From Separation Logic to Systems Software

Peter O'Hearn, Queen Mary

Based on work of the *SpaceInvader* team: Cristiano Calcagno, Dino Distefano, Hongseok Yang, and me

Special thanks to our SLAyer colleagues (MSR): Josh Berdine, Byron Cook

Talk at Seoul National Univ, 11 May 2009

Part 0,
Context

Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proofs about the software and how it works in order to guarantee reliability.

Bill Gates, WINHEC conference, 2002

Some Context

- Since 2000, striking progress in automatic program proving. E.g.:
  - SLAM: Protocol properties of procedure calls in device drivers, any call to `ReleaseSpinLock` is preceded by a call to `AquireSpinLock`
  - ASTRÉE: no run-time errors in Airbus code

Some Context

- Since 2000, striking progress in automatic program proving. E.g.:
  - SLAM: Protocol properties of procedure calls in device drivers, 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)

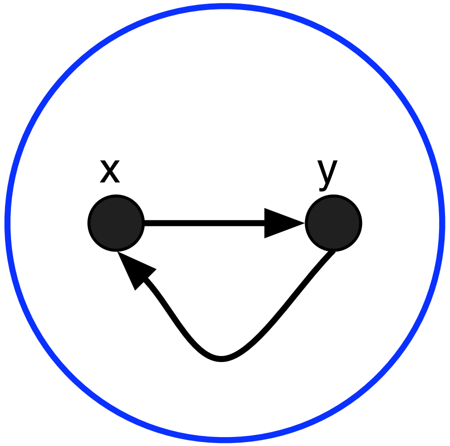
Some Context

- Since 2000, striking progress in automatic program proving. E.g.:
  - SLAM: Protocol properties of procedure calls in device drivers, 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,
Basics

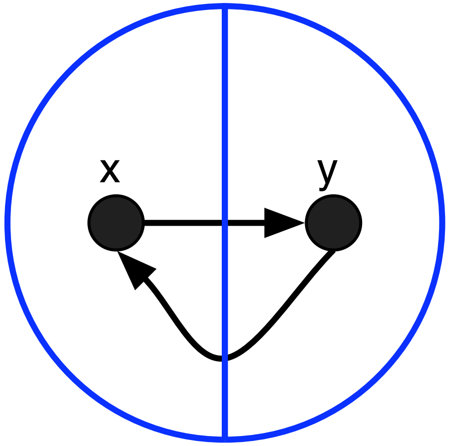
Separation Logic

xl->y * yl-> x



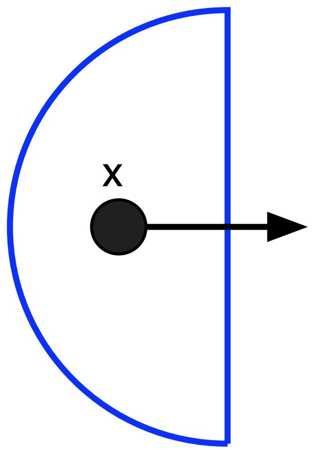
Separation Logic

xl->y * yl-> x



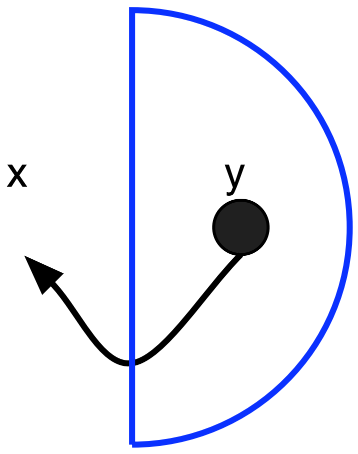
Separation Logic

xl->y



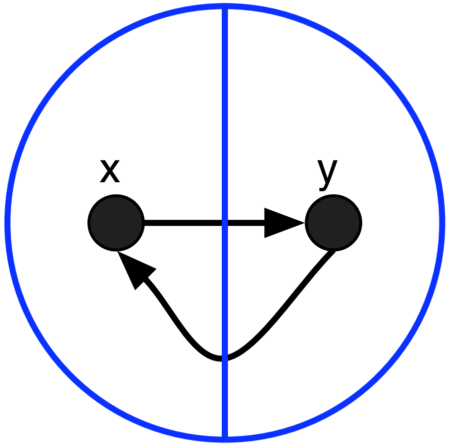
Separation Logic

yl-> x



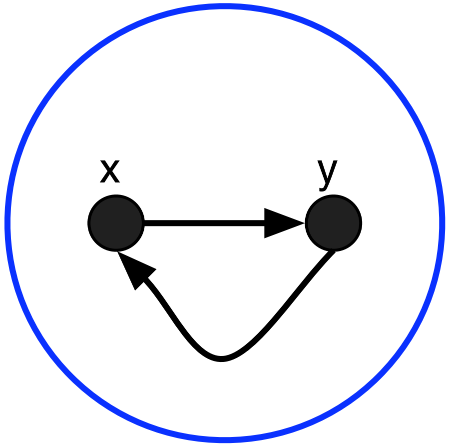
Separation Logic

xl->y * yl-> x



Separation Logic

xl->y * yl-> x



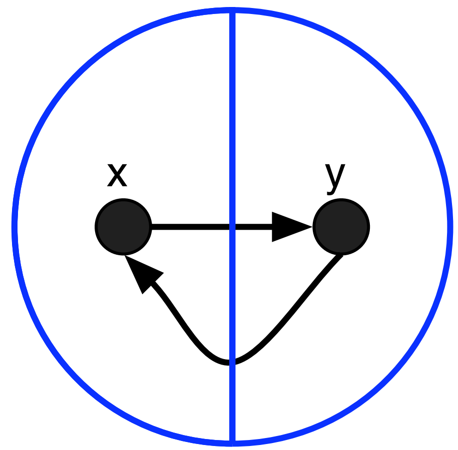
10	42
42	10

x=10

y=42

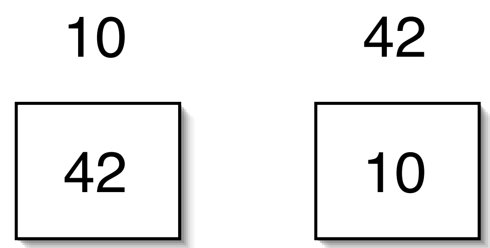
Separation Logic

xl->y * yl-> x



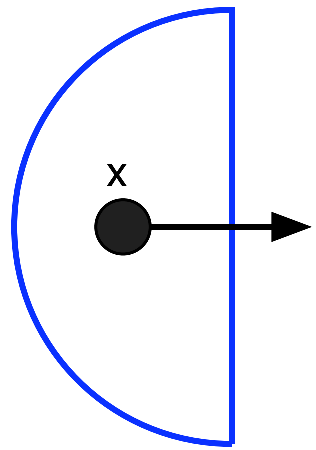
x=10

y=42

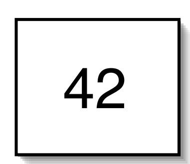


Separation Logic

xl->y



10

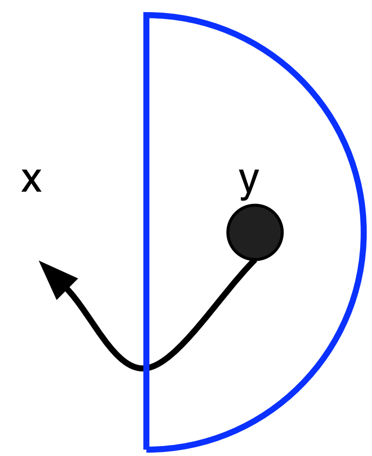


x=10

y=42

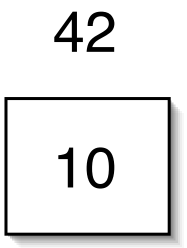
Separation Logic

y|-> x



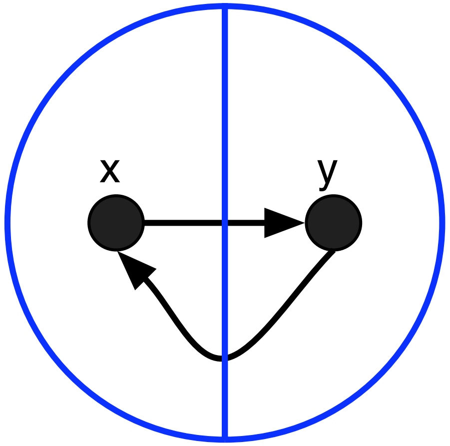
x=10

y=42



Separation Logic

xl->y * yl-> x



A Substructural Logic

$$A \nvdash A * A$$

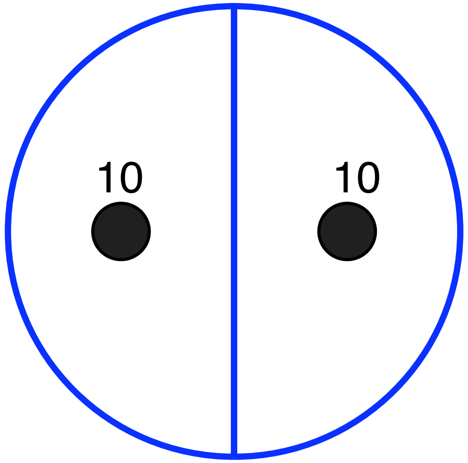
$$10 \mapsto 3 \nvdash 10 \mapsto 3 * 10 \mapsto 3$$

$$A * B \nvdash A$$

$$10 \mapsto 3 * 42 \mapsto 5 \nvdash 10 \mapsto 3$$

An inconsistency: trying to be two places at once

10|->3 * 10|->3



Heaplets (heap portions) as possible worlds (i.e., a kind of modal logic)

- Add to Classical Logic:
  - `emp` : "the heaplet is empty"
  - $x \mapsto y$: "the heaplet has *exactly* one cell x, holding y"
  - $A * B$: "the heaplet can be divided so A is true of one partition and B of the other".

Heaplets (heap portions) as possible worlds (i.e., a kind of modal logic)

- Add to Classical Logic:
  - `emp` : "the heaplet is empty"
  - $x \mapsto y$: "the heaplet has *exactly* one cell x, holding y"
  - $A * B$: "the heaplet can be divided so A is true of one partition and B of the other".
- Add inductive definitions , and other more exotic things ("magic wand", "septraction") as well.

Heaplets (heap portions) as possible worlds (i.e., a kind of modal logic)

- Add to Classical Logic:
  - `emp` : "the heaplet is empty"
  - $x \mapsto y$: "the heaplet has *exactly* one cell x, holding y"
  - $A * B$: "the heaplet can be divided so A is true of one partition and B of the other".
- Add inductive definitions , and other more exotic things ("magic wand", "septraction") as well.
- Standard model: RAM model

$$heap : N \rightharpoonup_f Z$$

and lots of variations (records, permissions, ownership... more later).

Algebraic Structure

- We can lift $\circ \colon H \times H \rightharpoonup H$ to $* \colon \mathcal{P}(H) \times \mathcal{P}(H) \to \mathcal{P}(H)$

$$h \in A * B \quad \text{iff} \quad \exists h_A, h_B.\, h = h_A \circ h_B \text{ and}$$
$$h_A \in A \text{ and } h_B \in B$$

- $\mathtt{emp} = \{e\}$.
  - "I have a heap, and it is empty" (not the empty set of heaps)
  - $(\mathcal{P}(H), *, \mathtt{emp})$ is a *total* commutative monoid
- $\mathcal{P}(H)$ is (in the subset order) *both*
  - A Boolean Algebra, and
  - A Residuated Monoid

$$A * B \subseteq C \quad \Leftrightarrow \quad A \subseteq B \mathrel{-\!\!*} C$$

- cf. Boolean BI logic (O'Hearn, Pym)

In-place Reasoning

$$[(x \mapsto -) * P] \ [x]{:=}7 \ [(x \mapsto 7) * P]$$

$$[P * (x \mapsto -)] \ \mathtt{dispose}(x) \ [P]$$

$$[P] \ x = \mathtt{cons}(a, b) \ [P * (x \mapsto a, b)] \qquad (x \notin \mathit{free}(P))$$

In-place reasoning and Inductive Definitions

Example Inductive Definition:

$$\mathtt{tree}(E) \iff \begin{array}{l} \mathit{if}\ E{=}nil\ \mathit{then}\ \mathtt{emp} \\ \mathit{else}\ \exists x, y.\,(E{\mapsto}l\colon x, r\colon y) * \mathtt{tree}(x) * \mathtt{tree}(y) \end{array}$$

Example Proof:

$$\{\mathtt{tree}(p) \land p \neq \mathtt{nil}\}$$

$$i{:=}\,p{\rightarrow}l;\ \ j{:=}\,p{\rightarrow}r;$$

$$\mathtt{dispose}(p);$$

$$\{\,\mathtt{tree}(i) * \mathtt{tree}(j)\}$$

In-place reasoning and Inductive Definitions

Example Inductive Definition:

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

Example Proof:

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

$\{(p{\mapsto}l\colon x', r\colon y') * \mathtt{tree}(x') * \mathtt{tree}(y')\}$

$i{:=}\,p{\rightarrow}l;\;\; j{:=}\,p{\rightarrow}r;$

$\mathtt{dispose}(p);$

$\{\,\mathtt{tree}(i) * \mathtt{tree}(j)\}$

In-place reasoning and Inductive Definitions

Example Inductive Definition:

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

Example Proof:

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

$\{(p \mapsto l : x', r : y') * \mathtt{tree}(x') * \mathtt{tree}(y')\}$

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

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

$\mathtt{dispose}(p);$

$\{\mathtt{tree}(i) * \mathtt{tree}(j)\}$

In-place reasoning and Inductive Definitions

Example Inductive Definition:

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

Example Proof:

$$\{\mathtt{tree}(p) \wedge p \neq \mathtt{nil}\}$$
$$\{(p{\mapsto}l\colon x', r\colon y') * \mathtt{tree}(x') * \mathtt{tree}(y')\}$$
$$i{:=}\,p{\rightarrow}l;\;\; j{:=}\,p{\rightarrow}r;$$
$$\{(p{\mapsto}l\colon i, r\colon j) * \mathtt{tree}(i) * \mathtt{tree}(j)\}$$
$$\mathtt{dispose}(p);$$
$$\{\mathtt{emp} * \mathtt{tree}(i) * \mathtt{tree}(j)\}$$
$$\{\,\mathtt{tree}(i) * \mathtt{tree}(j)\}$$

Extended In-place Reasoning

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

- Rest of proof of evident recursive procedure

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

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

Extended In-place Reasoning

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

- Rest of proof of evident recursive procedure

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

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

Extended In-place Reasoning

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

- Rest of proof of evident recursive procedure

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

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

Extended In-place Reasoning

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

- Rest of proof of evident recursive procedure

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

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

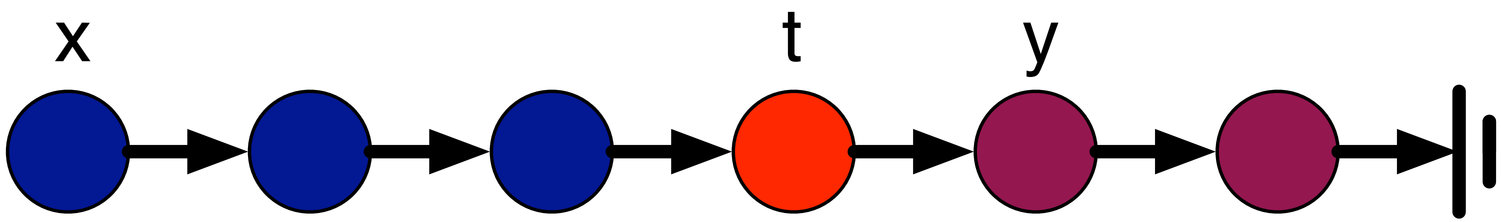
Part II,
Cooking a Static Analyzer

Linked Lists

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

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

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



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

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

Calculated Loop Invariant

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

$\vee$

$\vee$

Example

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

Calculated Loop Invariant

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

Example

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

Calculated Loop Invariant

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

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

$$\vee$$

Example

$\{\texttt{emp}\}$

x=`nil`;

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

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

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

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

$\quad$ $x = y$;

}

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=`nil`*;*
while (_){ $\mathsf{ls}(x, \texttt{nil})$
    `new`*(y);*
    y ->tl = x;
    x=y;
}

Calculated Loop Invariant

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

Example

$\{\texttt{emp}\}$

x=$\texttt{nil}$;

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

new(y);

y ->tl = x;

x=y;

}

Calculated Loop Invariant

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

Fixed-point reached!

Part III:
A new recipe from East London


Shelf

Footprints and Small Specs

- Semantics: Program P, with
  - $P, h \Rightarrow h'$ or $P, h \Rightarrow \text{memfault}$
- Footprint (Input Footprint)

$$h \in Foot(P) \quad \Leftrightarrow \quad P, h \not\Rightarrow \text{memfault} \qquad (Safety)$$
$$\wedge \quad \forall h' \subset h.\ P, h' \Rightarrow \text{memfault} \qquad (Minimality)$$

- Small Spec of P: $[Foot(P)]\ P\ [Post(P)]$

We achieve compositionality,

by aiming for ``small specs"

that describe the footprint

We achieve compositionality,

by aiming for ``small specs”

that describe the footprint

An Example Small Spec

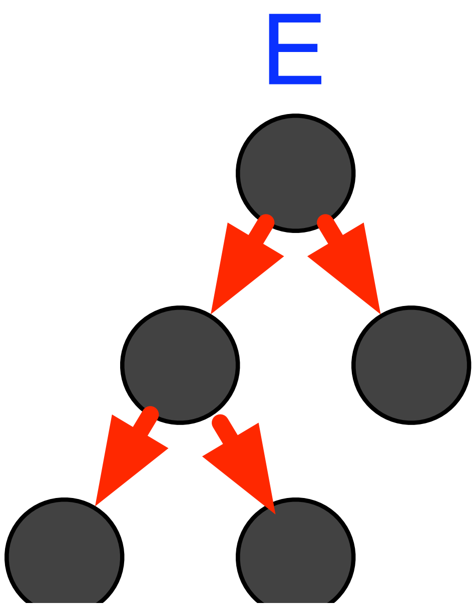
$$\{\mathtt{tree}(p)\}\ \mathtt{DispTree}(p)\ \{\mathtt{emp}\}$$

where

$$\mathtt{tree}(E) \iff \begin{array}{l} \textit{if } E{=}\textit{nil} \textit{ then } \mathtt{emp} \\ \textit{else } \exists x, y.\,(E \mapsto l\colon x, r\colon y) * \mathtt{tree}(x) * \mathtt{tree}(y) \end{array}$$

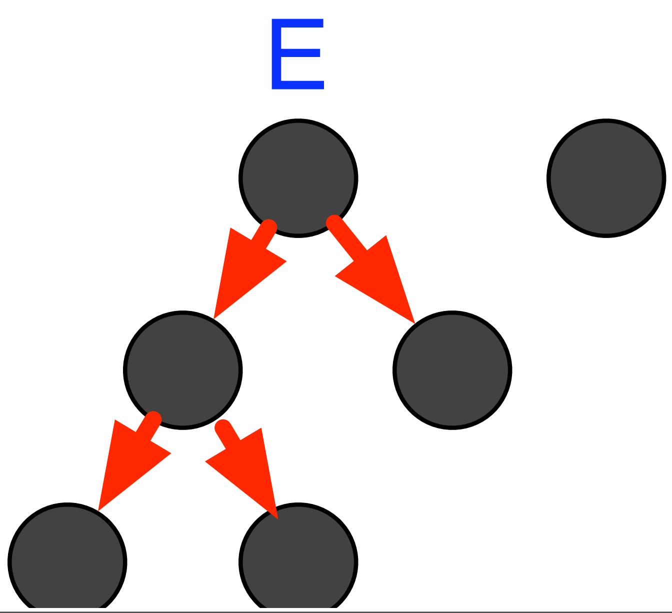
The "smallness" of the tree assertion

- $$\mathtt{tree}(E) \iff \textit{if } E{=}\textit{nil} \textit{ then } \mathtt{emp}$$
  $$\textit{else } \exists x, y.\, (E \mapsto l\colon x, r\colon y) * \mathtt{tree}(x) * \mathtt{tree}(y)$$
- $\mathtt{tree}(E)$ is true of



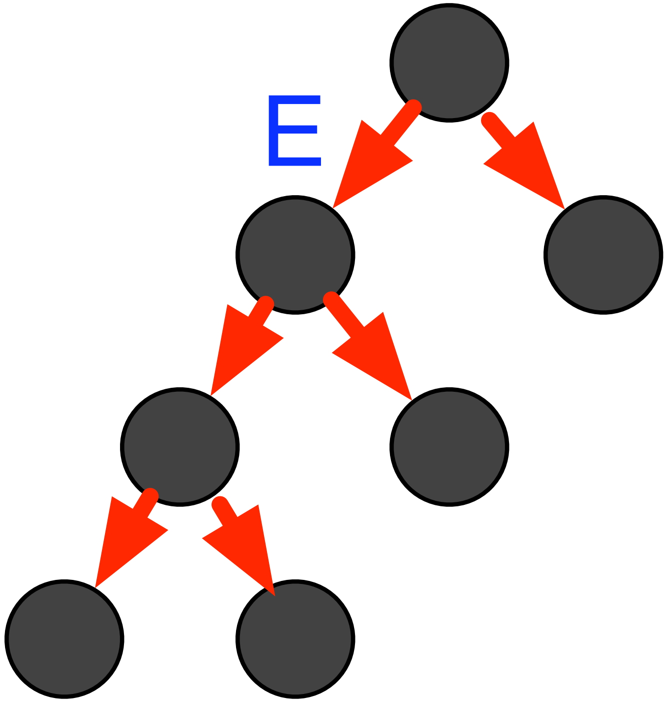
The "smallness" of the tree assertion

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



The "smallness" of the tree assertion

- $$\texttt{tree}(E) \iff \textit{if } E{=}\textit{nil} \textit{ then } \texttt{emp}$$
  $$\textit{else } \exists x, y. (E {\mapsto} l\colon x, r\colon y) * \texttt{tree}(x) * \texttt{tree}(y)$$
- and even false of



The AI Frame Problem (McCarthy-Hayes, 1969)

- When you specify an action {P}act{Q}, an inordinate amount of effort is needed to say what "act" DOSN'T do.
- { not(holding(block)) } pick-up(block) { holding(block) }
- { holding(block2) } pick-up(block) { holding(block2) } ???

Some Philosophical Problems from the Standpoint of Artifical Intelligence, McCarthy-Hayes, Machine Intelligence, 1969

The AI Frame Problem (McCarthy-Hayes, 1969)

- When you specify an action {P}act{Q}, an inordinate amount of effort is needed to say what "act" DOSN'T do.
- { not(holding(block)) } pick-up(block) { holding(block) }
- { holding(block2) } pick-up(block) { holding(block2) } ???

Frame Axiom

Some Philosophical Problems from the Standpoint of Artific
McCarthy-Hayes, Machine Intelligence, 1969

A Small Spec, and a Small Proof

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

- Proof of body of recursive procedure

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

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

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”

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

Extensions of the entailment question I: Frame Inference

$$A \vdash B$$

Extensions of the entailment question I: Frame Inference

$$A \vdash B * ?$$

Extensions of the entailment question I: Frame Inference

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

Extensions of the entailment question I: Frame Inference

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

Extensions of the entailment question I: Frame Inference

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

Extensions of the entailment question I: Frame Inference

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

Extensions of the entailment question I: Frame Inference

$$A \vdash B * ?$$

A Small Spec, and a Small Proof

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

- Proof of body of recursive procedure

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

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

Wait a minute, where are you gonna get preconditions? How to get started?

Wait a minute, where are you gonna get preconditions? How to get started?

Oh, don't tell me, that sounds... out of this world...


Abductive Inference
(Charles Peirce, circa 1900, writing about the scientific process)

"Abduction is the process of forming an explanatory hypothesis. It is the only logical operation which introduces any new idea"

"A man must be downright crazy to deny that science has made many true discoveries. But every single item of scientific theory which stands established today has been due to Abduction."

The Collected Papers of Charles Sanders Peirce, Volume V, Pragmatism and Pragmaticism

Extensions of the entailment question II: abduction

- $$A * ? \;\vdash\; B$$

[1] Calcagno, Distefano, O'Hearn, Yang, POPL'09

Extensions of the entailment question II: abduction

- $$x \mapsto \texttt{nil} * ? \;\vdash\; \mathit{list}(x) * \mathit{list}(y)$$
- We call the ? here an “anti-frame”.[1]

[1] Calcagno, Distefano, O’Hearn, Yang, POPL’09

Extensions of the entailment question II: abduction

- $$x \mapsto \texttt{nil} * list(y) \ \vdash \ list(x) * list(y)$$
- We call the ? here an "anti-frame".[1]

[1] Calcagno, Distefano, O'Hearn, Yang, POPL'09

Extensions of the entailment question II: abduction

- $$x \mapsto y * ? \;\vdash\; x \mapsto a * list(a)$$

- We call the ? here an "anti-frame".[1]

[1] Calcagno, Distefano, O'Hearn, Yang, POPL'09

Extensions of the entailment question II: abduction

- $$x \mapsto y * \left(y = a \wedge \mathit{list}(a)\right) \;\vdash\; x \mapsto a * \mathit{list}(a)$$

- We call the ? here an "anti-frame".[1]

[1] Calcagno, Distefano, O'Hearn, Yang, POPL'09

Abduction Example: Inferring a pre/post pair

```
void p(list-item *y) {
    list-item *x;
    x=malloc(sizeof(list-item));
    x→tail = 0;
    foo(x,y);
    return(x); }
```

Abductive Inference:

Given Summary/spec: $[\mathrm{list}(x) * \mathrm{list}(y)]\, foo(x, y)\, [\mathrm{list}(x)]$

Abduction Example: Inferring a pre/post pair

```
void p(list-item *y) {                emp
    list-item *x;
    x=malloc(sizeof(list-item));
    x→tail = 0;
    foo(x,y);
    return(x); }
```

Abductive Inference:

Given Summary/spec: $[\mathsf{list}(x) * \mathsf{list}(y)]\, foo(x, y)\, [\mathsf{list}(x)]$

Abduction Example: Inferring a pre/post pair

```
void p(list-item *y) {                emp
    list-item *x;
    x=malloc(sizeof(list-item));
    x→tail = 0;                       x ↦ 0
    foo(x,y);
    return(x); }
```

Abductive Inference:

Given Summary/spec: $[\mathrm{list}(x) * \mathrm{list}(y)]\, \mathit{foo}(x, y)\, [\mathrm{list}(x)]$

Abduction Example: Inferring a pre/post pair

```
void p(list-item *y) {                    emp
    list-item *x;
    x=malloc(sizeof(list-item));
    x→tail = 0;                           x ↦ 0
    foo(x,y);
    return(x); }
```

Abductive Inference: $x \mapsto 0 * \,?\quad \vdash \mathrm{list}(x) * \mathrm{list}(y)$

Given Summary/spec: $[\mathrm{list}(x) * \mathrm{list}(y)]\,foo(x, y)\,[\mathrm{list}(x)]$

Abduction Example: Inferring a pre/post pair

`void p(list-item *y) {`	$\mathtt{emp}$
`    list-item *x;`	
`    x=malloc(sizeof(list-item));`	
`    x→tail = 0;`	$x \mapsto 0$
`    foo(x,y);`	
`    return(x); }`	

Abductive Inference: $x \mapsto 0 * \ \mathrm{list}(y) \vdash \mathrm{list}(x) * \mathrm{list}(y)$

Given Summary/spec: $[\mathrm{list}(x) * \mathrm{list}(y)]\mathit{foo}(x, y)[\mathrm{list}(x)]$

Abduction Example: Inferring a pre/post pair

`void p(list-item *y) {`	$\mathtt{emp}$	$\mathsf{list}(y)$
`    list-item *x;`		
`    x=malloc(sizeof(list-item));`		
`    x→tail = 0;`	$x \mapsto 0$	
`    foo(x,y);`		
`    return(x); }`		

Abductive Inference: $x \mapsto 0 \ast \ \mathsf{list}(y) \vdash \mathsf{list}(x) \ast \mathsf{list}(y)$

Given Summary/spec: $[\mathsf{list}(x) \ast \mathsf{list}(y)] \mathit{foo}(x, y) [\mathsf{list}(x)]$

Abduction Example: Inferring a pre/post pair

`void p(list-item *y) {`	$\mathtt{emp}$	$\mathsf{list}(y)$
`    list-item *x;`		
`    x=malloc(sizeof(list-item));`		
`    x→tail = 0;`	$x \mapsto 0$	
`    foo(x,y);`	$\mathsf{list}(x)$	
`    return(x); }`		

Abductive Inference: $x \mapsto 0 * \mathsf{list}(y) \vdash \mathsf{list}(x) * \mathsf{list}(y)$

Given Summary/spec: $[\mathsf{list}(x) * \mathsf{list}(y)]\, foo(x, y)\, [\mathsf{list}(x)]$

Abduction Example: Inferring a pre/post pair

`void p(list-item *y) {`	$\mathtt{emp}$	$\mathsf{list}(y)$
`    list-item *x;`		
`    x=malloc(sizeof(list-item));`		
`    x→tail = 0;`	$x \mapsto 0$	
`    foo(x,y);`	$\mathsf{list}(x)$	
`    return(x); }`		$\mathsf{list}(ret)$

Abductive Inference: $x \mapsto 0 * \mathsf{list}(y) \vdash \mathsf{list}(x) * \mathsf{list}(y)$

Given Summary/spec: $[\mathsf{list}(x) * \mathsf{list}(y)]\mathit{foo}(x, y)[\mathsf{list}(x)]$

Abduction Example: Inferring a pre/post pair

void p(list-item *y) {	`emp`	$\mathsf{list}(y)$(Inferred Pre)
list-item *x;		
x=malloc(sizeof(list-item));		
x→tail = 0;	$x \mapsto 0$	
foo(x,y);	$\mathsf{list}(x)$	
return(x); }		$\mathsf{list}(ret)$(Inferred Post)

Abductive Inference: $x \mapsto 0 * \mathsf{list}(y) \vdash \mathsf{list}(x) * \mathsf{list}(y)$

Given Summary/spec: $[\mathsf{list}(x) * \mathsf{list}(y)]\, foo(x, y)\, [\mathsf{list}(x)]$

Bi-Abduction

$$A \ast \text{?anti-frame} \vdash B \ast \text{?frame}$$

- Generally, we have to solve both inference questions at each procedure call site (and each heap dereference)
- It lets us do a bottom-up analysis: callees before callers. Generates pre/post specs without being given preconditions or postconditions.

Experimental Results

STRESS:specs should fit together

small example to test how accurate the specs are

Experimental Results

STRESS:specs should fit together

small example to test how accurate the specs are

- Small examples
  - Recursive procedures for traversing/deleting/inserting in acyclic/cyclic nested lists
- Medium examples
  - Firewire device driver (10K LOC) found specs for 121 procedures out of 121

Abductor on larger programs

Program	MLOC	Num. Procs	Proven Procs	Procs %	Time (sec)
Linux 2.6.25.4	2.473	101330	59215	58.4	6869.09
Gimp 2.4.6	0.708	15114	6364	42.1	3601.16
OpenSSL 0.9.8g	0.214	4818	2967	61.6	605.36
Sendmail 8.14.3	0.108	684	353	51.6	184.50
Apache 2.2.8	0.102	1870	881	47.1	294.67
OpenSSH 5.0	0.073	1135	519	45.7	142.56
Spin 5.1.6	0.019	357	197	55.2	772.82

Confessions/Admissions

- Sound wrt “Idealized” model (e.g., no concurrency...)
- Don’t know good general criterion for “quality” of specs (anecdotal evidence, eyeball some examples)
- Lots of heuristics (in abduction, and in abstraction, and in join, and in predicate discovery...)
- Timeout is involved
- Hard things in extra 40% procs in Linux

Still...

$$A * \text{?anti-frame} \vdash B * \text{?frame}$$

- Bi-abduction fits conceptually very naturally with the ideas of *small specs* that talk about *footprints*
- It leads to an *extreme modular* shape analysis
- Maybe it can be used for other things too...