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)
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

\[ x \mid \rightarrow y \quad * \quad y \mid \rightarrow x \]
Separation Logic

\[ x \rightarrow{} y \quad \ast \quad y \rightarrow{} x \]
Separation Logic

\( x \mid \rightarrow y \)
Separation Logic

\[ y \mid \rightarrow x \]
Separation Logic

\[ x \rightarrow y \quad \ast \quad y \rightarrow x \]

\[ x=10 \]
\[ y=42 \]
Separation Logic

\[ x \rightarrow y \ast y \rightarrow x \]

\[ x = 10 \]
\[ y = 42 \]
Separation Logic

\( x \rightarrow y \)

\( x = 10 \)

\( y = 42 \)
Separation Logic

\[ y \vdash \neg x \]

\[ x \]

\[ y \]

\[ x = 10 \]

\[ y = 42 \]

\[ 42 \]

\[ 10 \]
Separation Logic

\[ x \mid \rightarrow y \quad \ast \quad y \mid \rightarrow x \]
A Substructural Logic

\[ A \not\vdash A \otimes A \]

\[ 10 \leftrightarrow 3 \not\vdash 10 \leftrightarrow 3 \otimes 10 \leftrightarrow 3 \]

\[ A \otimes B \not\vdash A \]

\[ 10 \leftrightarrow 3 \otimes 42 \leftrightarrow 5 \not\vdash 10 \leftrightarrow 3 \]
An inconsistency: trying to be two places at once

$10 \mapsto 3 \times 10 \mapsto 3$

10 \rightarrow 3 \times 10 \rightarrow 3
Heaplets (heap portions) as possible worlds (i.e., a kind of modal logic)

- Add to Classical Logic:
  - \texttt{emp} : “the heaplet is empty”
  - \texttt{x \mapsto y} : “the heaplet has exactly one cell \texttt{x}, holding \texttt{y}”
  - \texttt{A * B} : “the heaplet can be divided so \texttt{A} is true of one partition and \texttt{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 $y$”
  - $A \ast B$ : “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:
  - \( \text{emp} \) : “the heaplet is empty”
  - \( x \leftrightarrow y \) : “the heaplet has exactly one cell \( x \), holding \( y \)”
  - \( A \star B \) : “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

\[
\text{heap} : N \rightarrow_f Z
\]

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

- We can lift $\circ : H \times H \rightarrow H$ to $\ast : \mathcal{P}(H) \times \mathcal{P}(H) \rightarrow \mathcal{P}(H)$

  $h \in A \ast B$ iff $\exists h_A, h_B. h = h_A \circ h_B$ and $h_A \in A$ and $h_B \in B$

- $\text{emp} = \{ e \}$.
  - “I have a heap, and it is empty” (not the empty set of heaps)
  - $(\mathcal{P}(H), \ast, \text{emp})$ is a total commutative monoid

- $\mathcal{P}(H)$ is (in the subset order) **both**
  - A Boolean Algebra, and
  - A Residuated Monoid

  $A \ast B \subseteq C \iff A \subseteq B \rightarrow C$

- cf. Boolean BI logic (O’Hearn, Pym)
**In-place Reasoning**

\[
[(x \mapsto -) \ast P] \quad [x] := 7 \quad [(x \mapsto 7) \ast P]
\]

\[
[P \ast (x \mapsto -)] \quad \text{dispose}(x) \quad [P]
\]

\[
[P] \quad x = \text{cons}(a, b) \quad [P \ast (x \mapsto a, b)] \quad (x \not\in \text{free}(P))
\]
In-place reasoning and Inductive Definitions

Example Inductive Definition:

\[
\text{tree}(E) \iff \begin{cases} 
\text{if } E = \text{nil} \text{ then emp} \\
\text{else } \exists x, y. (E \mapsto l: x, r: y) \ast \text{tree}(x) \ast \text{tree}(y) 
\end{cases}
\]

Example Proof:

\[
\begin{aligned}
\{ \text{tree}(p) \land p \neq \text{nil} \} \\

i := p \mapsto l; \quad j := p \mapsto r; \\

\text{dispose}(p); \\

\{ \text{tree}(i) \ast \text{tree}(j) \}
\end{aligned}
\]
In-place reasoning and Inductive Definitions

Example Inductive Definition:

\[
\text{tree}(E) \iff \begin{cases} 
\text{if } E = \text{nil} \text{ then emp} \\
\text{else } \exists x, y. (E \mapsto l: x, r: y) \ast \text{tree}(x) \ast \text{tree}(y)
\end{cases}
\]

Example Proof:

\{
\text{tree}(p) \land p \neq \text{nil}
\}
\{(p \mapsto l: x', r: y') \ast \text{tree}(x') \ast \text{tree}(y')\}
\begin{align*}
i &:= p \mapsto l; \\
j &:= p \mapsto r;
\end{align*}

\text{dispose}(p);

\{
\text{tree}(i) \ast \text{tree}(j)
\}
In-place reasoning and Inductive Definitions

Example Inductive Definition:

\[
\text{tree}(E) \iff \begin{cases} 
\text{if } E = \text{nil} \text{ then emp} \\
\text{else } \exists x, y. (E \mapsto \xrightarrow{l}: x, \xrightarrow{r}: y) * \text{tree}(x) * \text{tree}(y) 
\end{cases}
\]

Example Proof:

\{
\text{tree}(p) \land p \neq \text{nil}
\}
\{(p \mapsto l: x', r: y') * \text{tree}(x') * \text{tree}(y')\}

i := p \mapsto l; j := p \mapsto r;

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

dispose(p);

\{ \text{tree}(i) * \text{tree}(j) \}
In-place reasoning and Inductive Definitions

Example Inductive Definition:

\[
\text{tree}(E) \iff \begin{cases}
\text{if } E = \text{nil} \text{ then emp} \\
\text{else } \exists x, y. (E \mapsto l: x, r: y) \ast \text{tree}(x) \ast \text{tree}(y)
\end{cases}
\]

Example Proof:

\[
\begin{align*}
\{& \text{tree}(p) \land p \neq \text{nil} \\
& (p \mapsto l: x', r: y') \ast \text{tree}(x') \ast \text{tree}(y') \} \\
& i := p \mapsto l; \quad j := p \mapsto r; \\
& (p \mapsto l: i, r: j) \ast \text{tree}(i) \ast \text{tree}(j) \\
& \text{dispose}(p); \\
& \{\text{emp} \ast \text{tree}(i) \ast \text{tree}(j)\} \\
& \{\text{tree}(i) \ast \text{tree}(j)\}
\end{align*}
\]
Extended In-place Reasoning

- Spec
  \{\text{tree}(p)\} \text{DispTree}(p) \{\text{emp}\}

- Rest of proof of evident recursive procedure

  \{\text{tree}(i) \ast \text{tree}(j)\}
  \text{DispTree}(i);
  \{\text{emp} \ast \text{tree}(j)\}
  \text{DispTree}(j);

\[ \{P\} C \{Q\} \quad \frac{\{P \ast R\} C \{Q \ast R\}}{} \quad \text{Frame Rule} \]
Extended In-place Reasoning

- Spec
  \{tree(p)\} \text{DispTree}(p) \{\text{emp}\}

- Rest of proof of evident recursive procedure

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

\[
\frac{\{P\} C \{Q\}}{\{P*R\} C \{Q*R\}} \quad \text{Frame Rule}
\]
Extended In-place Reasoning

Spec
\{tree(p)\} DispTree(p) \{emp\}

Rest of proof of evident recursive procedure

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

\[
\begin{align*}
\{P\} & C\{Q\} \\
\{P*R\} & C\{Q*R\}
\end{align*}
\]
Frame Rule
Extended In-place Reasoning

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

- Rest of proof of evident recursive procedure

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

\[
\frac{\{P\} C \{Q\}}{\{P*R\} C \{Q*R\}} \quad \text{Frame Rule}
\]
Part II,
Cooking a Static Analyzer
**Linked Lists**

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

\[
\text{lseg}(E, F) \iff \begin{cases} \text{emp} & \text{if } E = F \\ \exists y. E \rightarrow tl : y \ast \text{lseg}(y, F) & \text{else} \end{cases}
\]

\[
\text{lseg}(x, t) \ast t \rightarrow [tl : y] \ast \text{list}(y)
\]
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 abstract in rules like

\[ \text{ls}(x, t') \ast \text{list}(t') \vdash \text{list}(x) \]

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

\[
\begin{align*}
\{P\} C \{Q\} & \quad Q \vdash Q' \\
\{P\} C \{Q'\} & \\
\{I \land B\} C \{I\} & \\
\{I\} \text{while } B \text{ do } \{I \land \neg B\}
\end{align*}
\]

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

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

Calculated Loop Invariant

\( \vee \)
\( \vee \)
Example

{emp}
\(x = \text{nil};\)

while (\_ ){
\(x = \text{nil} \land \text{emp}\)
\(\text{new}(y);\)
\(y \rightarrow tl = x;\)
\(x = y;\)
}

Calculated Loop Invariant

\(x = \text{nil} \land \text{emp}\)
\(\lor\)
\(\lor\)
Example

\{\text{emp}\}
\ x = \text{nil};
\begin{align*}
\text{while } (\_ )\{ & \quad x \mapsto \text{nil} \\
& \quad \text{new}(y); \\
& \quad y \rightarrow t/l = x; \\
& \quad x = y;
\end{align*}

Calculated Loop Invariant

\ x = \text{nil} \land \text{emp}
\lor \quad x \mapsto \text{nil}
\lor
\lor

Example

{emp}
\( x = \text{nil}; \)
while (\_ ){
    \( x \mapsto x' \ast x' \mapsto \text{nil}\)
    new(y);
    \( y \rightarrow tl = x; \)
    \( x = y; \)
}

Calculated Loop Invariant

\[ x = \text{nil} \land \text{emp} \]
\[ \lor x \mapsto \text{nil} \]
\[ \lor \]
Example

\{emp\}

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

\texttt{while (\ldots)\{ \texttt{ls(x,nil)}

\texttt{ \hspace{1em} new(y);}

\texttt{ \hspace{1em} y \rightarrow tl = x;}

\texttt{ \hspace{1em} x=y;}

\texttt{\}}}

Calculated Loop Invariant

\( x = \texttt{nil} \land \texttt{emp} \)

\( \forall \ x \mapsto \texttt{nil} \)

\( \forall \ \texttt{ls(x,nil)} \)
Example

{x:emp}
x = nil;

while ( ) {
    x ← x' * l(x', nil)
    new(y);
    y → tl = x;
    x = y;
}

Calculated Loop Invariant

x = nil ∧ emp
∀ x → nil
∀ l(x, nil)
Example

\{\text{emp}\}
\begin{align*}
x &= \text{nil}; \\
\text{while} (\_ ) \{ &\text{ls}(x, \text{nil}) \\
&\text{new}(y); \\
&y \rightarrow tl = x; \\
&x = y; \\
\}
\end{align*}

Calculated Loop Invariant

\begin{align*}
x &= \text{nil} \land \text{emp} \\
\forall x &\rightarrow \text{nil} \\
\forall \text{ls}(x, \text{nil})
\end{align*}
Example

\{\text{emp}\}
\begin{align*}
x &= \text{nil}; \\
\text{while } (\_ ) \{ &
\begin{align*}
\text{ls}(x, \text{nil}) \\
\text{new}(y); \\
y \to tl &= x; \\
x &= y;
\end{align*}
\}
\end{align*}

Calculated Loop Invariant

\[ x = \text{nil} \land \text{emp} \]
\[ \forall x \mapsto \text{nil} \]
\[ \forall \text{ls}(x, \text{nil}) \]

Fixed-point reached!
Part III:
A new recipe from East London
Part III: A new recipe from East London
Footprints and Small Specs

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

Footprint (Input Footprint)

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

Small Spec of $P$: $[\text{Foot}(P)] P [\text{Post}(P)]$
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

\{\text{tree}(p)\} \ \text{DispTree}(p) \ \{\text{emp}\}

where

\text{tree}(E) \iff \begin{cases} \text{if } E = \text{nil} \text{ then emp} \\
\text{else } \exists x, y. (E \mapsto l: x, r: y) \ast \text{tree}(x) \ast \text{tree}(y) \end{cases}
The “smallness” of the tree assertion

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

\[
\text{tree}(E) \text{ is true of }
\]

![Diagram of a tree structure with labels E, x, and y connected by arrows to depict the tree structure.]
The “smallness” of the tree assertion

\[
\text{tree}(E) \iff \text{if } E = \text{nil} \text{ then emp}
\]

\[
\text{else } \exists x, y. (E \mapsto l : x, r : y) \ast \text{tree}(x) \ast \text{tree}(y)
\]

\[
\text{tree}(E) \text{ is false of}
\]
The “smallness” of the tree assertion

\[
\text{tree}(E) \iff \begin{cases} 
\text{if } E = \text{nil} \text{ then } \text{emp} \\
\text{else } \exists x, y. (E \mapsto l: x, r: y) * \text{tree}(x) * \text{tree}(y)
\end{cases}
\]

and even false of
The AI Frame Problem
(McCarthy-Hayes, 1969)

- When you specify an action \{P}\text{act}\{Q\}, an inordinate amount of effort is needed to say what “act” DOESN’T do.

- \{ \text{not(holding(block))} \} \ pick-up(block) \{ \text{holding(block)} \} 

- \{ \text{holding(block2)} \} \ pick-up(block) \{ \text{holding(block2)} \} ???

Some Philosophical Problems from the Standpoint of Artificial Intelligence,
McCarthy-Hayes, Machine Intelligence, 1969
The AI Frame Problem
(McCarthty-Hayes, 1969)

- When you specify an action \{P\}act\{Q\}, an inordinate amount of effort is needed to say what “act” DOESN’T do.

  - \{ not(holding(block)) \} pick-up(block) \{ holding(block) \}
  - \{ holding(block2) \} pick-up(block) \{ holding(block2) \} ???
A Small Spec, and a Small Proof

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

- Proof of body of recursive procedure

\[[\text{tree}(i) \ast \text{tree}(j)]\]
\text{DispTree}(i);
\[[\text{emp} \ast \text{tree}(j)]\]
\text{DispTree}(j);
\[[\text{emp}]\]

\[
\frac{\{P\} C \{Q\}}{\{P \ast R\} C \{Q \ast R\}} \quad \text{Frame Rule}
\]
A Small Spec, and a Small Proof

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

- Proof of body of recursive procedure

\[
\begin{align*}
\text{tree}(i) \ast \text{tree}(j) \\
\text{DispTree}(i); \\
\text{emp} \ast \text{tree}(j) \\
\text{DispTree}(j); \\
\text{emp}
\end{align*}
\]

To automate we must infer frames during ``execution``

\[
\frac{\{P\} C \{Q\}}{\{P \ast R\} C \{Q \ast R\}} \quad \text{Frame Rule}
\]
Extensions of the entailment question I: Frame Inference

\[ A \vdash B \]
Extensions of the entailment question I: Frame Inference

\[ A \vdash B \ast ? \]
Extensions of the entailment question I: Frame Inference

\[
\text{tree}(i) * \text{tree}(j) \vdash \text{tree}(i) * ?
\]
Extensions of the entailment question I: Frame Inference

\[ \text{tree}(i) \ast \text{tree}(j) \vdash \text{tree}(i) \ast \text{tree}(j) \]
Extensions of the entailment question I: Frame Inference

\( x \neq \text{nil} \land \text{list}(x) \vdash \exists x'. x \mapsto x' * ? \)
Extensions of the entailment question I: Frame Inference

\[ x \neq \text{nil} \land \text{list}(x) \vdash \exists x'. x \leftrightarrow x' \ast \text{list}(x') \]
Extensions of the entailment question I: Frame Inference

\[ A \vdash B * ? \]
A Small Spec, and a Small Proof

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

- Proof of body of recursive procedure

  \[\text{tree}(i) \ast \text{tree}(j)\]
  \text{DispTree}(i);
  \[\text{emp} \ast \text{tree}(j)\]
  \text{DispTree}(j);
  \[\text{emp}\]

\[
\frac{\{P\} C\{Q\}}{\{P \ast R\} C\{Q \ast R\}} \quad \text{Frame Rule}
\]
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

\[ A \ast ? \vdash B \]

1Calcagno, Distefano, O’Hearn, Yang, POPL’09
Extensions of the entailment question II: abduction

\[ x \mapsfrom \text{nil} \ast ? \vdash \text{list}(x) \ast \text{list}(y) \]

- We call the ? here an “anti-frame”.\textsuperscript{1}

\textsuperscript{1}Calcagno, Distefano, O’Hearn, Yang, POPL’09
Extensions of the entailment question II: abduction

\[ x \mapsto \text{nil} \ast \text{list}(y) \vdash \text{list}(x) \ast \text{list}(y) \]

We call the \( ? \) here an “anti-frame”.\(^1\)

\(^1\)Calcagno, Distefano, O’Hearn, Yang, POPL’09
Extensions of the entailment question II: abduction

\[ x \leftrightarrow y \ast ? \vdash x \leftrightarrow a \ast \text{list}(a) \]

- We call the ? here an “anti-frame”.\(^1\)

---

\(^1\)Calcagno, Distefano, O’Hearn, Yang, POPL’09
Extensions of the entailment question II: abduction

\[ x \mapsto y \ast (y = a \land \text{list}(a)) \vdash x \mapsto a \ast \text{list}(a) \]

We call the ? here an “anti-frame”.\(^1\)

\(^1\)Calcagno, Distefano, O’Hearn, Yang, POPL’09
Abduction Example: Inferring a pre/post pair

```c
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:  
[list(x) * list(y)]foo(x, y)[list(x)]
Abduction Example: Inferring a pre/post pair

```c
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:

\[ [\text{list}(x) \ast \text{list}(y)] \text{foo}(x, y) [\text{list}(x)] \]
Abduction Example: Inferring a pre/post pair

1 void p(list-item *y) {
2     list-item *x;
3     x = malloc(sizeof(list-item));
4     x→tail = 0;
5     foo(x,y);
6     return(x); }

Abductive Inference:

Given Summary/spec: [list(x) * list(y)] foo(x, y)[list(x)]
Abduction Example: Inferring a pre/post pair

1 void p(list-item *y) {
2   list-item *x;
3   x = malloc(sizeof(list-item));
4   x->tail = 0;  \( x \mapsto 0 \)
5   foo(x,y);
6   return(x); }

Abductive Inference: \( x \mapsto 0 \ast ? \vdash \text{list}(x) \ast \text{list}(y) \)

Given Summary/spec: \([\text{list}(x) \ast \text{list}(y)] \text{foo}(x, y)[\text{list}(x)]\)
Abduction Example: Inferring a pre/post pair

1 void pxlist-item zyy {
2 list-item *x;
3 x = malloc(sizeof(list-item));
4 x→tail = 0;
5 foo(x,y);
6 return(x); }

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

Given Summary/spec: [list(x) \ast list(y)] foo(x, y)[list(x)]
Abduction Example: Inferring a pre/post pair

```c
1 void p(list-item *y) {
   2   list-item *x;
   3   x = malloc(sizeof(list-item));
   4   x->tail = 0;
   5   foo(x, y);
   6   return(x);
}
```

Abductive Inference: \( x \mapsto 0 \) \( \vdash \) \( \text{list}(y) \)

Given Summary/spec: \( [\text{list}(x) \ast \text{list}(y)] \) \( \text{foo}(x, y) \text{list}(x) \)
**Abduction Example: Inferring a pre/post pair**

```c
1 void p(list-item *y) {
2     list-item *x;
3     x = malloc(sizeof(list-item));
4     x->tail = 0;
5     foo(x,y);
6     return(x);
}
```

Abductive Inference:  
\[ x \leftrightarrow 0 \ast \text{list}(y) \vdash \text{list}(x) \ast \text{list}(y) \]

Given Summary/spec:  
\[ \text{list}(x) \ast \text{list}(y) \text{foo}(x, y) \text{list}(x) \]
Abduction Example: Inferring a pre/post pair

```c
1 void pxlist-item zyy {
2     emp listx y y
3     list-item zx;
4     x = malloc(sizeof(list-item));
5     x → tail = 0;
6     foo(x, y);
7     return(x); }
```

Abductive Inference: \( x \mapsto 0 \ast list(y) \vdash list(x) \ast list(y) \)

Given Summary/spec: \([list(x) \ast list(y)]foo(x, y)[list(x)]\)
Abduction Example: Inferring a pre/post pair

1 void p(list-item *y) {
2     list-item *x;
3     x = malloc(sizeof(list-item));
4     x->tail = 0;
5     foo(x, y);
6     return(x); }

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

Given Summary/spec:  \text{[list}(x) * \text{list}(y)] \text{foo}(x, y) \text{[list}(x)]
Bi-Abduction

\[ A \ast \texttt{?anti-frame} \vdash B \ast \texttt{?frame} \]

- 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

- 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

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

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

- 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...