
  $\texttt{DispTree}(i);$
  $[\texttt{emp} * \texttt{tree}(j)]$
  $\texttt{DispTree}(j);$
  $[\texttt{emp}]$

To automate
we must infer frames
during ``execution”

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}}\ \text{Frame Rule}$$

Frame Inference: An Extension to the Entailment Question

$$A \vdash B$$

Frame Inference: An Extension to the Entailment Question

$$A \vdash B * ?$$

Frame Inference: An Extension to the Entailment Question

$$\texttt{tree}(i) * \texttt{tree}(j) \ \vdash \ \texttt{tree}(i) * ?$$

Frame Inference: An Extension to the Entailment Question

$$\texttt{tree}(i) * \texttt{tree}(j) \;\vdash\; \texttt{tree}(i) * \texttt{tree}(j)$$

Frame Inference: An Extension to the Entailment Question

$$x \neq \texttt{nil} \wedge \mathsf{list}(x) \;\vdash\; \exists x'.\, x \mapsto x' * ?$$

Frame Inference: An Extension to the Entailment Question

$$x \neq \mathtt{nil} \wedge \mathsf{list}(x) \ \vdash \ \exists x'.\, x \mapsto x' * \mathsf{list}(x')$$

Frame Inference: An Extension to the Entailment Question

$$A \vdash B * \textcolor{red}{?}$$

How to infer a frame

Convert a failed derivation

$\mathsf{list}(y) \vdash \mathsf{emp}$	Junk: Not Axiom!
$\mathsf{list}(x) * \mathsf{list}(y) \vdash \mathsf{list}(x)$	Subtract
$\mathsf{lseg}(x, t) * t \mapsto \mathtt{nil} * \mathsf{list}(y) \vdash \mathsf{list}(x)$	Abstract (Inductive)

into a successful one

$\mathsf{emp} \vdash \mathsf{emp}$	Axiom
$\mathsf{list}(y) \vdash \mathsf{list}(y)$	Subtract
$\mathsf{list}(x) * \mathsf{list}(y) \vdash \mathsf{list}(x) * \mathsf{list}(y)$	Subtract
$\mathsf{lseg}(x, t) * t \mapsto \mathtt{nil} * \mathsf{list}(y) \vdash \mathsf{list}(x) * \mathsf{list}(y)$	Abstract (Inductive)

How to infer a frame, more generally

- Problem: $A \vdash B * ?$
- Apply abstraction and subtraction to shrink your goal: if you get to $F \vdash \texttt{emp}$ then F is your frame axiom.

$$
\begin{array}{cc}
F \vdash \texttt{emp} & \Uparrow \\
\vdots & \Uparrow \\
A \vdash B & \Uparrow
\end{array}
$$

- Sometimes you need to deal with multiple leaves at top (case analysis)

Part IV

Abstract Interpretation

Cooking a Program Analyzer

1. Just write an interpreter. (Well, an *abstract* interpreter.)
2. Symbolically execute statements using in-place reasoning (all true Hoare triples).
3. Interpret while loops by using abstractin rules like

$$\mathsf{ls}(x, t') * \mathsf{list}(t') \vdash \mathsf{list}(x)$$

   to automatically find loop invariants. This uses the rule of consequence on the right to find the invariant for the while rule

$$\frac{\{P\}C\{Q\} \quad Q \vdash Q'}{\{P\}C\{Q'\}} \qquad \frac{\{I \wedge B\}C\{I\}}{\{I\}\texttt{while}\ B\ \texttt{do}\ \{I \wedge \neg B\}}$$

4. A terminating run of the interpreter will give us a **proof** of assertions at all program points.

Example

```
{emp}
x=nil;
while (_ ){
        new(y);
        y ->tl = x;
        x=y;
}
```

Calculated Loop Invariant

$\vee$

$\vee$

Example

```
{emp}
x=nil;
while (_ ){      x = nil ∧ emp
        new(y);
        y ->tl = x;
        x=y;
}
```

Calculated Loop Invariant

$$x = \mathtt{nil} \land \mathsf{emp}$$

$\lor$

$\lor$

Example

$\{\texttt{emp}\}$
$x{=}\texttt{nil};$
$while\ (\_\ )\{ \qquad x \mapsto \texttt{nil}$
$\qquad \texttt{new}(y);$
$\qquad y \texttt{->} tl = x;$
$\qquad x{=}y;$
$\}$

Calculated Loop Invariant

$$x = \texttt{nil} \wedge \mathsf{emp}$$
$$\vee \quad x \mapsto \texttt{nil}$$
$$\vee$$

Example

$\{\texttt{emp}\}$

x=$\texttt{nil}$;

while (_){ $\quad x \mapsto x' * x' \mapsto \texttt{nil}$

$\quad\quad \texttt{new}(y);$

$\quad\quad y$ -> $tl = x;$

$\quad\quad x=y;$

}

Calculated Loop Invariant

$$x = \texttt{nil} \wedge \texttt{emp}$$

$$\vee \quad x \mapsto \texttt{nil}$$

$$\vee$$

Example

$\{\texttt{emp}\}$

```
x=nil;
while (_ ){          ls(x, nil)
    new(y);
    y->tl = x;
    x=y;
}
```

Calculated Loop Invariant

$$
\begin{aligned}
& x = \texttt{nil} \wedge \mathsf{emp} \\
\vee \quad & x \mapsto \texttt{nil} \\
\vee \quad & \mathsf{ls}(x, \texttt{nil})
\end{aligned}
$$

Example

$\{\texttt{emp}\}$

x=$\texttt{nil}$;

while (_){ $\quad x \mapsto x' * \mathsf{ls}(x', \texttt{nil})$

  $\texttt{new}(y)$;

  y ->$tl = x$;

  x=y;

}

Calculated Loop Invariant

$$x = \texttt{nil} \land \mathsf{emp}$$

$$\lor \quad x \mapsto \texttt{nil}$$

$$\lor \quad \mathsf{ls}(x, \texttt{nil})$$

Example

$\{\texttt{emp}\}$

$x{=}\texttt{nil};$

while (_){ $\qquad \mathsf{ls}(x, \texttt{nil})$

$\quad \texttt{new}(y);$

$\quad y \texttt{->} tl = x;$

$\quad x{=}y;$

}

Calculated Loop Invariant

$$x = \texttt{nil} \wedge \mathsf{emp}$$

$$\vee \quad x \mapsto \texttt{nil}$$

$$\vee \quad \mathsf{ls}(x, \texttt{nil})$$

Example

$\{\mathtt{emp}\}$

$x{=}\mathtt{nil};$

$while\ (\_\ )\{$ $\qquad \mathsf{ls}(x, \mathtt{nil})$

$\qquad \mathtt{new}(y);$

$\qquad y \texttt{->} tl = x;$

$\qquad x{=}y;$

$\}$

Calculated Loop Invariant

$$x = \mathtt{nil} \wedge \mathsf{emp}$$
$$\vee \quad x \mapsto \mathtt{nil}$$
$$\vee \quad \mathsf{ls}(x, \mathtt{nil})$$

Fixed-point reached!

Perspective: this seemingly exotic logic has pretty effective proof technology

- Can mix with other provers. Apply subtraction rule to simplify formulae as a tactic, or make call-outs other proof procedures as you go.
  - **Parkinson***'s talk later today: call-out to Z3 SMT solver;*
  - **Shao***'s talk later: proof theory inside Coq tactics.*
  - **Tuerk***'s poster and paper in proceedings: HOLfoot.*
  - *And other tools (Tokyo, Sydney, Singapore) use Omega, Isabelle...*
- Frame inference can be used to modularize a program analysis or (I imagine) to increase automation in an interactive proof.
- The induction-avoiding abstraction rules, like

$$\mathsf{lseg}(x, t) * \mathsf{list}(t) \vdash \mathsf{list}(x)$$

can also be used to infer loop invariants (abstract interpretation) by helping us reach fixed-points.

See talks of **Parkinson**, **Gotsman**, **Magill**, **Distefano** *later today for more on analysis.*