Compositional Shape Analysis by means of Bi-Abduction

Cristiano Calcagno
Imperial College London
Visiting ETH

joint work with
Dino Distefano, Peter O'Hearn, Hongseok Yang

Separation Logic Tools and Projects

- Smallfoot (London)
- Space Invader (London)
- Slayer (Microsoft)
- SmallfootRG (Cambridge,London)
- Hip + Sleek (Singapore)
- JStar (Cambridge,London) one of EVE's back-ends
- Verifast + Vericool (Leuven)
- Thor (CMU)
- Heap-Hop (Paris,London)
- Xisa (Berkley,Colorado,Paris)
- HTT + Ynot (Harvard)
- Holfoot (Cambridge)
- Concurrent C Minor (Princeton,Paris,Singapore, Kansas)
- Compcert Project (INRIA)
- Flint (Yale)
- Certified Verifier (Yokyo)

A Story...

Great! I've got a preprocessed version of the Linux File System!....
and now what do I do with it?

Problems with real code

- Write a `main()`/environment code (not trivial!)
- Must we wait to have a super domain for the entire Linux?
- Can we say something for pointer manipulation when we have big code?

...and several others

Problems with real code

Need to handle incomplete code

- Write a `main()`/environment code (not trivial!)
- Must we wait to have a super domain for the entire Linux?
- Can we say something for pointer manipulation when we have big code?

...and several others

Problems with real code

Need to handle incomplete code

- Write a `main()`/environment code (not trivial!)

Start with something partial

- Must we wait to have a super domain for the entire Linux?

- Can we say something for pointer manipulation when we have big code?

...and several others

Problems with real code

Need to handle incomplete code

- Write a `main()`/environment code (not trivial!)

Start with something partial

- Must we wait to have a super domain for the entire Linux?

Need very high modularity

- Can we say something for pointer manipulation when we have big code?

...and several others

Facts about shape analyses

- Lots of real code out there uses pointer manipulation
- Shape analyses discover deep properties about the heap (eg a variable points to a cyclic/acyclic doubly linked list,...)
- Shape analyses are very expensive (hard to scale)
- Up to 2007 mostly applied to small toy programs (<100 LOC)

Facts about shape analyses

- Lots of real code out there uses pointer manipulation
- Shape analyses discover deep properties about the heap (eg a variable points to a cyclic/acyclic doubly linked list,...)
- Shape analyses are very expensive (hard to scale)
- Up to 2007 mostly applied to small toy programs (<100 LOC)

Our response:
compositional Space Invader

- ✓ Handles incomplete code
- ✓ Admits partial results
- ✓ Modular

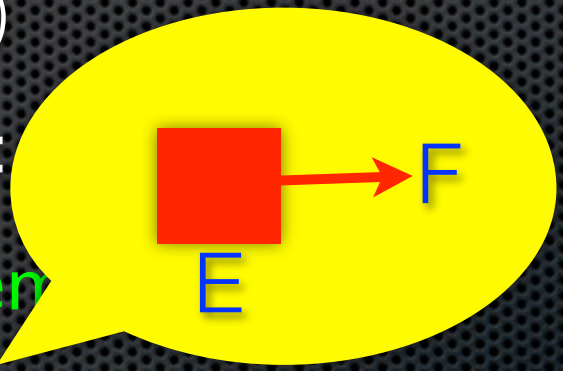
Basics

Notation

- Separation Logic's formulae to represent program states (Symbolic Heaps)
- Some useful predicates:
  - The empty heap: emp
  - An allocated cell: $E \mapsto F$
  - A "complete" list: list(E)

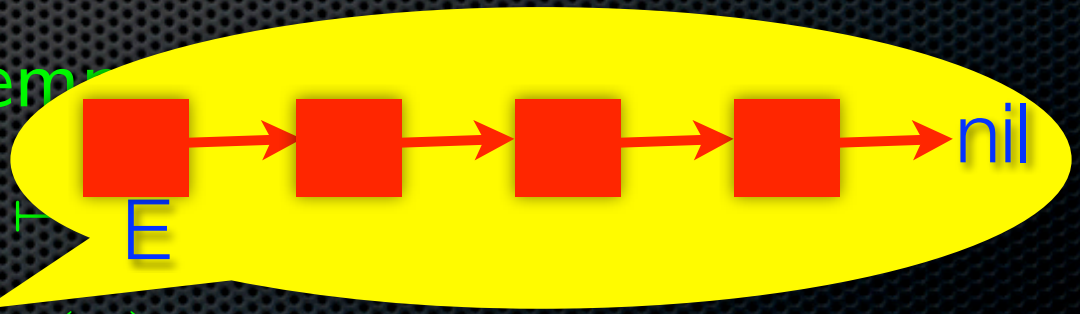
Notation

- Separation Logic's formulae to represent program states (Symbolic Heaps)
- Some useful predicates:
  - The empty heap: emp
  - An allocated cell: $E \mapsto F$
  - A "complete" list: list(E)



Notation

- Separation Logic's formulae to represent program states (Symbolic Heaps)
- Some useful predicates:
  - The empty heap: emp
  - An allocated cell: $E \mapsto$
  - A "complete" list: list(E)



Notation

- Separation Logic's formulae to represent program states (Symbolic Heaps)
- Some useful predicates:
  - The empty heap: emp
  - An allocated cell: $E \mapsto F$
  - A "complete" list: list(E)

Separating conjunction

- Separating conjunction combines spatial predicates

$$P * Q$$

- in words: P and Q hold for disjoint portion of memory

Separating conjunction

- Separating conjunction combines spatial predicates

P * Q

- in words: P and Q hold for disjoint portion of memory

Example: $x \mapsto y * y \mapsto x$

is satisfied by

Small specs

- Small specs encourage local reasoning and help to get small proofs
- When proving code involving procedures we use only their footprint

Example: use of small specs in proofs

```
{list(l1)*list(l2)}
Dispose(l1);

Dispose(l2);
```

Spec: `{list(l)} Dispose(l) {emp}`

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

Example: use of small specs in proofs

```
{list(l1)*list(l2)}
Dispose(l1);

Dispose(l2);
```

Spec: `{list(l)} Dispose(l) {emp}`

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

Example: use of small specs in proofs

```
{list(l1)*list(l2)}
Dispose(l1);
{emp*list(l2)}
Dispose(l2);
```

Spec: `{list(l)} Dispose(l) {emp}`

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

Example: use of small specs in proofs

```
{list(l1)*list(l2)}
Dispose(l1);
{list(l2)}
Dispose(l2);
```

Spec: `{list(l)} Dispose(l) {emp}`

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

Example: use of small specs in proofs

```
{list(l1)*list(l2)}
Dispose(l1);
{list(l2)}
Dispose(l2);
{emp}
```

Spec: `{list(l)} Dispose(l) {emp}`

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

Novelties

Frame Inference

```
{list(l1)*list(l2)}
Dispose(l1);

Dispose(l2);
```

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

Spec: `{list(l)} Dispose(l) {emp}`

Frame Inference

```
{list(l1)*list(l2)}
Dispose(l1);

Dispose(l2);
```

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

Spec: `{list(l)} Dispose(l) {emp}`

- To use the Frame Rule we must first compute R
- Frame inference problem: given A and B, compute X such that $A \vdash B * X$

Frame Inference

```
{list(l1)*list(l2)}
Dispose(l1);

Dispose(l2);
```

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

Spec: `{list(l)} Dispose(l) {emp}`

- To use the Frame Rule we must first compute R
- Frame inference problem: given A and B, compute X such that $A \vdash B * X$

Example:

`list(l1)*list(l2)` $\vdash$ `list(l1)* X`

Frame Inference

```
{list(l1)*list(l2)}
Dispose(l1);

Dispose(l2);
```

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

Spec: `{list(l)} Dispose(l) {emp}`

- To use the Frame Rule we must first compute R
- Frame inference problem: given A and B, compute X such that $A \vdash B * X$

Example:

$$\texttt{list(l1)*list(l2)} \vdash \texttt{list(l1)*list(l2)}$$

Frame Inference

```
{list(l1)*list(l2)}
Dispose(l1);

Dispose(l2);
```

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

Spec: `{list(l)} Dispose(l) {emp}`

- To use the Frame Rule we must first compute R
- Frame inference problem: given A and B, compute X such that $A \vdash B * X$

Example:

`list(l1)*list(l2)` $\vdash$ `list(l1)*list(l2)`

Frame Inference

```
{list(l1)*list(l2)}
Dispose(l1);
{emp*list(l2)}
Dispose(l2);
```

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

Spec: `{list(l)} Dispose(l) {emp}`

- To use the Frame Rule we must first compute R
- Frame inference problem: given A and B, compute X such that $A \vdash B * X$

Example:

$$\texttt{list(l1)*list(l2)} \vdash \texttt{list(l1)*list(l2)}$$

Abduction

Inference of explanatory hypotheses (introduced by C. Peirce in early 1900)

Abduction Inference:

given A and B compute X such that $A * X \vdash B$

Abduction

Inference of explanatory hypotheses (introduced by C. Peirce in early 1900)

Abduction Inference:
given A and B compute X such that $A * X \vdash B$

Example:

Spec: `{list(l1)*list(l2)}` `Dispose_Two_Lists(l1,l2)` `{emp}`

`list(l1)`

Abduction

Inference of explanatory hypotheses (introduced by C. Peirce in early 1900)

Abduction Inference:

given A and B compute X such that $A * X \vdash B$

Example:

Spec: `{list(l1)*list(l2)} Dispose_Two_Lists(l1,l2) {emp}`

`list(l1)*X ⊢ list(l1)*list(l2)`

Abduction

Inference of explanatory hypotheses (introduced by C. Peirce in early 1900)

Abduction Inference:

given A and B compute X such that $A * X \vdash B$

Example:

Spec: `{list(l1)*list(l2)} Dispose_Two_Lists(l1,l2) {emp}`

`list(l1)*list(l2)` $\vdash$ `list(l1)*list(l2)`

Abduction is not enough

If heaps A and B are incomparable abduction and frame inference alone are not enough.

We need to synthesize both missing portion of state and leftover portion of state

Heap A

$$x \mapsto y \quad * \quad y \mapsto y'$$

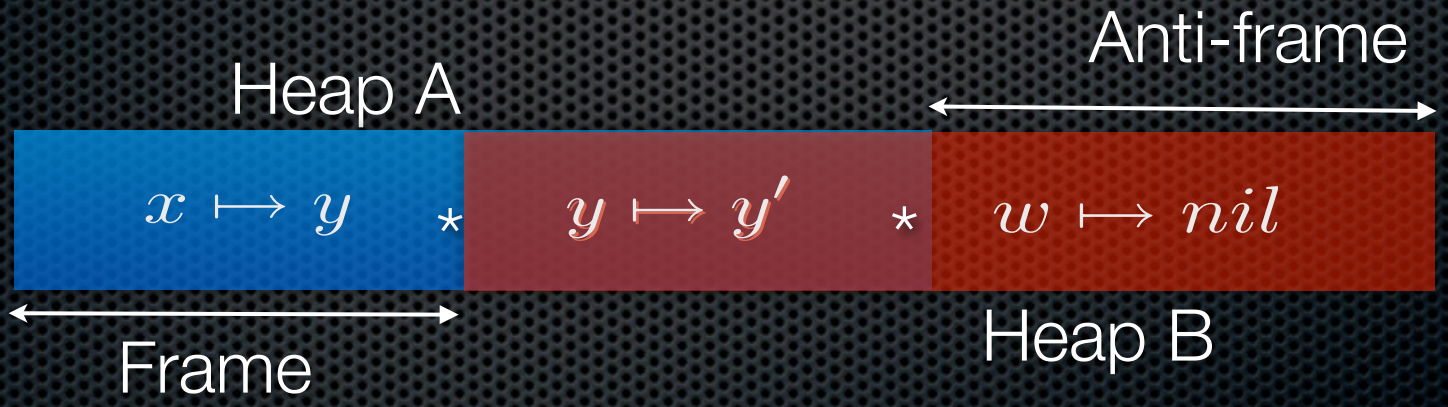
$$y \mapsto y' \quad * \quad w \mapsto nil$$

Heap B

Abduction is not enough

If heaps A and B are incomparable abduction and frame inference alone are not enough.

We need to synthesize both missing portion of state and leftover portion of state



Bi-abduction

Synthesizing both missing portion of state (anti-frame) and leftover portion of state (frame) requires a new notion

Bi-abduction:

given A and B compute ?antiframe and ?frame such that

$$A * ?antiframe \vdash B * ?frame$$

Bi-abduction

Synthesizing both missing portion of state (anti-frame) and leftover portion of state (frame) requires a new notion

Bi-abduction:

given A and B compute ?antiframe and ?frame such that

$$A * ?antiframe \vdash B * ?frame$$

Example:

$$x \mapsto 0 * z \mapsto 0 * ?antiframe \vdash \mathsf{list}(x) * \mathsf{list}(y) * ?frame$$

Bi-abduction

Synthesizing both missing portion of state (anti-frame) and leftover portion of state (frame) requires a new notion

Bi-abduction:

given A and B compute ?antiframe and ?frame such that

$$A * ?antiframe \vdash B * ?frame$$

Example:

$$x \mapsto 0 * z \mapsto 0 * \mathsf{list}(y) \vdash \mathsf{list}(x) * \mathsf{list}(y) * z \mapsto 0$$

Bi-abduction

Synthesizing both missing portion of state (anti-frame) and leftover portion of state (frame) requires a new notion

Bi-abduction:

given A and B compute ?antiframe and ?frame such that

$$A * ?antiframe \vdash B * ?frame$$

Example:

$$x \mapsto 0 * z \mapsto 0 * \mathsf{list}(y) \vdash \mathsf{list}(x) * \mathsf{list}(y) * z \mapsto 0$$

Our POPL'09 paper describes a theorem prover for bi-abduction

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail =  0;
4    z=malloc(sizeof(list_item)); z->tail =  0;
5    foo(x,y);
6    foo(x,z);
7  }
```

emp

emp

Bi-abductive prover

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail =  0;
4    z=malloc(sizeof(list_item)); z->tail =  0;
5    foo(x,y);
6    foo(x,z);
7  }
```

emp

emp

$x \mapsto 0$

Bi-abductive prover

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail =  0;
4    z=malloc(sizeof(list_item)); z->tail =  0;
5    foo(x,y);
6    foo(x,z);
7  }
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$

Bi-abductive prover

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

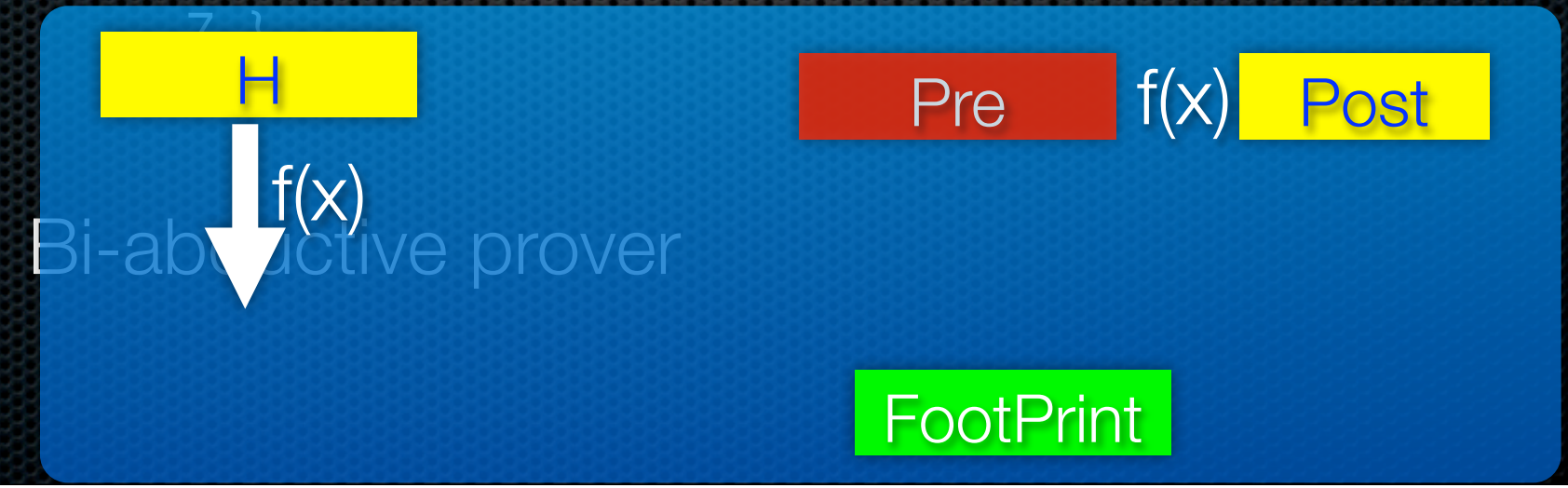
```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail =  0;
4    z=malloc(sizeof(list_item)); z->tail =  0;
5    foo(x,y);
6    foo(x,z);
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$



Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

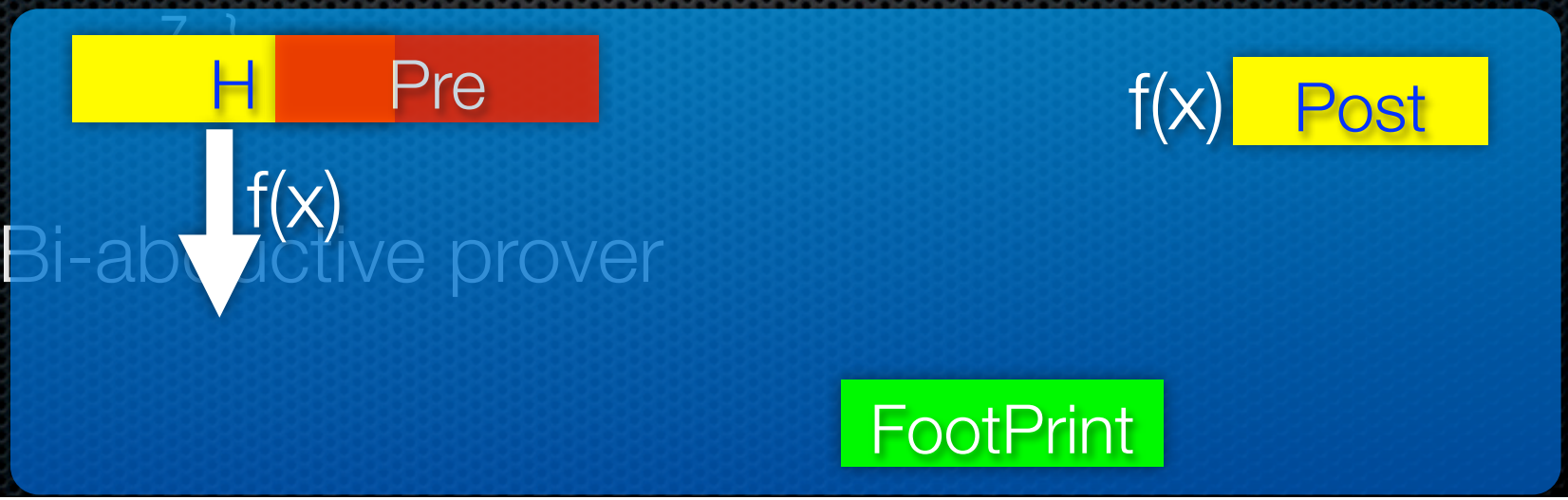
```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail =  0;
4    z=malloc(sizeof(list_item)); z->tail =  0;
5    foo(x,y);
6    foo(x,z);
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$



Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

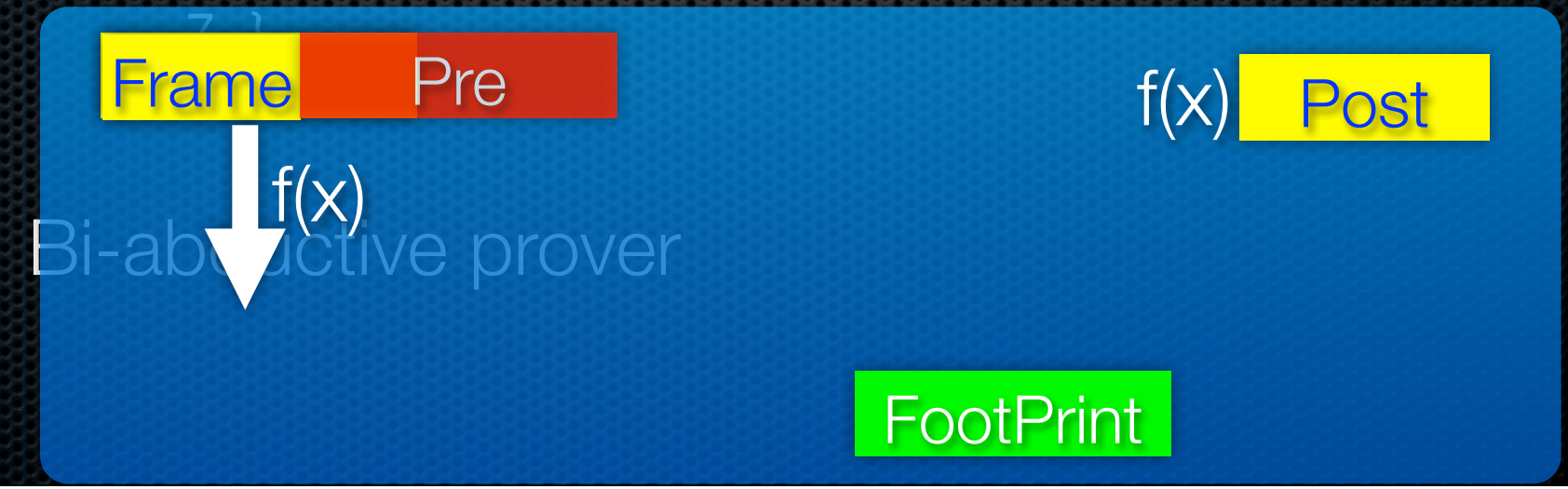
```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail =  0;
4    z=malloc(sizeof(list_item)); z->tail =  0;
5    foo(x,y);
6    foo(x,z);
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$



Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

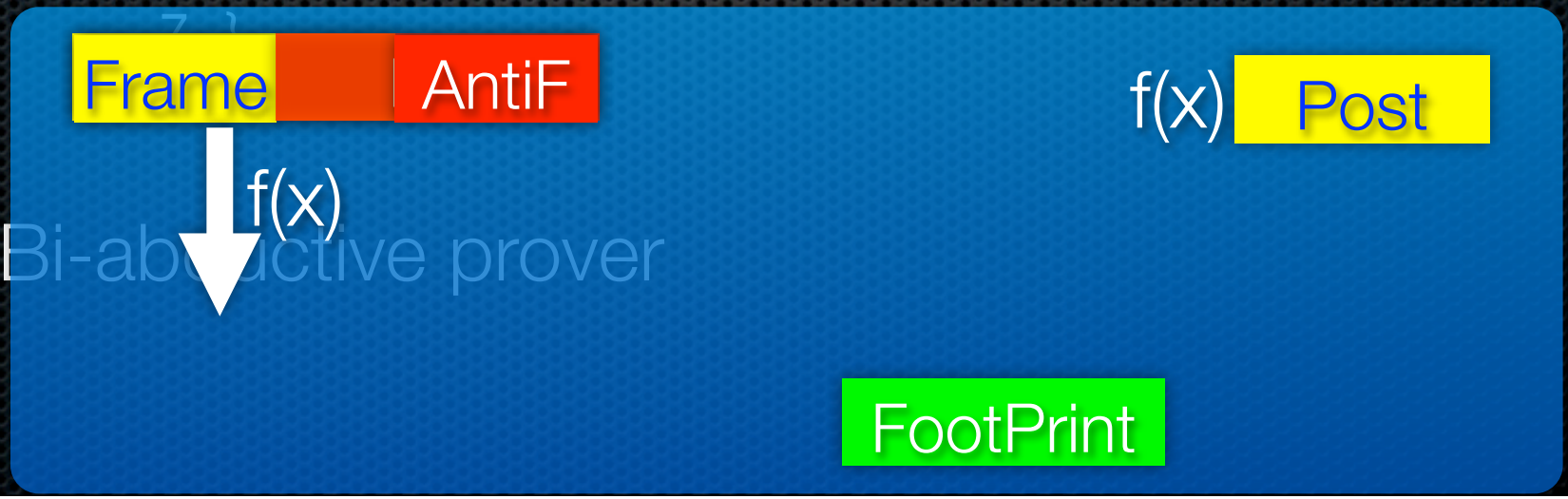
```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail =  0;
4    z=malloc(sizeof(list_item)); z->tail =  0;
5    foo(x,y);
6    foo(x,z);
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$



Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

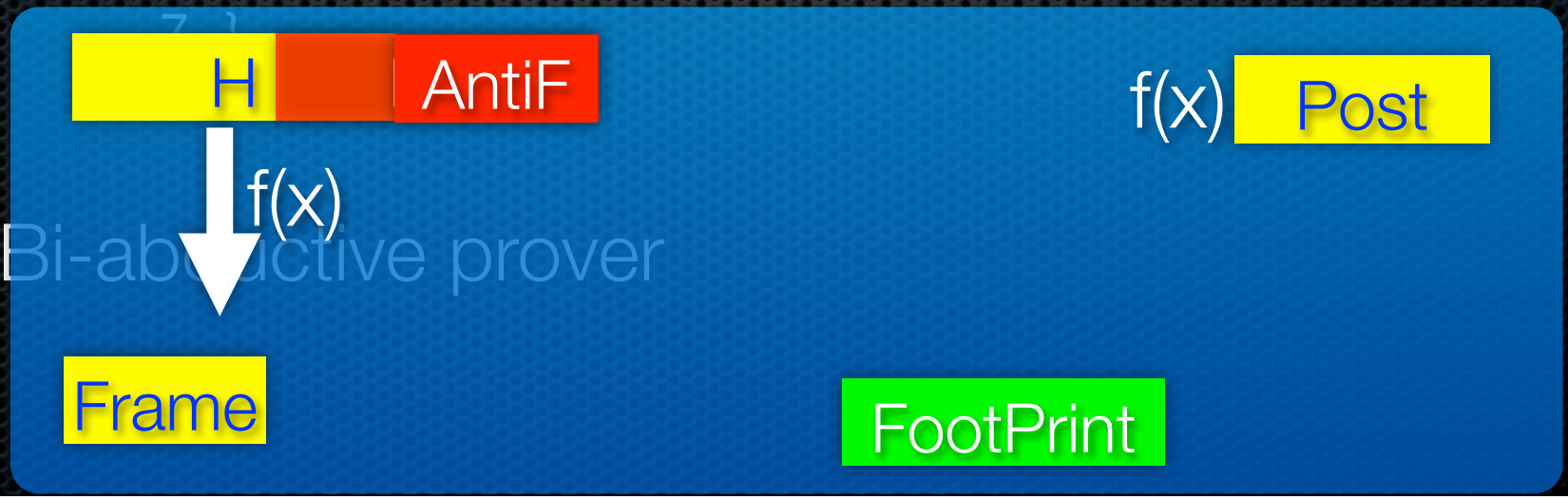
```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail =  0;
4    z=malloc(sizeof(list_item)); z->tail =  0;
5    foo(x,y);
6    foo(x,z);
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$



Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

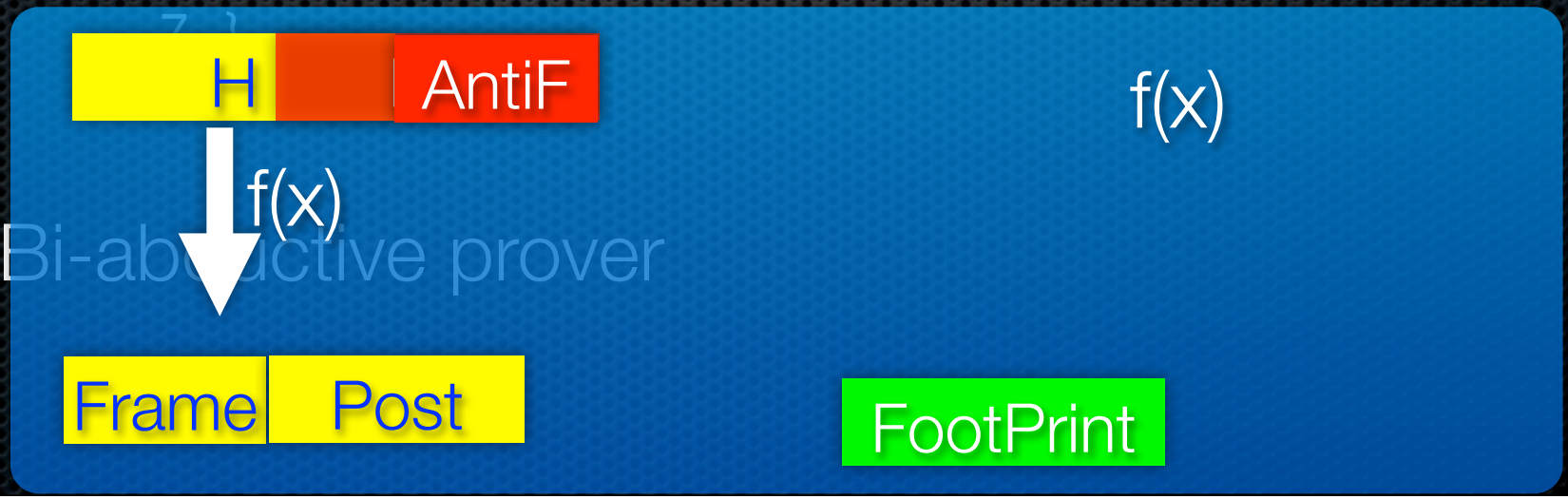
```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail =  0;
4    z=malloc(sizeof(list_item)); z->tail =  0;
5    foo(x,y);
6    foo(x,z);
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$



Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail =  0;
4    z=malloc(sizeof(list_item)); z->tail =  0;
5    foo(x,y);
6    foo(x,z);
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail =  0;
4    z=malloc(sizeof(list_item)); z->tail =  0;
5    foo(x,y);
6    foo(x,z);
7  }
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$

Bi-abductive prover

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail =  0;
4    z=malloc(sizeof(list_item)); z->tail =  0;
5    foo(x,y);
6    foo(x,z);
7  }
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$

Bi-abductive prover

$$x \mapsto 0 * z \mapsto 0 * ?antiframe \vdash \mathsf{list}(x) * \mathsf{list}(y) * ?frame$$

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail =  0;
4    z=malloc(sizeof(list_item)); z->tail =  0;
5    foo(x,y);
6    foo(x,z);
7  }
```

emp

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$

Bi-abductive prover

$$x \mapsto 0 * z \mapsto 0 * \mathsf{list}(y) \vdash \mathsf{list}(x) * \mathsf{list}(y) * z \mapsto 0$$

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail =  0;
4    z=malloc(sizeof(list_item)); z->tail =  0;
5    foo(x,y);
6    foo(x,z);
7  }
```

$\mathsf{list}(y)$

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$

$\mathsf{list}(x) * z \mapsto 0$

Bi-abductive prover

$$x \mapsto 0 * z \mapsto 0 * \mathsf{list}(y) \vdash \mathsf{list}(x) * \mathsf{list}(y) * z \mapsto 0$$

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail =  0;
4    z=malloc(sizeof(list_item)); z->tail =  0;
5    foo(x,y);
6    foo(x,z);
7  }
```

$\mathsf{list}(y)$

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$

$\mathsf{list}(x) * z \mapsto 0$

Bi-abductive prover

$$\mathsf{list}(x) * z \mapsto 0 * ?antiframe \vdash \mathsf{list}(x) * \mathsf{list}(z) * ?frame$$

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail =  0;
4    z=malloc(sizeof(list_item)); z->tail =  0;
5    foo(x,y);
6    foo(x,z);
7  }
```

$\mathsf{list}(y)$

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$

$\mathsf{list}(x) * z \mapsto 0$

Bi-abductive prover

$$\mathsf{list}(x) * z \mapsto 0 * \mathsf{emp} \vdash \mathsf{list}(x) * \mathsf{list}(z) * \mathsf{emp}$$

Bi-Abductive spec synthesis

Pre: list(x) * list(y)

void foo(list_item *x,list_item *y)

Post: list(x)

```
1  void p(list_item *y) {
2    list_item *x, *z;
3    x=malloc(sizeof(list_item)); x->tail =  0;
4    z=malloc(sizeof(list_item)); z->tail =  0;
5    foo(x,y);
6    foo(x,z);
7  }
```

$\mathsf{list}(y)$

emp

$x \mapsto 0$

$x \mapsto 0 * z \mapsto 0$

$\mathsf{list}(x) * z \mapsto 0$

$\mathsf{list}(x)$

Bi-abductive prover

$$\mathsf{list}(x) * z \mapsto 0 * \mathsf{emp} \vdash \mathsf{list}(x) * \mathsf{list}(z) * \mathsf{emp}$$

General Schema Compositional Analysis

For functions in the program we compute tables of specs

$$\{T_{f_1}, \ldots, T_{f_n}\}$$

Tables are sets of entries of type: $(pre, \{post_1, post_2, \ldots\})$

The computation follows the call graph (start from leaves)

Recursive functions are analyzed with an iterative method until a fixed point is reached

Sum up

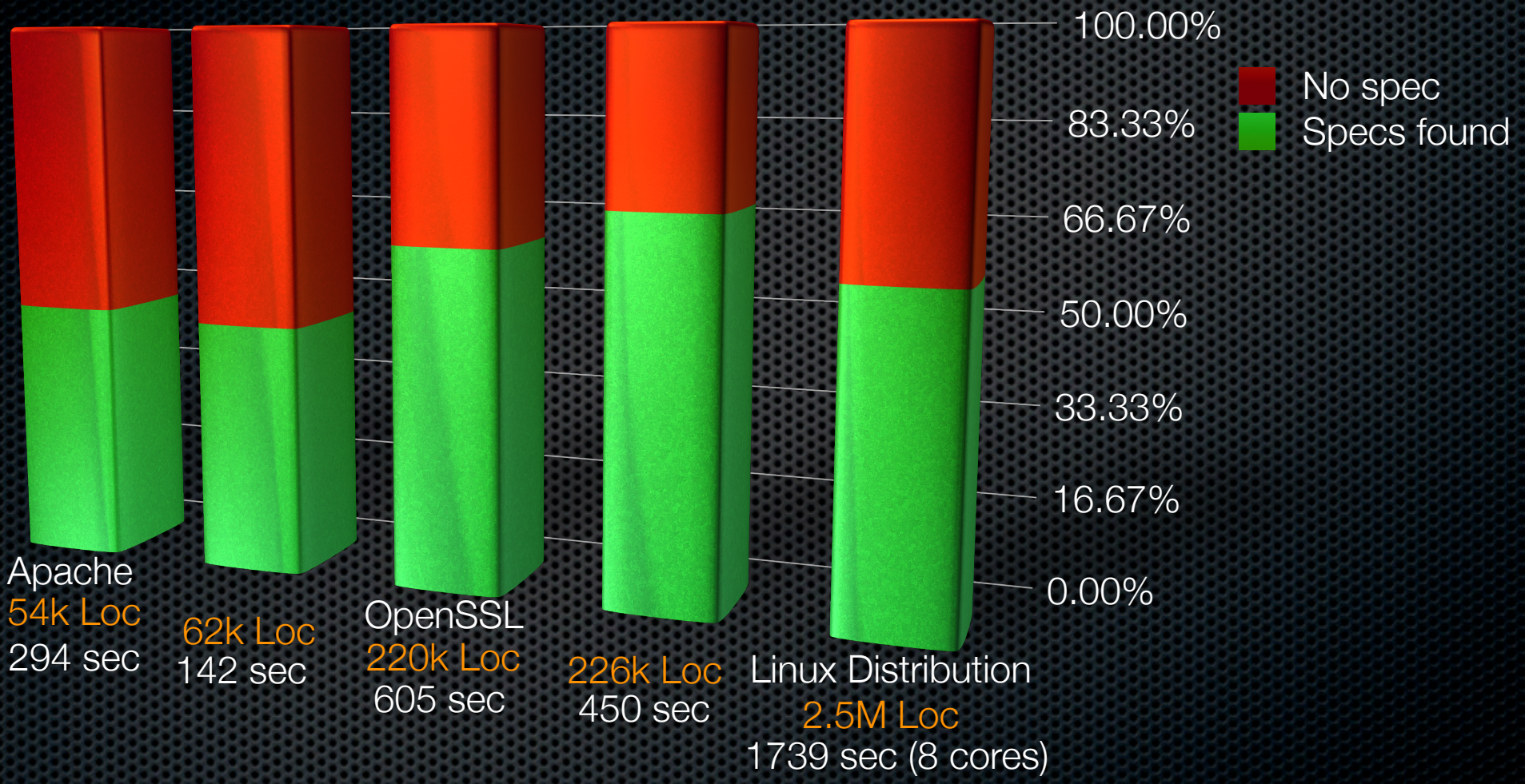
- We can discover specs for procedures without knowing their calling context
- Bi-abduction is used at every step to work out what is missing (anti-frame) and what is left alone (frame)
- This leads to a bottom-up compositional inter-procedural analysis

Experimental Results

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

Running on real code



Test for precision: run on Firewire device driver and small recursive procedures handling nested data structures

The bi-abduction manifesto

- Frame inference $A \vdash B * X$ allows an analyzer to use small specs
- Abduction $A * X \vdash B$ helps to synthesize small specs
- Their combination, bi-abduction

$$A * X \vdash B * X'$$

helps to achieve compositional bottom-up analysis. Furthermore it brings the benefits of local reasoning (as introduced in Separation Logic) to automatic program verification