Lectures on Separation Logic.

Lecture 4: A New Recipe from East London

Peter O'Hearn

Queen Mary, University of London

Marktoberdorf Summer School, 2011

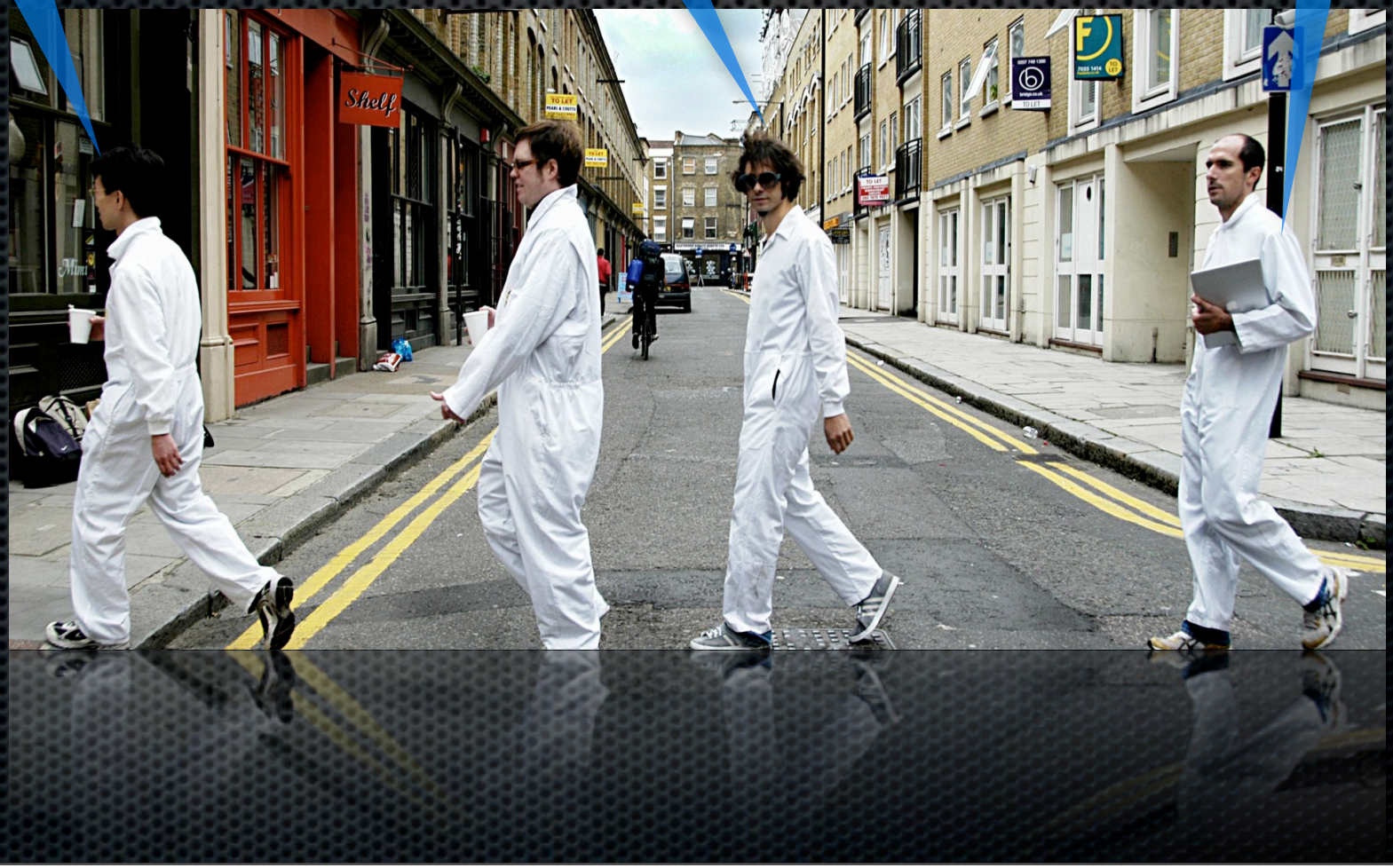
...joint work with

...joint work with

Hongseok Yang

Dino Distefano

Cristiano Calcagno



Source for this lecture

Compositional shape analysis by means of bi-abduction

CDOY, to appear in JACM. Prelim version in POPL'09

Principle of Compositionality

Originally, in Language Semantics

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

Applied to Program Analysis

The analysis result of a composite program is computed from the analysis results of its parts

Problem

Huge abstract domain. Too many abstract states to tabulate all in a procedure summary.

We achieve compositionality,

by aiming for ``small specs''

that describe the footprint

We achieve compositionality,

by aiming for ``small specs''

that describe the footprint

A Small Spec, and a Small Proof

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

- Proof of body of recursive procedure

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

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

A Small Spec, and a Small Proof

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

- Proof of body of recursive procedure

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

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

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

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

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

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

$$A \vdash B * ?$$

How to infer a frame

Convert a failed derivation

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

into a successful one

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

A Small Spec, and a Small Proof

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

- Proof of body of recursive procedure

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

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

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, writing about 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

The Abduction Question

- Given symbolic heaps A and B, find M such that

$$A * M \vdash B$$

The Abduction Question

- Given symbolic heaps A and B, find M such that

$$A * M \vdash B$$

- We would like M to be consistent, and 'minimal' (there is an order)
  - Spatially smallness

$$\mathsf{ls}(y, 0) \preceq \mathsf{ls}(y, 0) * z \mapsto 0$$

  - logical strength

$$\exists z.\mathsf{ls}(y, z) \preceq \mathsf{ls}(y, 0)$$

The Abduction Question

- $$A * ? \vdash B$$

The Abduction Question

- $$x \mapsto \texttt{nil} * ? \;\vdash\; \mathit{list}(x) * \mathit{list}(y)$$

The Abduction Question

- $$x \mapsto \texttt{nil} * \mathit{list}(y) \;\vdash\; \mathit{list}(x) * \mathit{list}(y)$$

The Abduction Question

- $$x \mapsto - * ?? \ \vdash \ y \mapsto - * \texttt{true}$$

The Abduction Question

- $$x \mapsto - * \left(x = y \wedge \mathtt{emp}\right) \;\vdash\; y \mapsto - * \mathtt{true}$$

The Abduction Question

- $$x \mapsto - * y \mapsto - \;\vdash\; y \mapsto - * \mathtt{true}$$

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: $[\text{list}(x) * \text{list}(y)]\, foo(x, y)\, [\text{list}(x)]$

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); }
```

emp

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

emp

$x \mapsto 0$

Abductive Inference:

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

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); }
```

emp

$x \mapsto 0$

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

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

emp

$x \mapsto 0$

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

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

$\mathtt{emp}$ $\quad$ $\mathsf{list}(y)$

$x \mapsto 0$

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

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

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); }
```

$\mathtt{emp}$ $\quad$ $\mathsf{list}(y)$

$x \mapsto 0$

$\mathsf{list}(x)$

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

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

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;	$x \mapsto 0$	
5 foo(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: $[\text{list}(x) * \text{list}(y)]\, foo(x, y)\, [\text{list}(x)]$

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;	$x \mapsto 0$	
5 foo(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: $[\text{list}(x) * \text{list}(y)]\, foo(x, y)\, [\text{list}(x)]$

Bi-Abduction

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

- 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.

Logical Aspects for Generating Pre's

- If path π doesn't modify vars in M...

$$\frac{\{pre\}\pi\{post\} \quad post * M \vdash A}{\{pre * M\}\pi\{A\}}$$

  Derived rule: consequence of Frame Rule and Postcondition Weakening.

- Says not to bother to re-analyze path π; continue on.

Logical Aspects for Generating Pre's

- If path π doesn't modify vars in M...

$$\frac{\{pre\}\pi\{post\} \quad post * M \vdash A}{\{pre * M\}\pi\{A\}}$$

  Derived rule: consequence of Frame Rule and Postcondition Weakening.
- Says not to bother to re-analyze path π; continue on.
- Works only for individual paths, so that abductively induced preconditions are *unsound in general*.

Abduction Soundness Example

$$z \mapsto - \quad * \quad ?? \quad \vdash \quad x \mapsto - * \mathit{true}$$

```
1 void nondet-example(struct node *x,*z) { (z ↦ −) ∗ (x = z ∧ emp)
2       if nondet()
3           z↦ tail = 0;    free(x);
4       else
5           free(x);   free(z)
6       return(); }
```

Abduction Soundness Example

$$z \mapsto - \quad * \quad ?? \quad \vdash \quad x \mapsto - * \mathit{true}$$

```
1 void nondet-example(struct node *x,*z) { (z ↦ −) ∗ (x = z ∧ emp)
2       if nondet()
3           z↦ tail = 0;    free(x);
4       else
5           free(x);   free(z)
6       return(); }
```

No good for path through line 5

Abduction Soundness Example

Good for both paths

$$z \mapsto - \quad * \quad ?? \quad \vdash \quad x \mapsto - * \mathit{true}$$

```
1 void nondet-example(struct node *x,*z) { (z ↦ −) ∗ (x ↦ −)
2       if nondet()
3           z↦ tail = 0;    free(x);
4       else
5           free(x);   free(z)
6       return(); }
```

Inductive

(Periodically Generalizing Precondition,
to stop Infinite Regress)

The Distefano Abstraction[2]

- Merge adjacent pointers/lists into a single list, as long as shared point is not reachable from more than one program var.
- Sample Abstraction (rewrite) Rule:

$$A * \rho(x, Y) * \rho'(Y, 0) \qquad \longrightarrow \qquad A * \mathtt{ls}(x, 0)$$

where logical var Y not free in A
and ρ, ρ' range over $\mathtt{ls}, \mapsto$

[2] Distefano's 2003 thesis, and ported to SL in TACAS'06

The Distefano Abstraction[2]

- Merge adjacent pointers/lists into a single list, as long as shared point is not reachable from more than one program var.
- Sample Abstraction (rewrite) Rule:

$$A * \rho(x, Y) * \rho'(Y, 0) \qquad \longrightarrow \qquad A * \mathtt{ls}(x, 0)$$

where logical var Y not free in A
and ρ, ρ' range over $\mathtt{ls}, \mapsto$

- Example:

$$x \mapsto X_1 * y \mapsto X_1 * X_1 \mapsto X_2 * X_2 \mapsto 0$$

is abstracted to

$$x \mapsto X_1 * y \mapsto X_1 * \mathrm{ls}(X_1, 0).$$

[2] Distefano's 2003 thesis, and ported to SL in TACAS'06

Distefano's Abstraction at work: Deductively

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

Calculated Loop Invariant

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

Distefano's Abstraction at work: Deductively

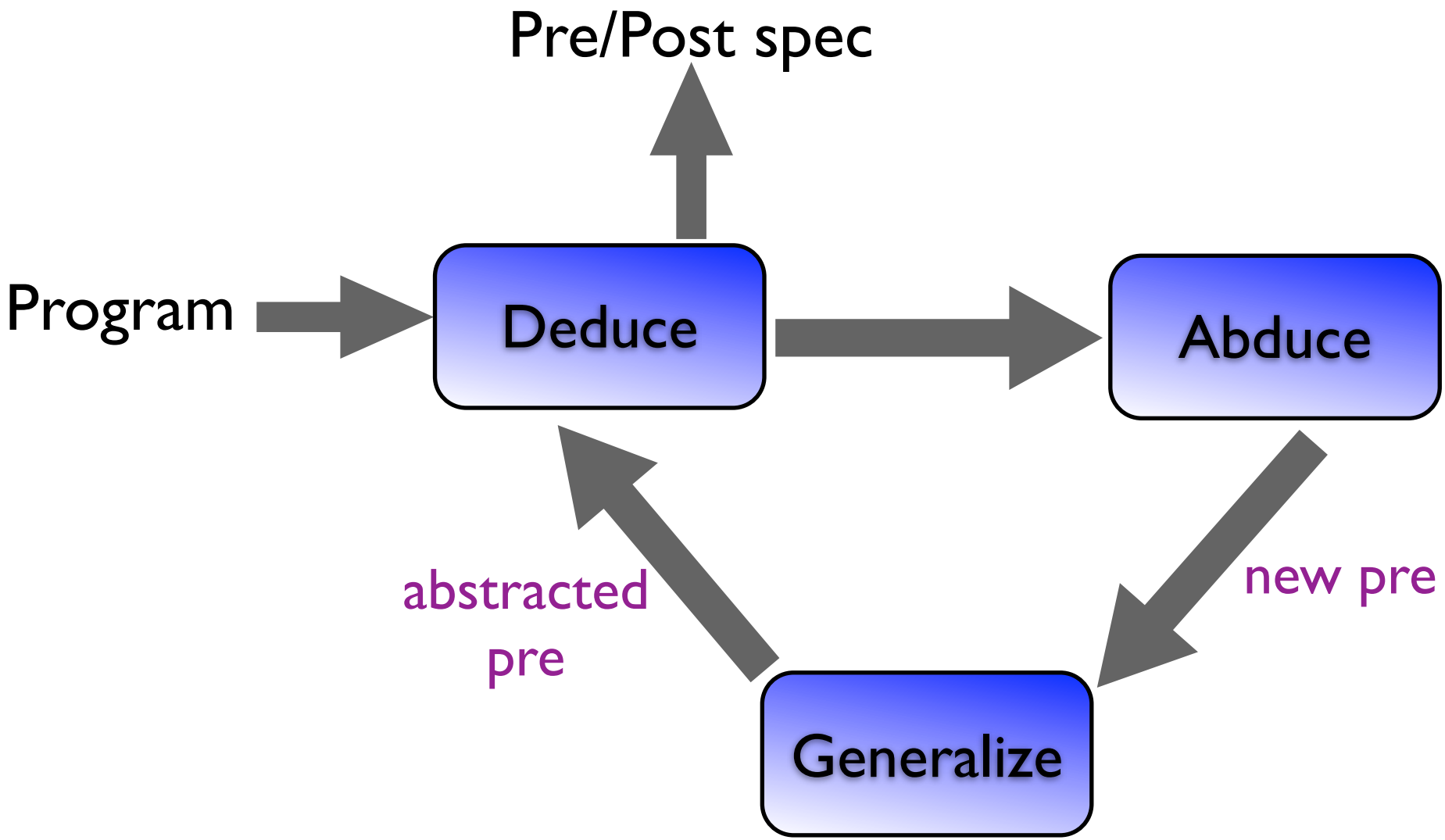
```
{emp}
x=nil;
while (_ ) {          ls(x,nil)
        new(y);
        y ->tl = x;
        x=y;
}
```

Calculated Loop Invariant

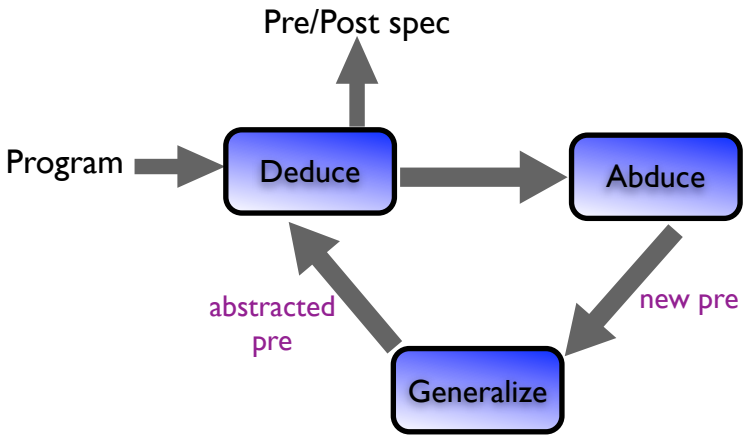
$$
\begin{array}{ll}
 & x = \texttt{nil} \wedge \texttt{emp} \\
\vee & x \mapsto \texttt{nil} \\
\vee & \texttt{ls}(x, \texttt{nil})
\end{array}
$$

Fixed-point reached!

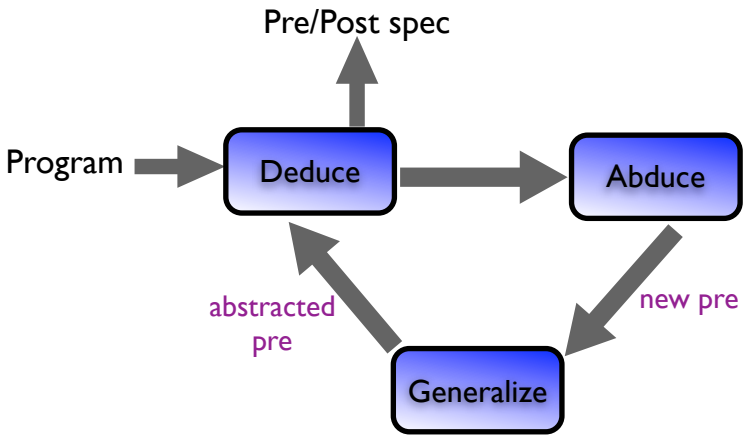
And Inductively (by pictures)

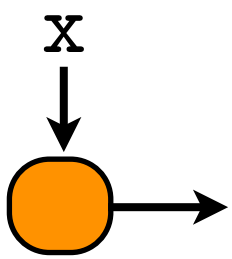

Pre/Post spec
Program
Deduce
Abduce
new pre
abstracted
pre
Generalize

```
while (x!= nil) do {
    t:=x;
    x:=x->tl
    free(t);
}
```



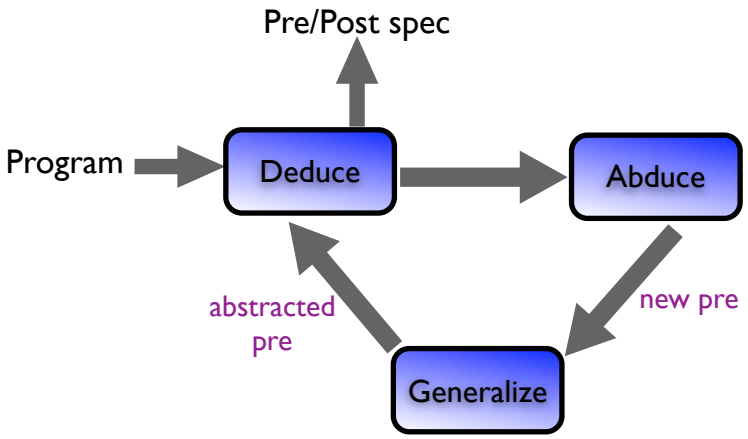
```
x
↓

while (x!= nil) do {
    t:=x;
    x:=x->tl
    free(t);
}
```

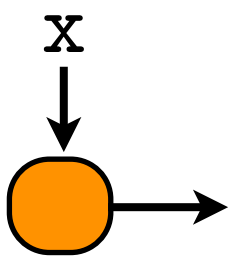




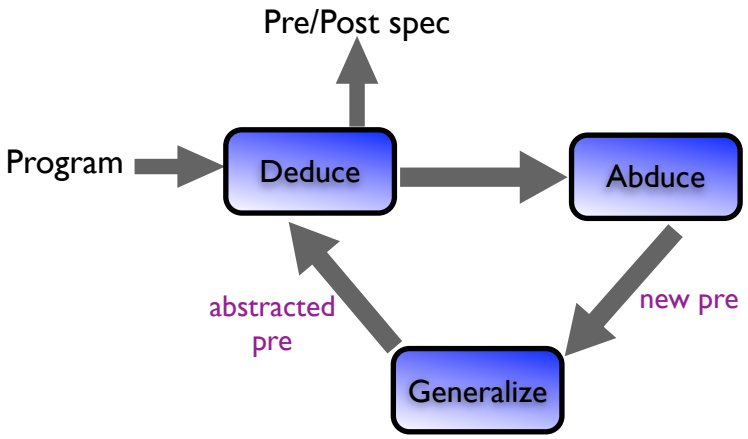

```
while (x!= nil) do {
    t:=x;
    x:=x->tl
    free(t);
}
```

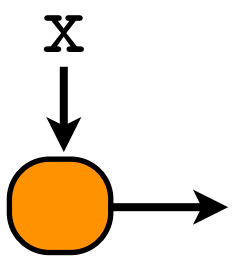
If Only...





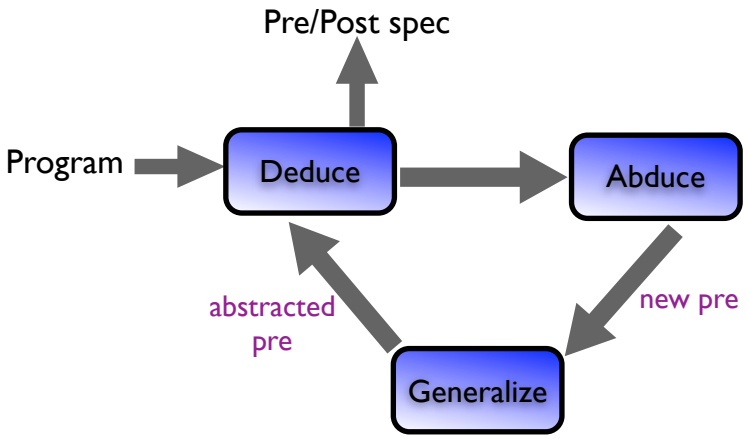

```
while (x!= nil) do {
    t:=x;
    x:=x->tl
    free(t);
}
```

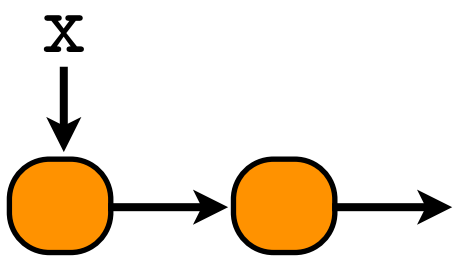





```
while (x!= nil) do {
    t:=x;
    x:=x->tl
    free(t);
}
```

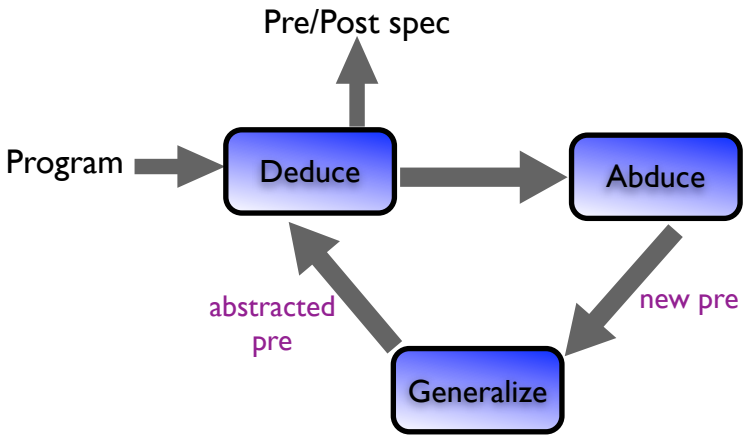
If Only...

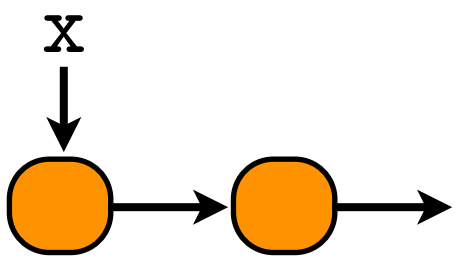




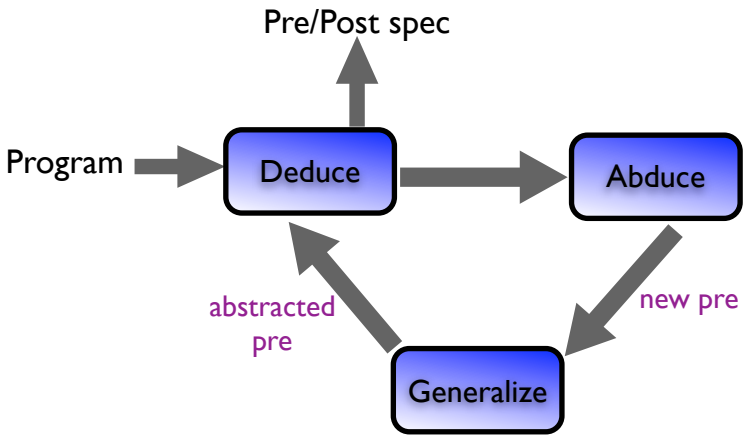

```
while (x!= nil) do {
    t:=x;
    x:=x->tl
    free(t);
}
```

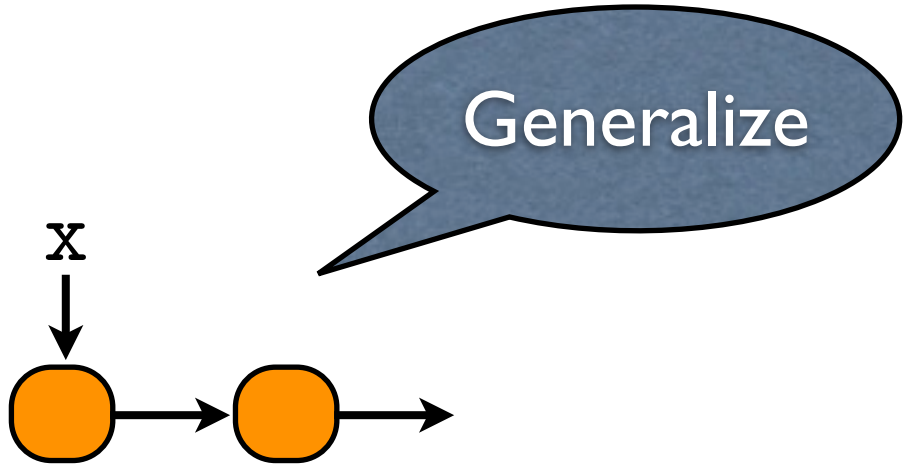
If Only...



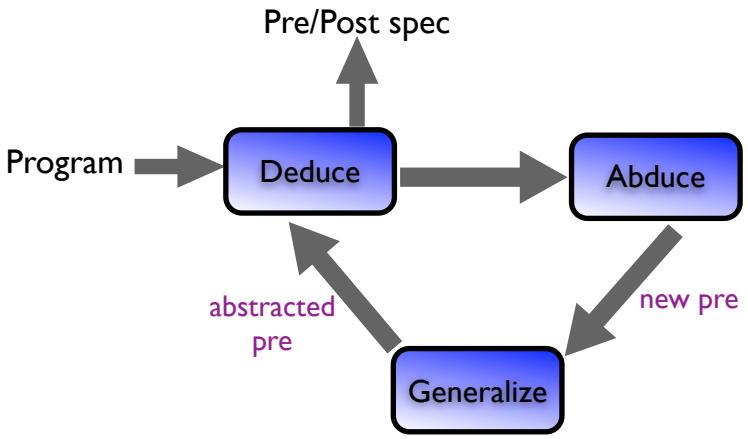


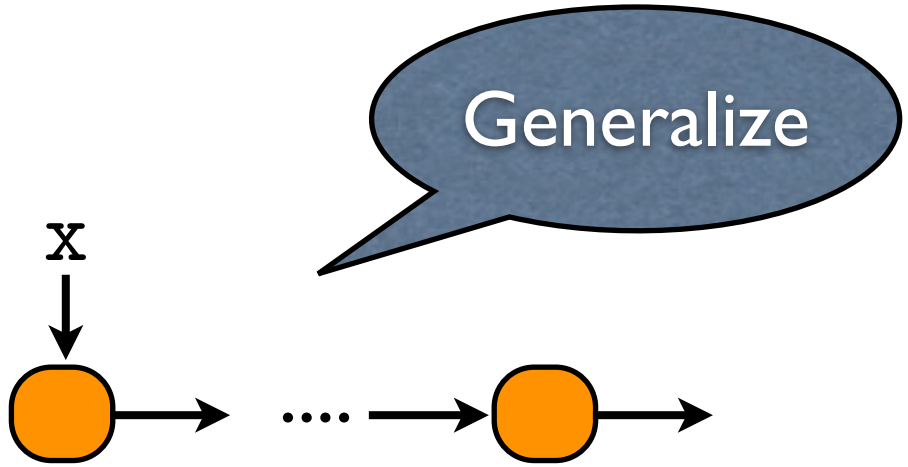

```
while (x!= nil) do {
    t:=x;
    x:=x->tl
    free(t);
}
```



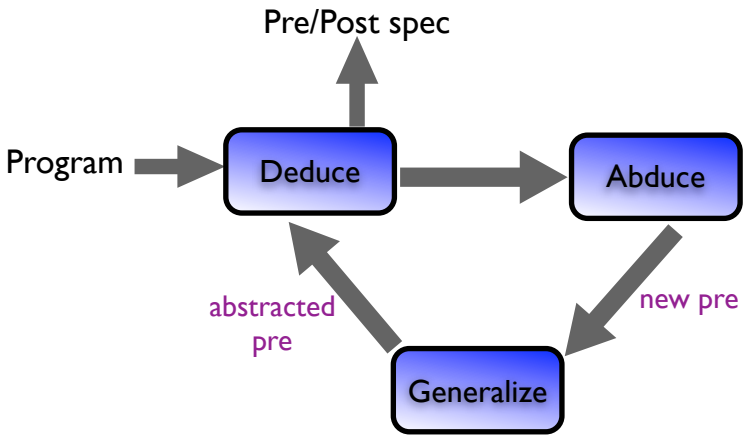


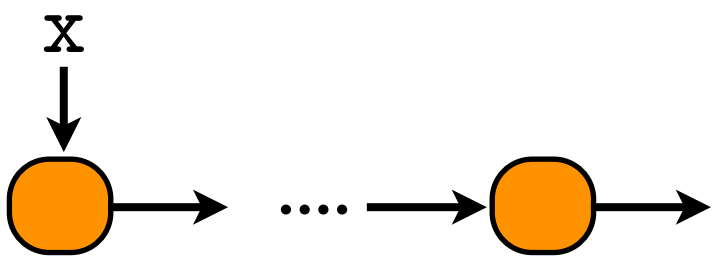

```
while (x!= nil) do {
    t:=x;
    x:=x->tl
    free(t);
}
```



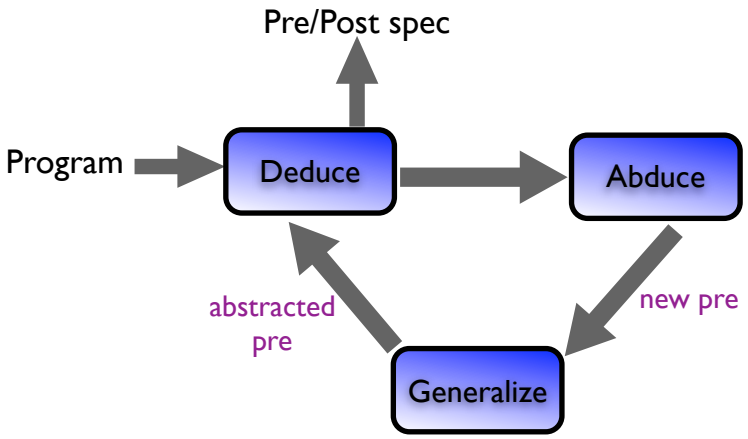


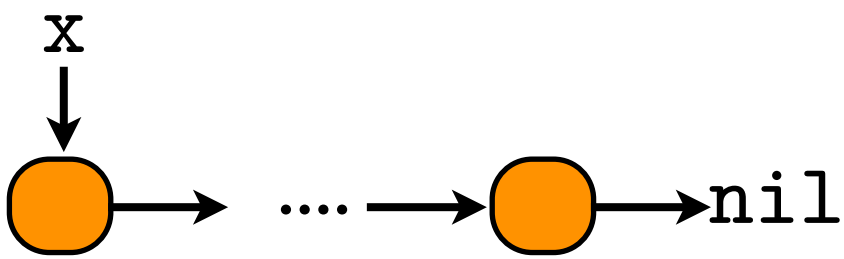

```
while (x!= nil) do {
    t:=x;
    x:=x->tl
    free(t);
}
```



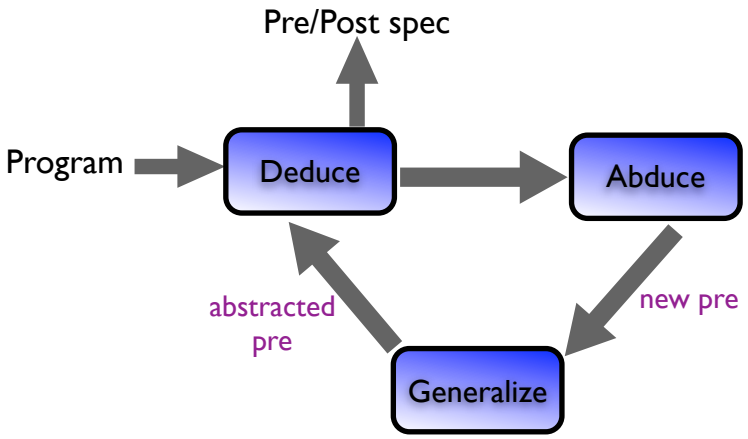


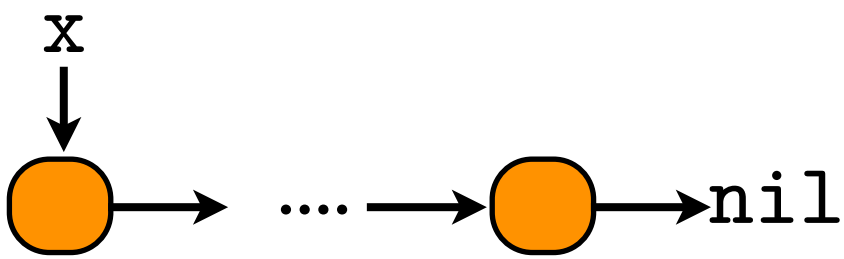

```
while (x!= nil) do {
    t:=x;
    x:=x->tl
    free(t);
}
```



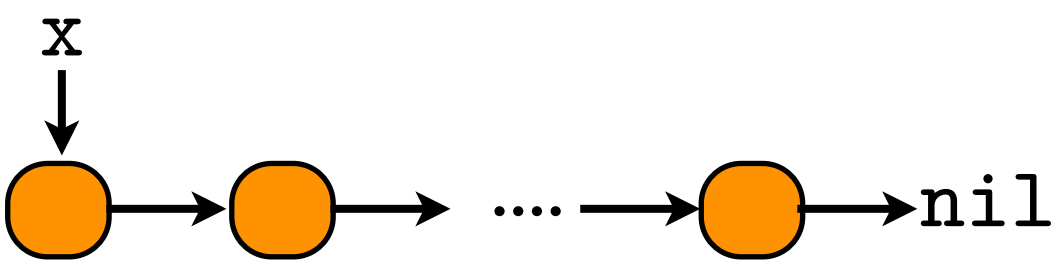


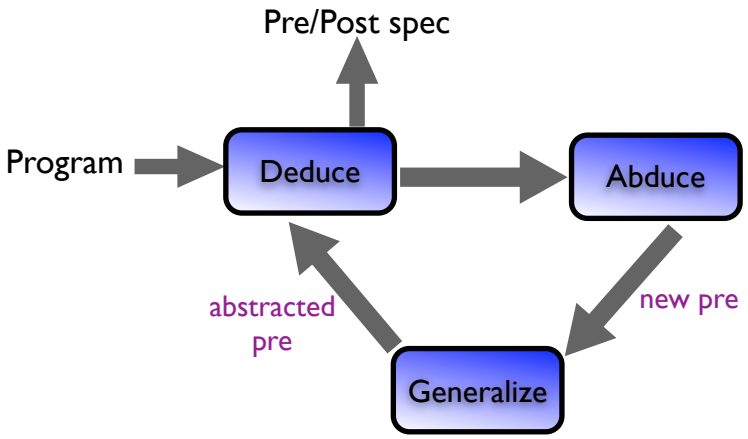

```
while (x!= nil) do {
    t:=x;
    x:=x->tl
    free(t);
}
```

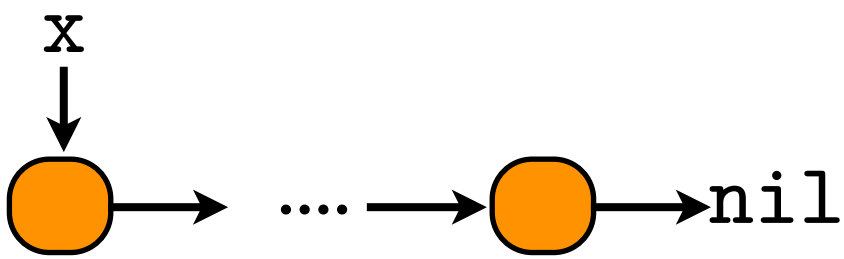




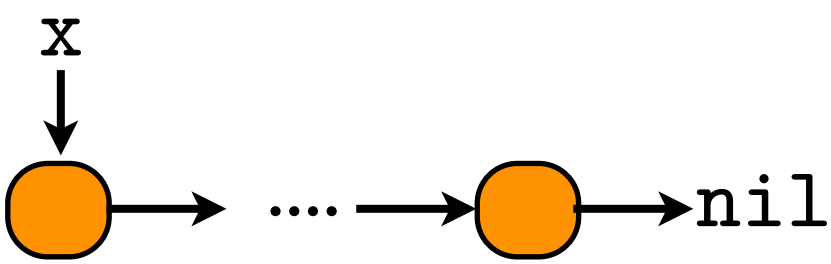

```
while (x!= nil) do {
    t:=x;
    x:=x->tl
    free(t);
}
```

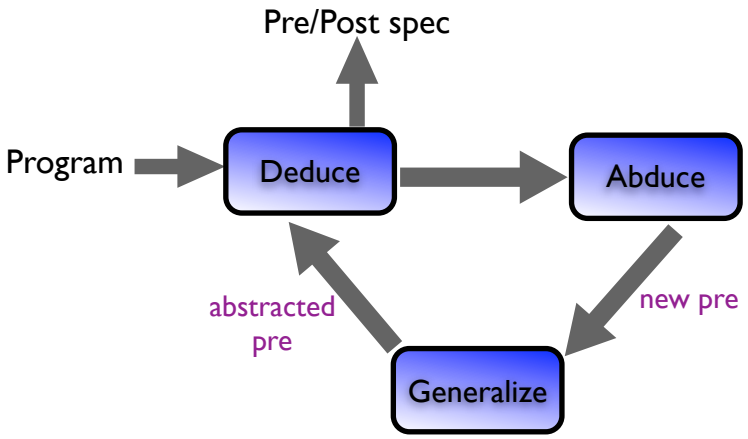


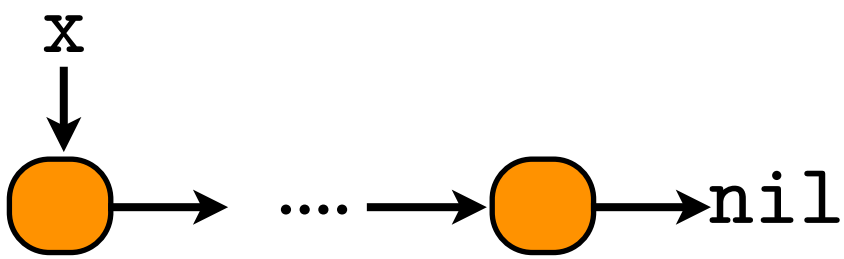




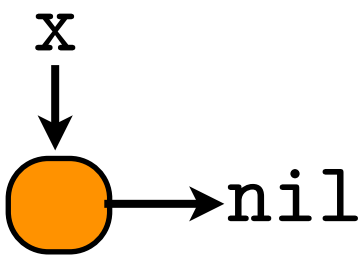

```
while (x!= nil) do {
    t:=x;
    x:=x->tl
    free(t);
}
```

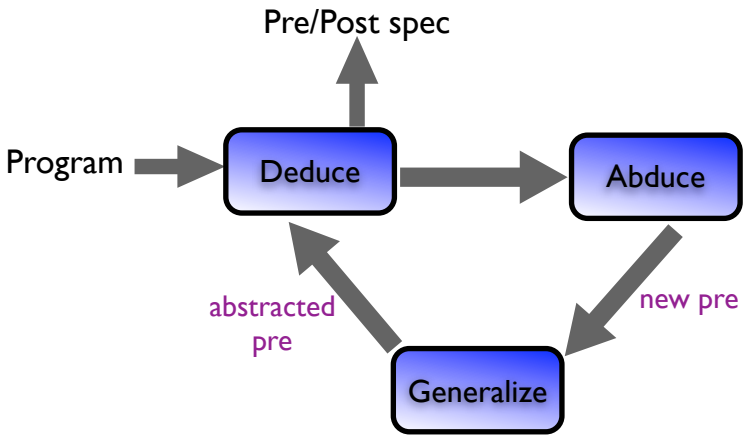


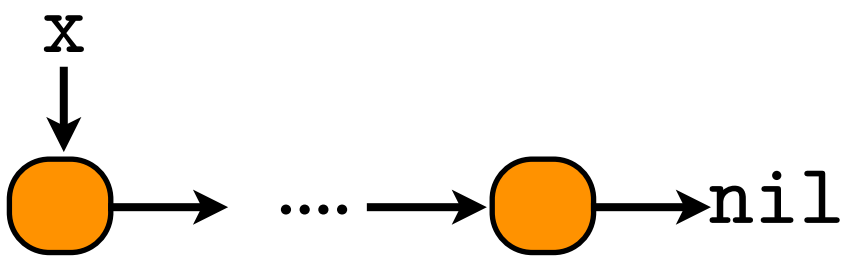




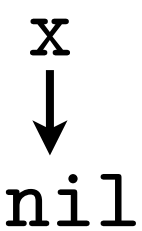

```
while (x!= nil) do {
    t:=x;
    x:=x->tl
    free(t);
}
```

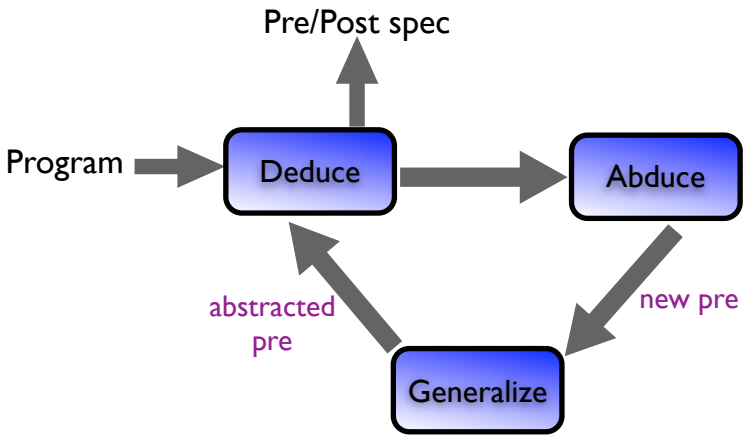







```
while (x!= nil) do {
    t:=x;
    x:=x->tl
    free(t);
}
```





Soundness issues for Abstraction

- Sound for deductive (to calculate loop invariants) because of:

$$\frac{\{A\}C\{B\} \quad B \vdash abs(B)}{\{A\}C\{abs(B)\}}$$

- Unsound for precondition generalization, because

$$\frac{A \vdash abs(A) \quad \{A\}C\{B\}}{\{abs(A)\}C\{B\}}$$

- So, our analysis has a second phase, where we check our work and throw away bad preconditions.

Soundness issues for Abstraction

- Sound for deductive (to calculate loop invariants) because of:

$$\frac{\{A\}C\{B\} \quad B \vdash abs(B)}{\{A\}C\{abs(B)\}}$$

- Unsound for precondition generalization, because

$$\frac{A \vdash abs(A) \quad \{A\}C\{B\}}{\{abs(A)\}C\{B\}}$$

- So, our analysis has a second phase, where we check our work and throw away bad preconditions.

Summary of reasoning method

- Input: A procedure (a bare piece of code)
- Output: Pre/post spec (or nothing)

1. Start by trying to deductively prove a program, from a beginning precondtion.
2. When proof fails, use abduction to infer what is needed. Add to precondition, and carry on.
3. Periodically generalize precondition.
4. Stop when you get to postconditon.
5. Then check that you have actually got a proof.

Note the different roles of abduction and inductive generalization

Conclusion

- This work has been about a passage $A \mapsto C[A]$, taking an abstract domain (or logic) A and making a compositional version $C[A]$.
- Using Abduction-Induction-Deduction gives a boost where we can approach large code bases, more quickly.
- Many proven procedures, but many *unproven* procedures. What next?

Conclusion

- This work has been about a passage $A \mapsto C[A]$, taking an abstract domain (or logic) A and making a compositional version $C[A]$.
- Using Abduction-Induction-Deduction gives a boost where we can approach large code bases, more quickly.
- Many proven procedures, but many *unproven* procedures. What next?
- Perhaps, bring human back into loop
  1. analysis-centric: alter A to A', get $C[A']$
  2. proof-centric: use abduction, etc, in concert with interactive provers

Other tools

In these lectures I have talked about the Smallfoot, Space Invader and Abductor tools developed in London (Queen Mary and Imperial). While these tools advanced interesting ideas, their active development has ceased and there are more recent tools which are probably easier for newcomers to use/try out.

Some other tools include (list not exhaustive)

- **Program Analyses.** SLAyer (Msoft), Predator (Brno), Xisa (Colorado, Paris), jStar (London, Cambridge) ...
- **Automatic Verifiers.** Verifast (Leuwen), jStar, Hip/Sleek (Singapore, Teeside)
- **Interactive.** Bedrock (MIT), Flint (Yale), Verismall (Princeton), Holfoot (Cambridge) ...

The tools in the first two lines at least have been publically released.