

Space Invading Systems Code

LOPSTR'08 Invited Talk

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

with special thanks to our

SLAyer Colleagues (MSR): Josh Berdine, Byron Cook

and acknowledgment to **the work of Sagiv-Reps-Wilhelm et. al.**



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.



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.
- ▶ In some (distant?) future: automatically crash-proof Apache, OpenSSL...



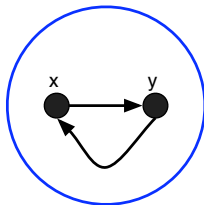
Part I

Basics



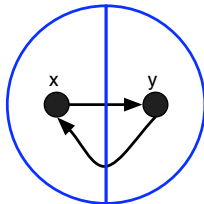
Separation Logic

$x \mapsto y * y \mapsto x$



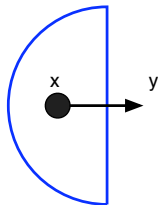
Separation Logic

$x \mapsto y * y \mapsto x$



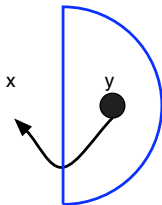
Separation Logic

$x \mapsto y$



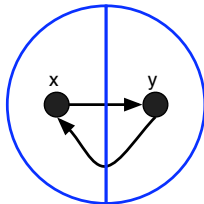
Separation Logic

$|y| \rightarrow x$



Separation Logic

$x \mapsto y * y \mapsto x$



In-place Reasoning

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

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

$[P] x = \text{cons}(a, b) [P * (x \mapsto a, b)] \quad (x \notin \text{free}(P))$

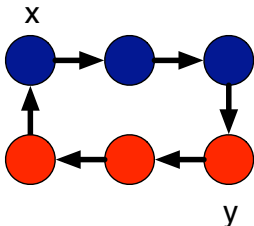


Simple Linked List Segments

List segments ($tl_list(E)$ is shorthand for $ls(E, nil)$)

$ls(E, F) \iff$ if $(E = F)$ then emp
else $\exists y. E \mapsto tl : y * ls(y, F)$

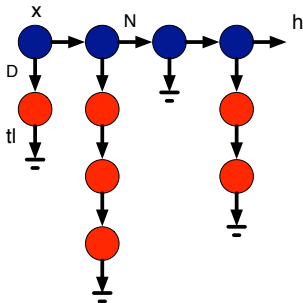
$ls(x, y) * ls(y, x)$



Example using Second-order Predicates

$\text{hls}(\phi, E, F) \iff \text{if } (E = F) \text{ then emp}$
 $\text{else } \exists y \exists d. E \mapsto (D:d, N:y) * \phi(d) * \text{hls}(y, F)$

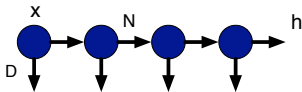
$\phi = \text{tl_list}$



Example using Second-order Predicates

$\text{hls}(\phi, E, F) \iff \text{if } (E = F) \text{ then emp}$
 $\text{else } \exists y \exists d. E \mapsto (D:d, N:y) * \phi(d) * \text{hls}(y, F)$

$\phi = \text{dangle} = (\lambda x. \text{emp})$



▶ $[hls(tl_list, x, h)]$

```
void p(x,h) {  
    if (x  $\neq$  h) {  
  
        dispose_list(x $\rightarrow$ D);  
  
        p(x $\rightarrow$ N,h);  
  
    }  
}
```

$[hls(dangle, x, h)]$

▶ Given Summary/Spec: $[tl_list(y)]$ dispose_list(y) $[dangle(y)]$



- ▶ $[hls(tl_list, x, h)]$

```
void p(x,h) {
    if (x  $\neq$  h) {
         $[x \mapsto (D: d', N: n') * tl\_list(d') * hls(tl\_list, n', h)]$ 
        dispose_list(x $\rightarrow$ D);

        p(x $\rightarrow$ N,h);
    }
}
[hls(dangle, x, h)]
```
- ▶ Given Summary/Spec: $[tl_list(y)]$ dispose_list(y) $[dangle(y)]$



► $[hls(tl_list, x, h)]$

```
void p(x,h) {  
    if (x  $\neq$  h) {  
         $[x \mapsto (D: d', N: n') * tl\_list(d') * hls(tl\_list, n', h)]$   
        dispose_list(x $\rightarrow$ D);  
         $[x \mapsto (D: d', N: n') * dangle(d') * hls(tl\_list, n', h)]$   
  
        p(x $\rightarrow$ N,h);  
  
    }  
}
```

$[hls(dangle, x, h)]$

► Given Summary/Spec: $[tl_list(y)]$ dispose_list(y) $[dangle(y)]$



▶ $[hls(tl_list, x, h)]$

```

void p(x,h) {
    if (x ≠ h) {
        [x ↦ (D: d', N: n') * tl_list(d') * hls(tl_list, n', h)]
        dispose_list(x→D);
        [x ↦ (D: d', N: n') * dangle(d') * hls(tl_list, n', h)]
        [x ↦ (D: d', N: n') * hls(tl_list, n', h)]
        p(x→N,h);
    }
}

```

$[hls(dangle, x, h)]$

▶ Given Summary/Spec: $[tl_list(y)]$ dispose_list(y) $[dangle(y)]$



- ▶ $[hls(tl_list, x, h)]$

```
void p(x,h) {
    if (x ≠ h) {
        [x ↦ (D: d', N: n') * tl_list(d') * hls(tl_list, n', h)]
        dispose_list(x→D);
        [x ↦ (D: d', N: n') * dangle(d') * hls(tl_list, n', h)]
        [x ↦ (D: d', N: n') * hls(tl_list, n', h)]
        p(x→N,h);
        [x ↦ (D: d', N: n') * hls(dangle, n', h)]
    }
}
[hls(dangle, x, h)]
```
- ▶ Given Summary/Spec: $[tl_list(y)]$ dispose_list(y) $[dangle(y)]$



*Baby Space Invader (2005-06)*¹

1. Proof search by abstract interpretation.
2. Symbolically executes statements using in-place reasoning.
3. Abstraction rules like

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

used to guarantee fixed-point convergence.

4. Uses its own toy input languages. Restricted to simple linked lists.

¹Distefano, O'Hearn, Yang, TACAS'06.

Builds on Smallfoot (Berdine, Calcagno, O'Hearn); Distefano's thesis; Sagiv, Reps, Wilhelm, TOPLAS'98.



An Invariant in Baby SpacInvader²

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

Calculated Loop Invariant

∨

∨

²Used necessarily nonempty version of list predicate



An Invariant in Baby SpacInvader²

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

Calculated Loop Invariant

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

∨

∨

²Used necessarily nonempty version of list predicate



An Invariant in Baby SpacInvader²

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

Calculated Loop Invariant

```
    x = nil ∧ emp
∨  x ↦ nil
∨
```

²Used necessarily nonempty version of list predicate



An Invariant in Baby SpacInvader²

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

Calculated Loop Invariant

```
    x = nil ∧ emp
∨ x ↦ nil
∨
```

²Used necessarily nonempty version of list predicate



An Invariant in Baby SpacInvader²

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

Calculated Loop Invariant

$x = \text{nil} \wedge \text{emp}$
 $\vee x \mapsto \text{nil}$
 $\vee \text{ls}(x, \text{nil})$

²Used necessarily nonempty version of list predicate



An Invariant in Baby SpacInvader²

```
{emp}
x=nil;
while (-){    x  $\mapsto$  x' * ls(x', nil)
    new(y);
    y ->tl = x;
    x=y;
}
```

Calculated Loop Invariant

```
x = nil  $\wedge$  emp
 $\vee$  x  $\mapsto$  nil
 $\vee$  ls(x, nil)
```

²Used necessarily nonempty version of list predicate



An Invariant in Baby SpacInvader²

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

Calculated Loop Invariant

$x = \text{nil} \wedge \text{emp}$
 $\vee x \mapsto \text{nil}$
 $\vee \text{ls}(x, \text{nil})$

²Used necessarily nonempty version of list predicate



Part II

Point of Departure



Question (May, 2006)

- ▶ Is it practically possible to automatically find Hoare logic (separation logic) proofs of data structure usage in substantial systems programs?



Pointer Safety

- ▶ More precisely, try to prove the generic property “pointer safety”

The program does not dereference null or a dangling pointer, or leak memory.



Pointer Safety

- ▶ More precisely, try to prove the generic property “pointer safety”

The program does not dereference null or a dangling pointer, or leak memory.

- ▶ Little to say, a lot to do



Recall

```
▶ void p(x,h) {  
    if (x ≠ h) {  
        dispose_list(x→D);  
        p(x→N,h);  
    }  
}
```



Recall

```
▶ void p(x,h) {  
    if (x ≠ h) {  
        dispose_list(x→D);  
        p(x→N,h);  
    }  
}
```

- ▶ What if we give this program an acyclic horizontal list not through h ?



Recall

```
▶ void p(x,h) {  
    if (x ≠ h) {  
        dispose_list(x→D);  
        p(x→N,h);  
    }  
}
```

- ▶ What if we give this program an acyclic horizontal list not through h ?
- ▶ Or a cyclic vertical list?



Initial Target: Device Drivers (summer 2006)

- ▶ Local inside knowledge (Byron Cook)
- ▶ Relatively small (<15K LOC)
- ▶ Relatively simple data structures: combinations of linked lists, not difficult sharing (like in OS kernel)



Initial Target: Device Drivers (summer 2006)

- ▶ Local inside knowledge (Byron Cook)
- ▶ Relatively small (<15K LOC)
- ▶ Relatively simple data structures: combinations of linked lists, not difficult sharing (like in OS kernel)

- ▶ **should be easy, right?**



Part III

Lift Off



A Situation

- ▶ 1394 firewire driver has approx 1K LOC in struct definitions.



A Situation

- ▶ 1394 firewire driver has approx 1K LOC in struct definitions.
- ▶ **Question:** what should the base predicates for the shape analysis be?
- ▶



A Situation

- ▶ 1394 firewire driver has approx 1K LOC in struct definitions.
- ▶ **Question:** what should the base predicates for the shape analysis be?
- ▶ **Hint:** I sort of already told you.



A Situation

- ▶ 1394 firewire driver has approx 1K LOC in struct definitions.
- ▶ **Question:** what should the base predicates for the shape analysis be?
- ▶ **Answer:** Use higher-order predicates, where the parameters allow the analysis to adapt to a given structure



Higher-order lists, again (slightly differently)



$$ls(\phi, e, f) \iff \phi(e, f) \vee (\exists y'. \phi(e, y') * ls(y', f))$$



Higher-order lists, again (slightly differently)



$$ls(\phi, e, f) \iff \phi(e, f) \vee (\exists y'. \phi(e, y') * ls(y', f))$$

▶ For an example, let

- ▶ ϕ_f be the predicate $\lambda xy. x \mapsto f : y$
- ▶ ϕ_g be the predicate $\lambda xy. x \mapsto g : y$



Higher-order lists, again (slightly differently)



$$ls(\phi, e, f) \iff \phi(e, f) \vee (\exists y'. \phi(e, y') * ls(y', f))$$

▶ For an example, let

- ▶ ϕ_f be the predicate $\lambda xy. x \mapsto f : y$
- ▶ ϕ_g be the predicate $\lambda xy. x \mapsto g : y$

▶ Then,

$$(x \mapsto f : y', g : z') * ls(\phi_f, y', x) * ls(\phi_g, z', x)$$

describes two circular lists with common head node, where one list links through f and the other links through g .



Higher-order lists, again (slightly differently)



$$ls(\phi, e, f) \iff \phi(e, f) \vee (\exists y'. \phi(e, y') * ls(y', f))$$

▶ For an example, let

- ▶ ϕ_f be the predicate $\lambda xy. x \mapsto f : y$
- ▶ ϕ_g be the predicate $\lambda xy. x \mapsto g : y$

▶ Then,

$$(x \mapsto f : y', g : z') * ls(\phi_f, y', x) * ls(\phi_g, z', x)$$

describes two circular lists with common head node, where one list links through f and the other links through g .

- ▶ Higher-order predicates allow us to describe **many complex variations** on simple linked lists.



Analysis with higher-order predicates

- ▶ An abstract domain using *-combinations of $ls(\phi, e, f)$ predicates
- ▶ Discovers candidate ϕ 's on-the-fly by scrutinizing linking structure in the heap; a certain heuristic based on graph isomorphism.
- ▶ Applied to firewire and (later) to other drivers.
- ▶ Ideas implemented in two tools: [Space Invader](#), [SLayer](#)



CAV'07 results: 1394 (firewire) Driver Routines³
(Not the entire driver)

Routine	LOC	Space (Mb)	Time (sec)
t1394_BusResetRoutine	718	322.44	663
t1394Diag_Cancellrp	697	263.45	724
t1394_GetAddressD	698	342.59	1036
t1394_SetAddressD	694	311.87	956
t1394Diag_PnpRemoveDevice	1885	>2000.00	T/O

10 *isomorphic* dereference errors found.

Table refers to pointer safety proofs, after bugs fixed

³CAV'07 paper of Berdine, Calcagno, Cook, Distefano, O'Hearn, Yang, Wies



Part III

Acceleration



Problem (Yang's obsession)

- ▶ Analysis has **thousands** of abstract states at program points.

⁴“usually” leads to only one state. We know that “perfect” join is too imprecise.



Problem (Yang's obsession)

- ▶ Analysis has **thousands** of abstract states at program points.
- ▶ **O'Hearn to Yang:** *Let me write specs.* You should usually have only one state (based on hand-proof intuition).

⁴“usually” leads to only one state. We know that “perfect” join is too imprecise.



Problem (Yang's obsession)

- ▶ Analysis has **thousands** of abstract states at program points.
- ▶ **O'Hearn to Yang:** *Let me write specs.* You should usually have only one state (based on hand-proof intuition).
- ▶ **Yang to O'Hearn:** Oh, you want the **human to do join?**

⁴“usually” leads to only one state. We know that “perfect” join is too imprecise.



Problem (Yang's obsession)

- ▶ Analysis has **thousands** of abstract states at program points.
- ▶ **O'Hearn to Yang:** *Let me write specs.* You should usually have only one state (based on hand-proof intuition).
- ▶ **Yang to O'Hearn:** Oh, you want the **human to do join?**
- ▶ **Join.** Given A_1, \dots, A_n , find “smaller” B where

$$A_1 \vee \dots \vee A_n \implies B$$

Can lead to imprecision (no proof). Always a balancing act.

⁴“usually” leads to only one state. We know that “perfect” join is too imprecise.



Problem (Yang's obsession)

- ▶ Analysis has **thousands** of abstract states at program points.
- ▶ **O'Hearn to Yang:** *Let me write specs.* You should usually have only one state (based on hand-proof intuition).
- ▶ **Yang to O'Hearn:** Oh, you want the **human to do join?**
- ▶ **Join.** Given A_1, \dots, A_n , find “smaller” B where

$$A_1 \vee \dots \vee A_n \implies B$$

Can lead to imprecision (no proof). Always a balancing act.

- ▶ **Question:** is there a “near perfect” join⁴ which retains enough precision to prove pointer safety?

⁴“usually” leads to only one state. We know that “perfect” join is too imprecise.



Recall:

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

Calculated Loop Invariant

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

$\vee x \mapsto \text{nil}$

$\vee \text{ls}(x, \text{nil})$



Recall:

```
{emp}  
x=nil;  
while (-){  
    new(y);  
    y->t1 = x;  
    x=y;  
}
```

Calculated Loop Invariant

PossiblyEmptyLseg(x, nil) ; (*smarter version*)



The Impact of Join (中)

Program	NO JOIN	JOIN
onelist_create.c	3	1
twolist_create.c	9	1
firewire_create.c	3969	1



Some Properties of \uparrow

- ▶ Both nonempty ($/s\ NE$) and possibly empty ($/s\ PE$) lists.
- ▶ Promote \mapsto to NE:

$$\begin{aligned} & y \mapsto \text{nil} \quad * \quad (/s\ NE\ x\ \text{nil}) \\ \uparrow \quad (/s\ NE\ y\ \text{nil}) \quad * \quad x \mapsto \text{nil} \\ = \quad & (/s\ NE\ y\ \text{nil}) \quad * \quad (/s\ NE\ x\ \text{nil}) \end{aligned}$$

Similarly, can promote NE to PE.

- ▶ Join is **partial**:

$$(y \mapsto -) \uparrow (y = -1) = \text{undef}$$

- ▶ Join must **make discoveries about possibly empty lists**.

$$\begin{aligned} & y = \text{nil} \wedge (/s\ NE\ x\ y) \quad \uparrow \quad x \mapsto y \quad * \quad (/s\ NE\ y\ 0) \\ = \quad & (/s\ NE\ x\ y) \quad * \quad (/s\ PE\ y\ \text{nil}) \end{aligned}$$



Proofs for Device Drivers

Program	LOC	Sec	Mb
pci-driver.c	2532	0.75	3.19
cdrom.c	6218	91.45	84.79
t1394Diag.c	10240	137.78	73.24
md.c	6635	1819.53	1010.81
ll_rw_blk.c	5469	947.20	511.43

> 60 bugs found in these drivers. We inserted our own fixes. Then..

Pointer Safety proofs found for fixed drivers

CAV'08 paper of Yang, Lee, Calcagno, Distefano, Berdine, Cook, O'Hea



Some Caveats

- ▶ Assume array dereferences $a[i]$ are in bounds.

Pointer safety is less than memory safety.

a precision rather than soundness issue (for pointer safety). But...

- ▶ Changed small part of 1394 due to inability (presently) to deal precisely+efficiently with arrays of pointers, when size of arrays is unknown (made array of size 1).
- ▶ Ignores concurrency.
- ▶ Some rewriting of doubly-linked lists to singly-linked.



- ▶ **Question:** Is it practically possible to automatically find Hoare logic (separation logic) proofs of data structure usage in substantial systems programs?



- ▶ **Question:** Is it practically possible to automatically find Hoare logic (separation logic) proofs of data structure usage in substantial systems programs?
- ▶ **Answer:** Yes. In some programs, anyhow.



Part IV

Abduction

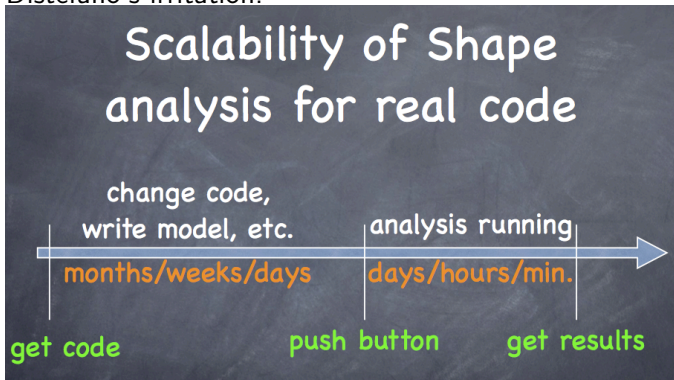


Space Invader? Abduction?



Context

- ▶ Distefano's irritation:



- ▶ Calcagno's embarrassment: Come on, only 10K LOC?



Trying to do something about it...⁵

- ▶ Frege's principle of compositionality

The meaning of a composite phrase can be defined in terms of the meanings of its parts

- ▶ **Desire:** generate Hoare triples without knowing context

⁵Footprint Analysis, SAS'07; Compositional Shape Analysis, 2008 Tech Rept.



Trying to do something about it...⁵

- ▶ Frege's principle of compositionality

The meaning of a composite phrase can be defined in terms of the meanings of its parts

- ▶ **Desire:** generate Hoare triples without knowing context
- ▶ **Problem:** What specs to aim for? (enormous abstract domain)

⁵Footprint Analysis, SAS'07; Compositional Shape Analysis, 2008 Tech Rept.



Trying to do something about it...⁵

- ▶ Frege's principle of compositionality

The meaning of a composite phrase can be defined in terms of the meanings of its parts

- ▶ **Desire:** generate Hoare triples without knowing context
- ▶ **Problem:** What specs to aim for? (enormous abstract domain)
- ▶ **Answer:** Go for the footprint.

⁵Footprint Analysis, SAS'07; Compositional Shape Analysis, 2008 Tech Rept.



Trying to do something about it...⁵

- ▶ Frege's principle of compositionality

The meaning of a composite phrase can be defined in terms of the meanings of its parts

- ▶ **Desire:** generate Hoare triples without knowing context
- ▶ **Problem:** What specs to aim for? (enormous abstract domain)
- ▶ **Answer:** Go for the footprint.
- ▶ **How?:** New technique, [abductive inference for separation logic](#).

⁵Footprint Analysis, SAS'07; Compositional Shape Analysis, 2008 Tech Rept.



Footprints

- ▶ **Intuition:** If our analysis can figure out what cells a program accesses (or an approximation), we won't have to consider *all* assertions.



Footprints

- ▶ **Intuition:** If our analysis can figure out what cells a program accesses (or an approximation), we won't have to consider *all* assertions.
- ▶ Like in our example

```
[hls(tl_list, x, h)]  
void p(x,h) {  
    if (x ≠ h) {  
        dispose_list(x→D);  
        p(x→N,h);  
    }  
}  
[hls(dangle, x, h)]
```

- ▶ Given Summary/Spec: `[tl_list(y)] dispose_list(y) [dangle(y)]`



The Abductive Inference Question



$$A * ? \vdash B$$

⁶Calcagno, Distefano, O'Hearn, Yang, 2008 (forthcoming)



The Abductive Inference Question



$$x \mapsto \text{nil} * ? \vdash \text{list}(x) * \text{list}(y)$$

- ▶ We call the ? here an “anti-frame”.⁶

⁶Calcagno, Distefano, O’Hearn, Yang, 2008 (forthcoming)



The Abductive Inference Question



$$x \mapsto \text{nil} * \textit{list}(y) \vdash \textit{list}(x) * \textit{list}(y)$$

⁶Calcagno, Distefano, O'Hearn, Yang, 2008 (forthcoming)



The Abductive Inference Question



$$x \mapsto y * ? \vdash x \mapsto a * \text{list}(a)$$

⁶Calcagno, Distefano, O'Hearn, Yang, 2008 (forthcoming)



The Abductive Inference Question



$$x \mapsto y * (y = a \wedge list(a)) \vdash x \mapsto a * list(a)$$

⁶Calcagno, Distefano, O'Hearn, Yang, 2008 (forthcoming)



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   merge(x,y);  
6   return(x); }
```

Abductive Inference:

Given Summary/spec: $[list(x) * list(y)]merge(x, y)[list(x)]$



Abduction Example: Inferring a pre/post pair

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

Abductive Inference:

Given Summary/spec: $[list(x) * list(y)]merge(x, y)[list(x)]$



Abduction Example: Inferring a pre/post pair

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

Abductive Inference:

Given Summary/spec: $[list(x) * list(y)]merge(x, y)[list(x)]$



Abduction Example: Inferring a pre/post pair

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

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

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



Abduction Example: Inferring a pre/post pair

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

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

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



Abduction Example: Inferring a pre/post pair

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

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

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



Abduction Example: Inferring a pre/post pair

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

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

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



Abduction Example: Inferring a pre/post pair

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

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

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



Abduction Example: Inferring a pre/post pair

1 void p(list-item *y) {	emp	list(y)(Inferred Pre)
2 list-item *x;		
3 x=malloc(sizeof(list-item));		
4 x→tail = 0;	x ↦ 0	
5 merge(x,y);	list(x)	
6 return(x); }		list(ret)(Inferred Post)

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

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



Earlier Example

- ▶ Bottom up analysis on

```
void p(x,h) {  
    if (x ≠ h) {  
        dispose_list(x→D);  
        p(x→N,h); }  
}
```

- ▶ First **finds** (essentially) the inner Summary:
[tl_list(y)] dispose_list(y) *[dangle(y)]*



Earlier Example

- ▶ Bottom up analysis on

```
void p(x,h) {  
    if (x ≠ h) {  
        dispose_list(x→D);  
        p(x→N,h); }  
}
```

- ▶ First **finds** (essentially) the inner Summary:
[*tl_list*(y)] dispose_list(y) [*dangle*(y)]
- ▶ Then it finds: [*hls*(*tl_list*, x, h)] p(x, h) [*hls*(*dangle*, x, h)]



On firewire driver

- ▶ Finds consistent specs for 121 out of 121 procedures.
- ▶ Discovered spec for one top-level procedure has more generality than hand-written one from CAV'08 case study.
- ▶ **Caveat:** Changed the code a bit, to deal with a problem with head-pointers which is easier to analyze top-down than bottom-up.



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

Caveats!: NO CONCURRENCY. The results are **partial**. **Timeout is involved**. **Many small procedures**. We are working on *improving the percentages*. Etc...



Lessons

- ▶ **Q:** Thousands of struct dfns, which shape predicates to use?
A: Go higher order (and make analysis adapt).



Lessons

- ▶ **Q:** Thousands of struct dfns, which shape predicates to use?
A: Go higher order (and make analysis adapt).
- ▶ **Q:** Thousands of states at program points, what to do (without losing too much precision)?
A: Pointer-safety admits a “near-perfect” *partial* join



Lessons

- ▶ **Q:** Thousands of struct dfns, which shape predicates to use?
A: Go higher order (and make analysis adapt).
- ▶ **Q:** Thousands of states at program points, what to do (without losing too much precision)?
A: Pointer-safety admits a “near-perfect” *partial* join
- ▶ **Q:** Compositionality would be nice, how to get it?
A: Aim for footprint, use abductive inference

