Program Verification Using Separation Logic

Dino Distefano
Queen Mary University of London

Lecture 3

Today's plan

- Automatic verification
  - Symbolic execution
  - Frame inference

Automatic Verification: Context

- Around 2000: impressive practical advances in automatic verification. Eg:
  - SLAM: Protocol properties in device drivers, eg. "any call to ReleaseSpinLock is preceded by a call to AquireSpinLock"
  - ASTREE: no run-time errors in Airbus code

but....

- ASTREE assumes: no dynamic pointer allocation
- SLAM: assumes memory safety
- Many important programs make serious use of the heap: Linux, Apache, TCP/IP....
- Could we every crash-proof Apache? OpenSSH?...

but....

- ASTREE assumes: no dynamic pointer allocation
- SLAM: assumes memory safety
- Many important programs make serious use of the heap: Linux, Apache, TCP/IP....
- Could we every crash-proof Apache? OpenSSH?...

Can we do automatic heap verification?

Assertions

Monday we saw the assertion language of separation logic

$$
\begin{array}{llll}
E, F & ::= & x \mid n \mid E{+}F \mid -E \mid \ldots & \text{Heap-independent Exprs} \\
P, Q & ::= & E = F \mid E \geq F \mid E \mapsto F & \text{Atomic Predicates} \\
 & \mid & \mathsf{emp} \mid P * Q & \text{Separating Connectives} \\
 & \mid & \mathsf{true} \mid P \wedge Q \mid \neg P \mid \forall x.\, P & \text{Classical Logic}
\end{array}
$$

It's very expressive, but hard to be dealt with in an automatic tool.

For automatic verification we want a subset:

- simple to automate
- expressive enough for interesting properties

Assertions

Slogan: say less do more!

Monday we saw the assertion language of separation logic

$$
\begin{array}{rcll}
E, F & ::= & x \mid n \mid E{+}F \mid -E \mid \ldots & \text{Heap-independent Exprs} \\
P, Q & ::= & E = F \mid E \geq F \mid E \mapsto F & \text{Atomic Predicates} \\
 & \mid & \mathsf{emp} \mid P * Q & \text{Separating Connectives} \\
 & \mid & \mathsf{true} \mid P \wedge Q \mid \neg P \mid \forall x.\, P & \text{Classical Logic}
\end{array}
$$

It's very expressive, but hard to be dealt with in an automatic tool.

For automatic verification we want a subset:

- simple to automate
- expressive enough for interesting properties

Symbolic Heaps

Symbolic Heaps $\Pi \wedge \Sigma$

Expressions $E \quad ::= \quad x \mid x' \mid \mathsf{nil}$

Pure Formulae

$$\Pi \quad ::= \quad \mathsf{true} \mid E{=}E \mid E \neq E \mid \Pi \wedge \Pi$$

Spatial Formulae

$$\Sigma \quad ::= \quad \mathsf{emp} \mid E \mapsto F \mid \mathsf{junk} \mid \mathsf{ls}\,(E, F) \mid \Sigma * \Sigma$$

Note: primed variable are existentially quantified

Symbolic Heaps

Symbolic Heaps $\Pi \wedge \Sigma$

Expressions $E \quad ::= \quad x \mid x' \mid \mathsf{nil}$

Pure Formulae

$$\Pi \quad ::= \quad \mathsf{true} \mid E =$$

the heap contains garbage

Spatial Formulae

$$\Sigma \quad ::= \quad \mathsf{emp} \mid E \mapsto F \mid \mathsf{junk} \mid \mathsf{ls}\,(E, F) \mid \Sigma * \Sigma$$

Note: primed variable are existentially quantified

Symbolic Heaps

Symbolic Heaps $\Pi \wedge \Sigma$

Expressions $E \quad ::= \quad x \mid x' \mid \mathsf{nil}$

Pure Formulae

$$\Pi \quad ::= \quad \mathsf{true} \mid E{=}E \mid E \neq E \mid \Pi \wedge \Pi$$

Spatial Formulae

$$\Sigma \quad ::= \quad \mathsf{emp} \mid E \mapsto F \mid \mathsf{junk} \mid \mathsf{ls}\,(E, F) \mid \Sigma * \Sigma$$

Note: primed variable are existentially quantified

Symbolic Heaps

Symbolic Heaps $\Pi \wedge \Sigma$

Expressions $E \quad ::= \quad x \mid x' \mid \mathsf{nil}$

Pure Form…

list segment from E to F

$\mathsf{ls}(E, F) \;\; \text{iff} \;\; (\mathsf{emp} \wedge E = F) \;\vee (\exists x'.\, E \mapsto x' * \mathsf{ls}(x', F))$

Spatial Form…

$$\Sigma \quad ::= \quad \mathsf{emp} \mid E \mapsto F \mid \mathsf{junk} \mid \mathsf{ls}\,(E, F) \mid \Sigma * \Sigma$$

Note: primed variable are existentially quantified

What can we express?

We can express:

Shape properties: e.g.

```
p:=nil;
while (c !=nil) do {
  t:=p;
  p:=c;
  c:=[c];
  [p]:=t;
}
```

Does a program preserve acyclicity/ciclicity?

but also: Does is core dump?

Does is create garbage?

What can we express?

We can express:

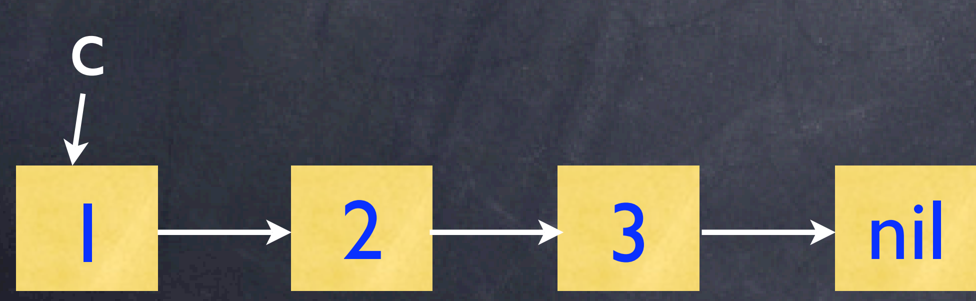
Shape properties: e.g.

Does a program preserve acyclicity/ ciclicity?

but also: Does is core dump?

Does is create garbage?

```
p:=nil;
while (c !=nil) do {
  t:=p;
  p:=c;
  c:=[c];
  [p]:=t;
}
```



What can we express?

We can express:

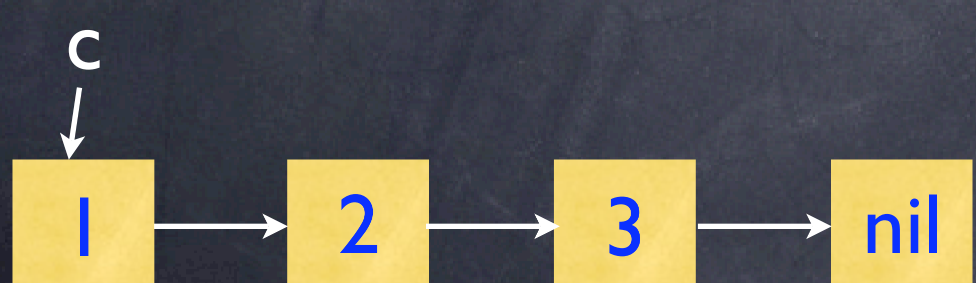
Shape properties: e.g.

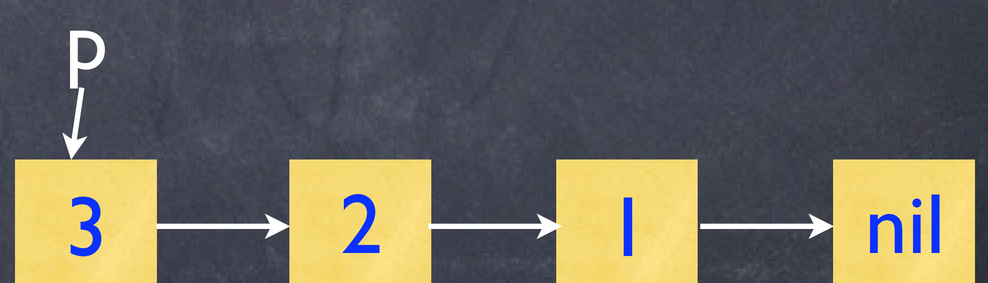
```
p:=nil;
while (c !=nil) do {
  t:=p;
  p:=c;
  c:=[c];
  [p]:=t;
}
```

Does a program preserve acyclicity/ ciclicity?

but also: Does is core dump?

Does is create garbage?

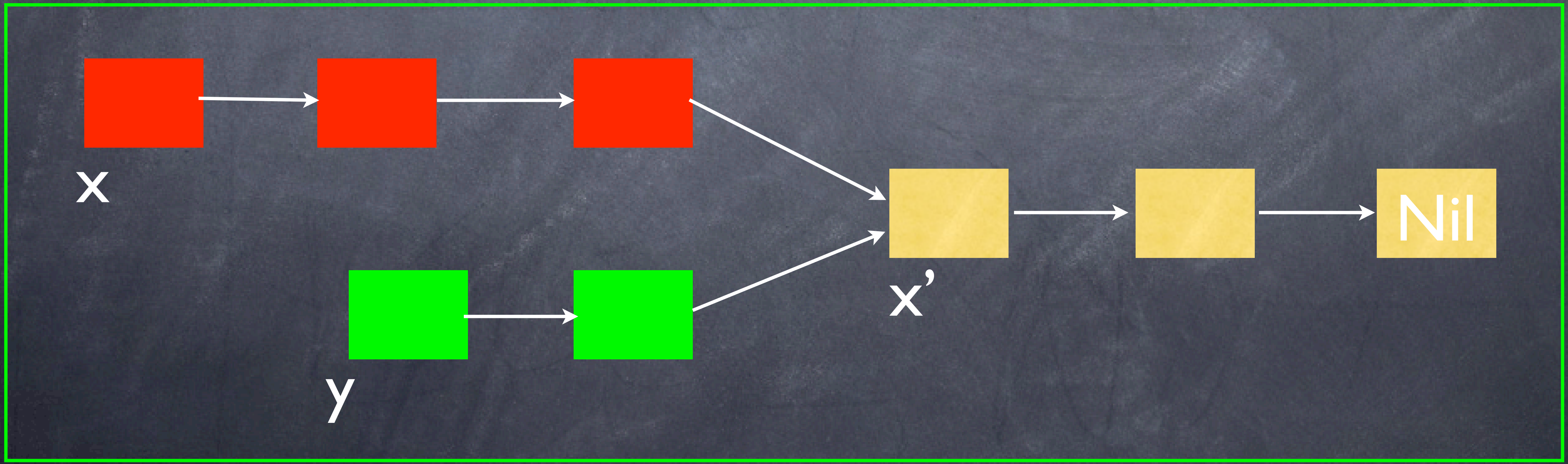




Examples

z is nil whereas x and y point to disjoint lists sharing the tail

$$z = \mathsf{nil} \wedge \mathsf{ls}(x, x') * \mathsf{ls}(y, x') * \mathsf{ls}(\mathsf{x}', \mathsf{nil})$$



Examples

$$x = y \wedge \mathsf{ls}(x, x') * \mathsf{ls}(x', x') * \mathsf{junk}$$

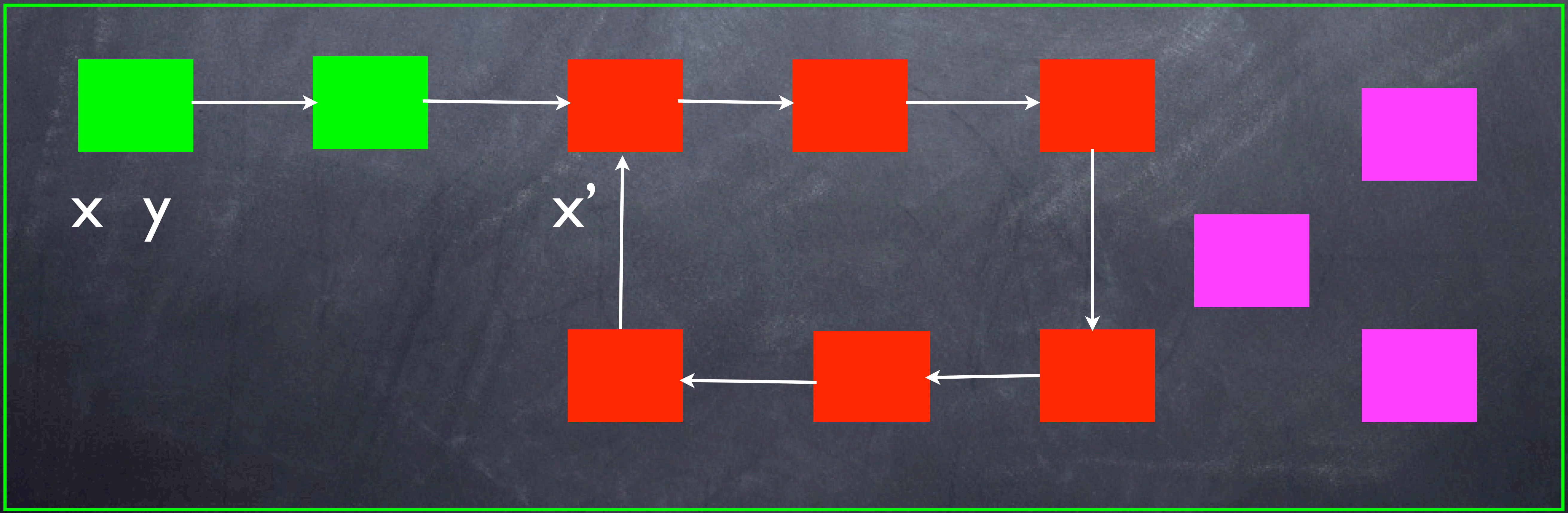
Which kind of heap does it describe?

Examples

$$x = y \wedge \mathsf{ls}(x, x') * \mathsf{ls}(x', x') * \mathsf{junk}$$

Which kind of heap does it describe?

x and y are aliases and they point to a pan-handle list and there is garbage



Symbolic Execution

- Symbolic execution executes the effect of a statement on a symbolic heap
- The result of the modification is another heap or the error state (or T).
- Defined with a relation:

$$\Pi|\Sigma,\ C \implies \Pi'|\Sigma'$$

Rule of Symbolic Execution

$\Pi \mid \Sigma,$	$x := E$	$\Longrightarrow$	$x = E[x'/x] \wedge (\Pi \mid \Sigma)[x'/x]$
$\Pi \mid \Sigma * E \mapsto F,$	$x := [E]$	$\Longrightarrow$	$x = F[x'/x] \wedge (\Pi \mid \Sigma * E \mapsto F)[x'/x]$
$\Pi \mid \Sigma * E \mapsto F,$	$[E] := G$	$\Longrightarrow$	$\Pi \mid \Sigma * E \mapsto G$
$\Pi ; \Sigma,$	$\mathsf{new}(x)$	$\Longrightarrow$	$(\Pi \mid \Sigma)[x'/x] * x \mapsto y'$
$\Pi \mid \Sigma * E \mapsto F,$	$\mathsf{dispose}(E)$	$\Longrightarrow$	$\Pi \mid \Sigma$

$$\frac{\Pi \mid \Sigma \nvdash \mathsf{Allocated}(E)}{\Pi \mid \Sigma,\ A(E) \ \Longrightarrow \top}$$

x′,y′ fresh existentially quantified variables

Rule of Symbolic Execution

$\Pi\vert\Sigma,$	$x := E$	$\Longrightarrow$	$x = E[x'/x] \wedge (\Pi\vert\Sigma)[x'/x]$
$\Pi\vert\Sigma * E \mapsto F,$	$x := [E]$	$\Longrightarrow$	$x = F[x'/x] \wedge (\Pi\vert\Sigma * E \mapsto F)[x'/x]$
$\Pi\vert\Sigma * E \mapsto F,$	$[E] := G$	$\Longrightarrow$	$\Pi\vert\Sigma * E \mapsto G$
$\Pi;\Sigma,$	$\mathsf{new}(x)$	$\Longrightarrow$	$(\Pi\vert\Sigma)[x'/x] * x \mapsto y'$
$\Pi\vert\Sigma * E \mapsto F,$	$\mathsf{dispose}(E)$	$\Longrightarrow$	$\Pi\vert\Sigma$

$$\frac{\Pi|\Sigma \nvdash \mathsf{Allocated}(E)}{\Pi|\Sigma,\ A(E) \Longrightarrow \top}$$

x′,y′ fresh existentially quantified variables

Rule of Symbolic Execution

$\Pi\|\Sigma,$	$x := E$	$\Longrightarrow$	$x = E[x'/x] \wedge (\Pi\|\Sigma)[x'/x]$
$\Pi\|\Sigma * E \mapsto F,$	$x := [E]$	$\Longrightarrow$	$x = F[x'/x] \wedge (\Pi\|\Sigma * E \mapsto F)[x'/x]$
$\Pi\|\Sigma * E \mapsto F,$	$[E] := G$	$\Longrightarrow$	$\Pi\|\Sigma * E \mapsto G$
$\Pi; \Sigma,$	$\mathsf{new}(x)$	$\Longrightarrow$	$(\Pi\|\Sigma)[x'/x] * x \mapsto y'$
$\Pi\|\Sigma * E \mapsto F,$	$\mathsf{dispose}(E)$	$\Longrightarrow$	$\Pi\|\Sigma$

$$\frac{\Pi|\Sigma \nvdash \mathsf{Allocated}(E)}{\Pi|\Sigma,\ A(E) \Longrightarrow \top}$$

x′,y′ fresh existentially quantified variables

Rule of Symbolic Execution

$\Pi\vert\Sigma,$	$x := E$	$\Longrightarrow$	$x = E[x'/x] \wedge (\Pi\vert\Sigma)[x'/x]$
$\Pi\vert\Sigma * E \mapsto F,$	$x := [E]$	$\Longrightarrow$	$x = F[x'/x] \wedge (\Pi\vert\Sigma * E \mapsto F)[x'/x]$
$\Pi\vert\Sigma * E \mapsto F,$	$[E] := G$	$\Longrightarrow$	$\Pi\vert\Sigma * E \mapsto G$
$\Pi;\Sigma,$	$\mathsf{new}(x)$	$\Longrightarrow$	$(\Pi\vert\Sigma)[x'/x] * x \mapsto y'$
$\Pi\vert\Sigma * E \mapsto F,$	$\mathsf{dispose}(E)$	$\Longrightarrow$	$\Pi\vert\Sigma$

$$\frac{\Pi|\Sigma \nvdash \mathsf{Allocated}(E)}{\Pi|\Sigma,\ \ A(E) \ \Longrightarrow \top}$$

x′,y′ fresh existentially quantified variables

Rule of Symbolic Execution

$\Pi \mid \Sigma,$	$x := E$	$\Longrightarrow$	$x = E[x'/x] \wedge (\Pi \mid \Sigma)[x'/x]$
$\Pi \mid \Sigma * E \mapsto F,$	$x := [E]$	$\Longrightarrow$	$x = F[x'/x] \wedge (\Pi \mid \Sigma * E \mapsto F)[x'/x]$
$\Pi \mid \Sigma * E \mapsto F,$	$[E] := G$	$\Longrightarrow$	$\Pi \mid \Sigma * E \mapsto G$
$\Pi ; \Sigma,$	$\mathsf{new}(x)$	$\Longrightarrow$	$(\Pi \mid \Sigma)[x'/x] * x \mapsto y'$
$\Pi \mid \Sigma * E \mapsto F,$	$\mathsf{dispose}(E)$	$\Longrightarrow$	$\Pi \mid \Sigma$

$$\frac{\Pi \mid \Sigma \nvdash \mathsf{Allocated}(E)}{\Pi \mid \Sigma,\ A(E) \ \Longrightarrow \top}$$

x′,y′ fresh existentially quantified variables

Soundness

- Is this symbolic semantics sound?
- Is which sense it is sounds?
- We need to show that the symbolic semantics describe an superset of all possible computation of the program (i.e., it is an over-approximation)

Concrete semantics

$$\frac{\mathcal{C}[\![E]\!]s = n}{s, h,\ x := E \implies (s|x \mapsto n), h}$$

$$\frac{\ell \notin dom(h)}{s, h,\ \mathsf{new}(x) \implies (s|x \mapsto \ell), (h|\ell \mapsto n)}$$

$$\frac{\mathcal{C}[\![E]\!]s = \ell \quad h(\ell) = n}{s, h,\ x := [E] \implies (s|x \mapsto n), h}$$

$$\frac{\mathcal{C}[\![E]\!]s = \ell}{s, h * [\ell \mapsto n],\ \mathsf{dispose}(E) \implies s, h}$$

$$\frac{\mathcal{C}[\![E]\!]s = \ell \quad \mathcal{C}[\![F]\!]s = n \quad \ell \in dom(h)}{s, h,\ [E] := F \implies s, (h|\ell \mapsto n)}$$

$$\frac{\mathcal{C}[\![E]\!]s \notin dom(h)}{s, h,\ A(E) \implies \top}$$

Theorem

The symbolic semantics is a sound over-approximation of the concrete semantics.

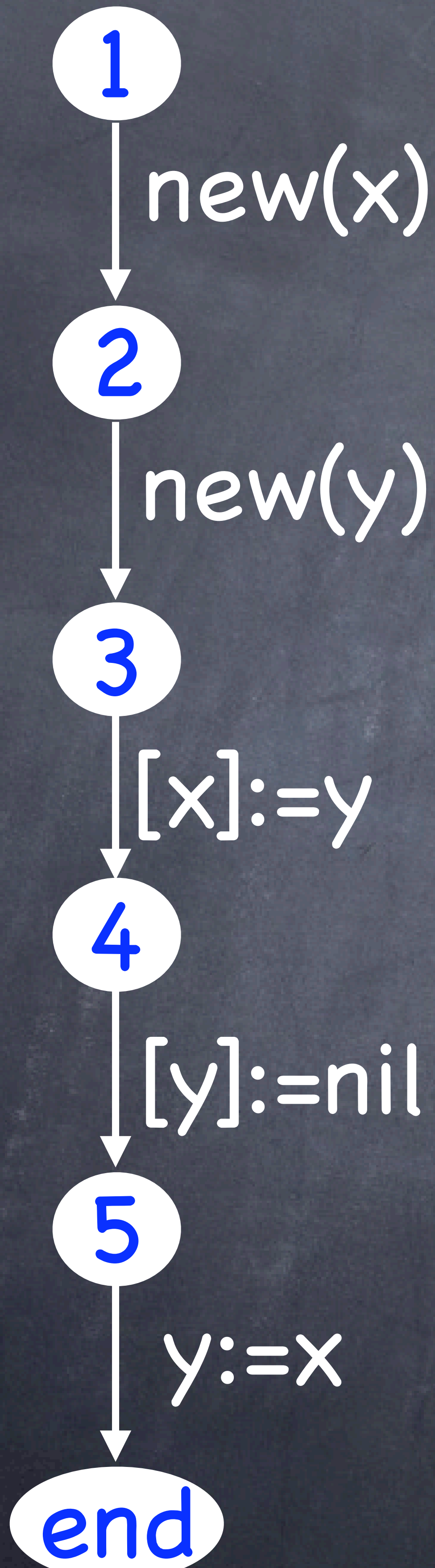
Example 1

$\Pi\|\Sigma,$	$x := E$	$\Longrightarrow$	$x = E[x'/x] \wedge (\Pi\|\Sigma)[x'/x]$
$\Pi\|\Sigma * E \mapsto F,$	$x := [E]$	$\Longrightarrow$	$x = F[x'/x] \wedge (\Pi\|\Sigma * E \mapsto F)[x'/x]$
$\Pi\|\Sigma * E \mapsto F,$	$[E] := G$	$\Longrightarrow$	$\Pi\|\Sigma * E \mapsto G$
$\Pi;\Sigma,$	$\mathsf{new}(x)$	$\Longrightarrow$	$(\Pi\|\Sigma)[x'/x] * x \mapsto y'$
$\Pi\|\Sigma * E \mapsto F,$	$\mathsf{dispose}(E)$	$\Longrightarrow$	$\Pi\|\Sigma$

$$\frac{\Pi|\Sigma \nvdash \mathsf{Allocated}(E)}{\Pi|\Sigma,\ A(E) \Longrightarrow \top}$$

```
new(x);
new(y);
[x]:=y;
[y]:=nil;
y:=x;
```

Example 1

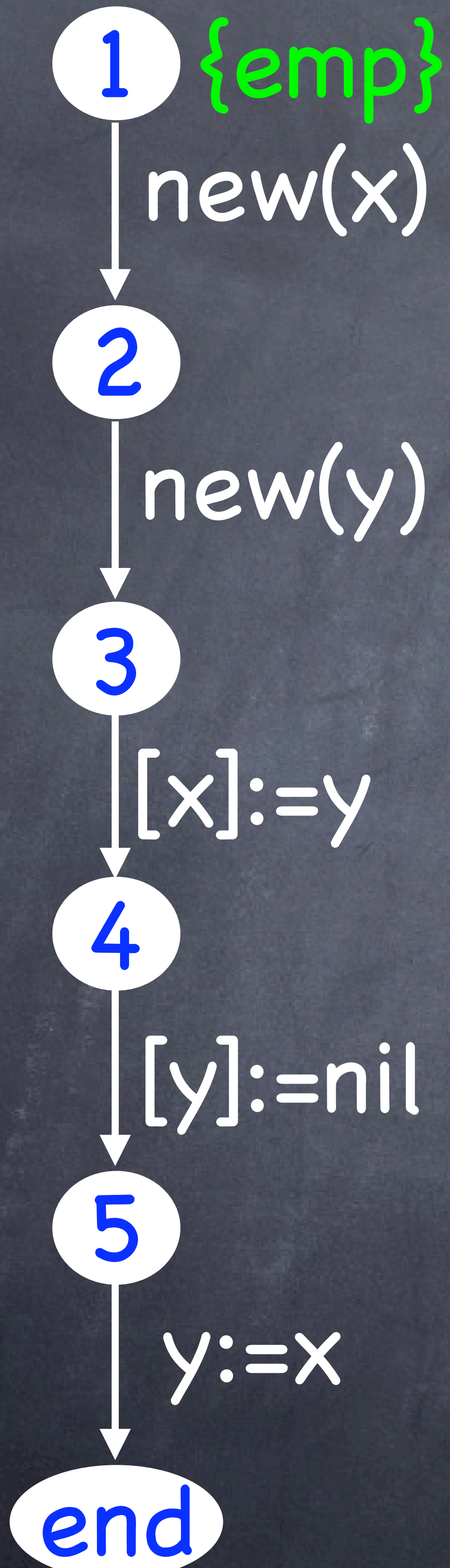


$$
\begin{array}{llll}
\Pi|\Sigma, & x := E & \Longrightarrow & x = E[x'/x] \wedge (\Pi|\Sigma)[x'/x] \\
\Pi|\Sigma * E \mapsto F, & x := [E] & \Longrightarrow & x = F[x'/x] \wedge (\Pi|\Sigma * E \mapsto F)[x'/x] \\
\Pi|\Sigma * E \mapsto F, & [E] := G & \Longrightarrow & \Pi|\Sigma * E \mapsto G \\
\Pi;\Sigma, & \mathsf{new}(x) & \Longrightarrow & (\Pi|\Sigma)[x'/x] * x \mapsto y' \\
\Pi|\Sigma * E \mapsto F, & \mathsf{dispose}(E) & \Longrightarrow & \Pi|\Sigma
\end{array}
$$

$$
\frac{\Pi|\Sigma \nvdash \mathsf{Allocated}(E)}{\Pi|\Sigma,\ A(E) \ \Longrightarrow \top}
$$

```
new(x);
new(y);
[x]:=y;
[y]:=nil;
y:=x;
```

Example 1



$$\Pi|\Sigma, \quad x := E \quad \Longrightarrow \quad x = E[x'/x] \wedge (\Pi|\Sigma)[x'/x]$$

$$\Pi|\Sigma * E \mapsto F, \quad x := [E] \quad \Longrightarrow \quad x = F[x'/x] \wedge (\Pi|\Sigma * E \mapsto F)[x'/x]$$

$$\Pi|\Sigma * E \mapsto F, \quad [E] := G \quad \Longrightarrow \quad \Pi|\Sigma * E \mapsto G$$

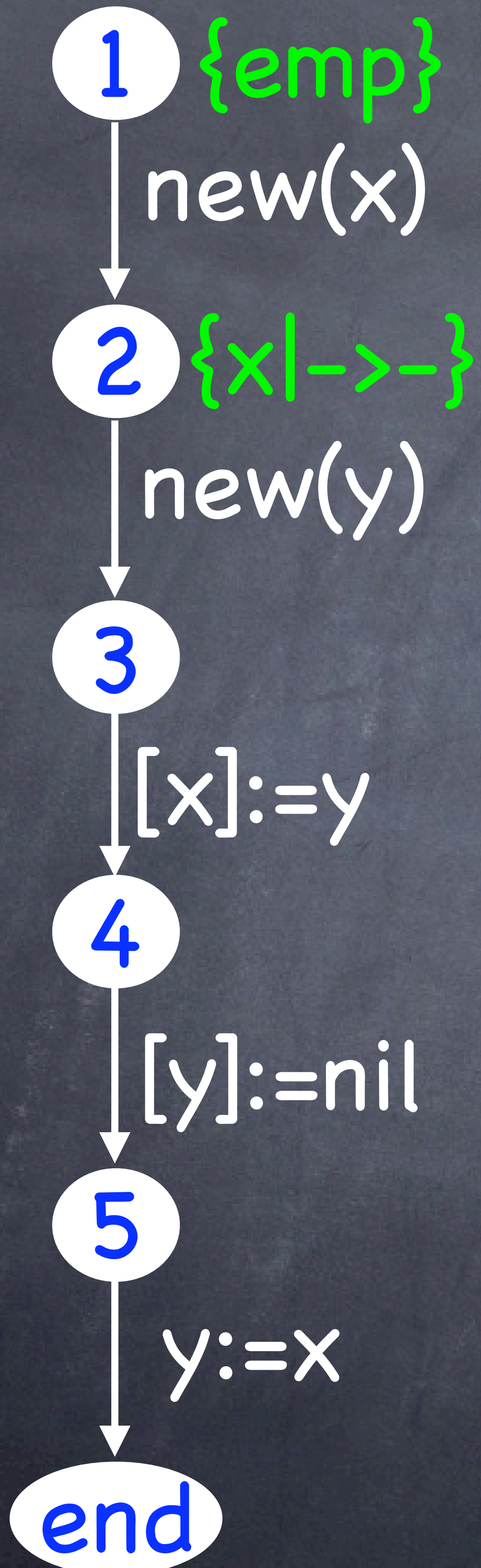
$$\Pi; \Sigma, \quad \mathsf{new}(x) \quad \Longrightarrow \quad (\Pi|\Sigma)[x'/x] * x \mapsto y'$$

$$\Pi|\Sigma * E \mapsto F, \quad \mathsf{dispose}(E) \quad \Longrightarrow \quad \Pi|\Sigma$$

$$\frac{\Pi|\Sigma \nvdash \mathsf{Allocated}(E)}{\Pi|\Sigma,\ A(E) \ \Longrightarrow \top}$$

```
new(x);
new(y);
[x]:=y;
[y]:=nil;
y:=x;
```

Example 1

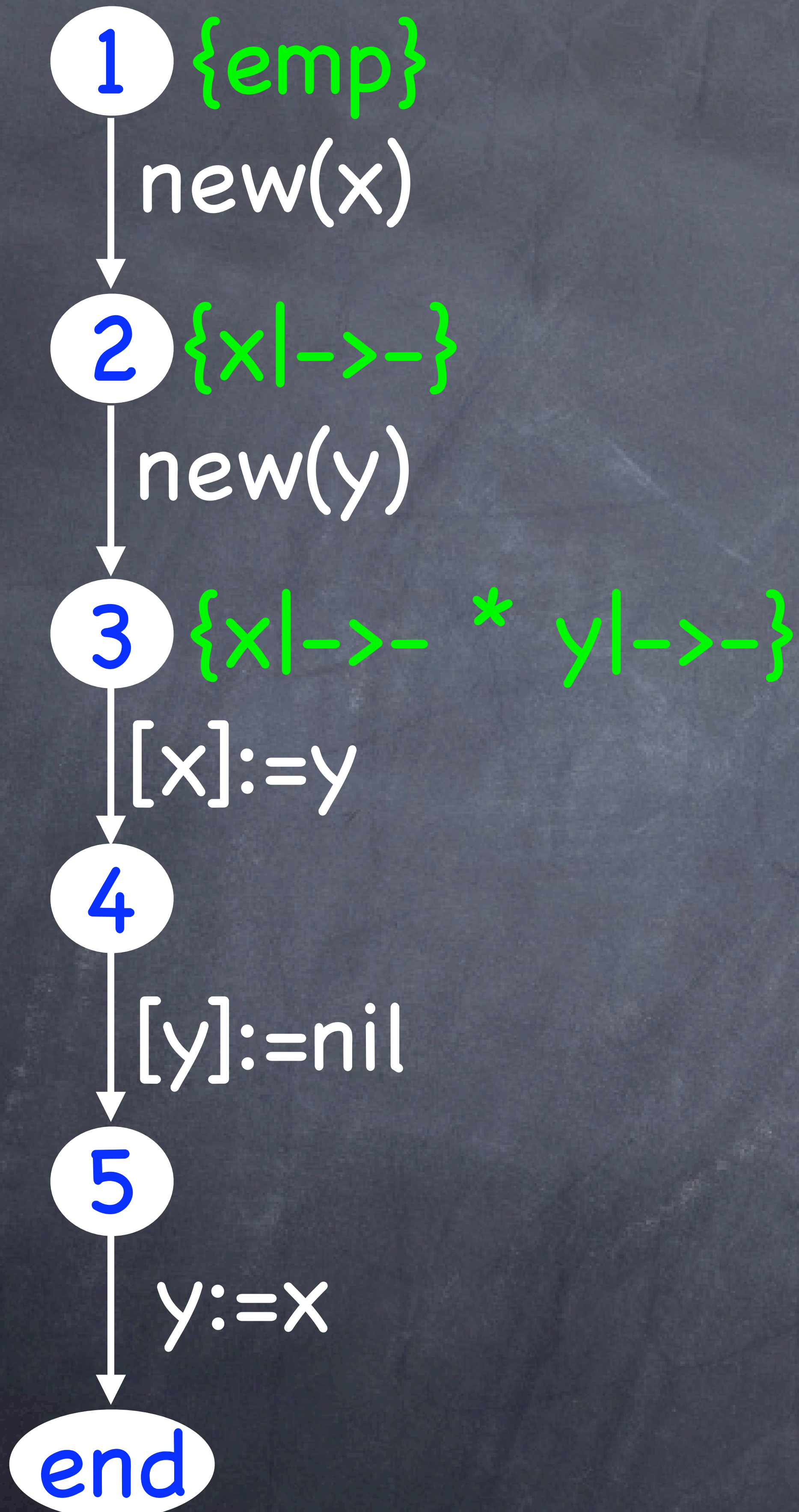


$$\begin{array}{llcl}
\Pi|\Sigma, & x := E & \Longrightarrow & x = E[x'/x] \wedge (\Pi|\Sigma)[x'/x] \\
\Pi|\Sigma * E \mapsto F, & x := [E] & \Longrightarrow & x = F[x'/x] \wedge (\Pi|\Sigma * E \mapsto F)[x'/x] \\
\Pi|\Sigma * E \mapsto F, & [E] := G & \Longrightarrow & \Pi|\Sigma * E \mapsto G \\
\Pi; \Sigma, & \mathsf{new}(x) & \Longrightarrow & (\Pi|\Sigma)[x'/x] * x \mapsto y' \\
\Pi|\Sigma * E \mapsto F, & \mathsf{dispose}(E) & \Longrightarrow & \Pi|\Sigma
\end{array}$$

$$\frac{\Pi|\Sigma \nvdash \mathsf{Allocated}(E)}{\Pi|\Sigma,\ A(E)\ \Longrightarrow \top}$$

```
new(x);
new(y);
[x]:=y;
[y]:=nil;
y:=x;
```

Example 1

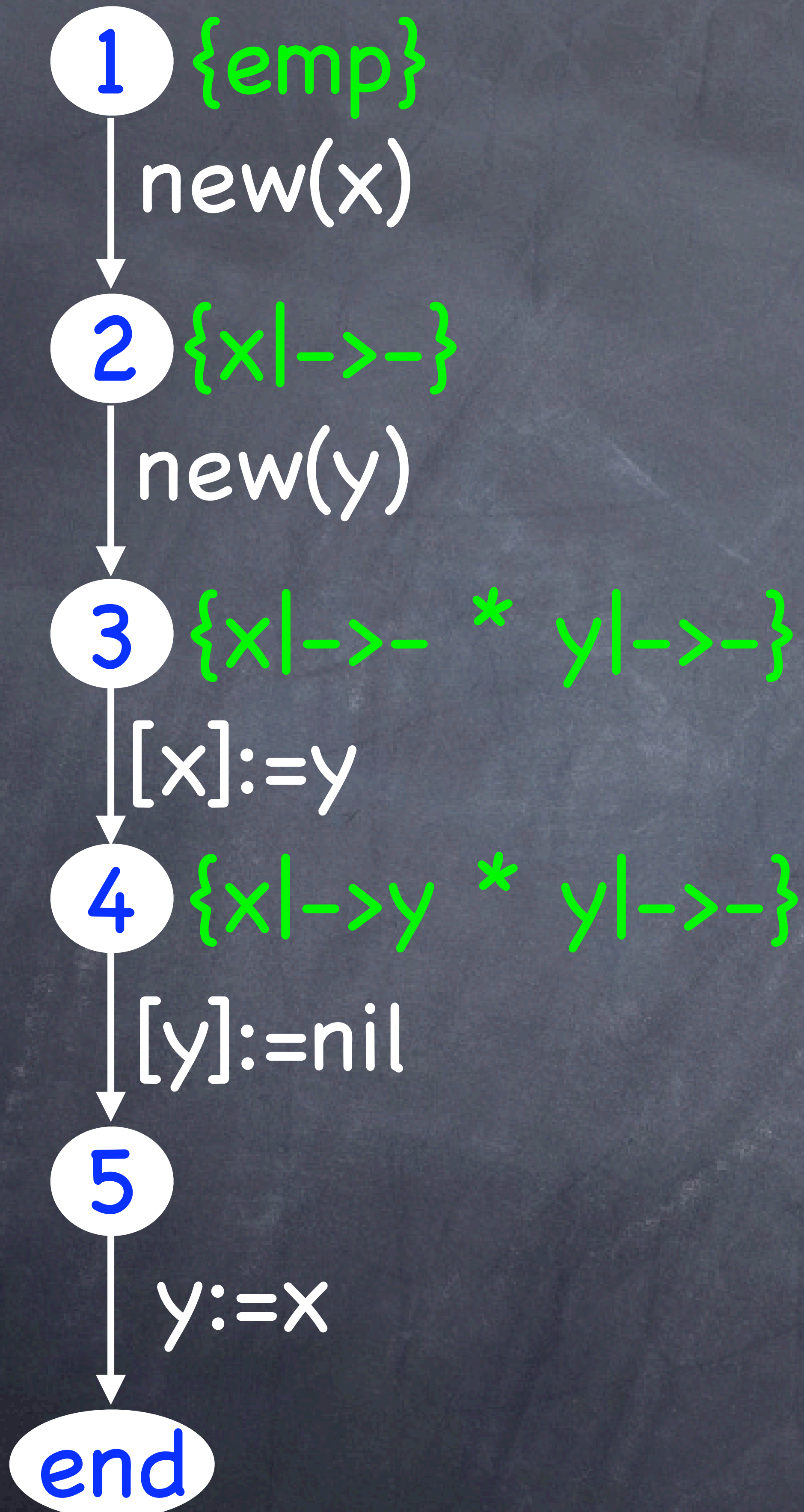


$$
\begin{array}{llll}
\Pi|\Sigma, & x := E & \Longrightarrow & x = E[x'/x] \wedge (\Pi|\Sigma)[x'/x] \\
\Pi|\Sigma * E \mapsto F, & x := [E] & \Longrightarrow & x = F[x'/x] \wedge (\Pi|\Sigma * E \mapsto F)[x'/x] \\
\Pi|\Sigma * E \mapsto F, & [E] := G & \Longrightarrow & \Pi|\Sigma * E \mapsto G \\
\Pi;\Sigma, & \mathsf{new}(x) & \Longrightarrow & (\Pi|\Sigma)[x'/x] * x \mapsto y' \\
\Pi|\Sigma * E \mapsto F, & \mathsf{dispose}(E) & \Longrightarrow & \Pi|\Sigma
\end{array}
$$

$$
\frac{\Pi|\Sigma \nvdash \mathsf{Allocated}(E)}{\Pi|\Sigma,\ A(E) \Longrightarrow \top}
$$

```
new(x);
new(y);
[x]:=y;
[y]:=nil;
y:=x;
```

Example 1

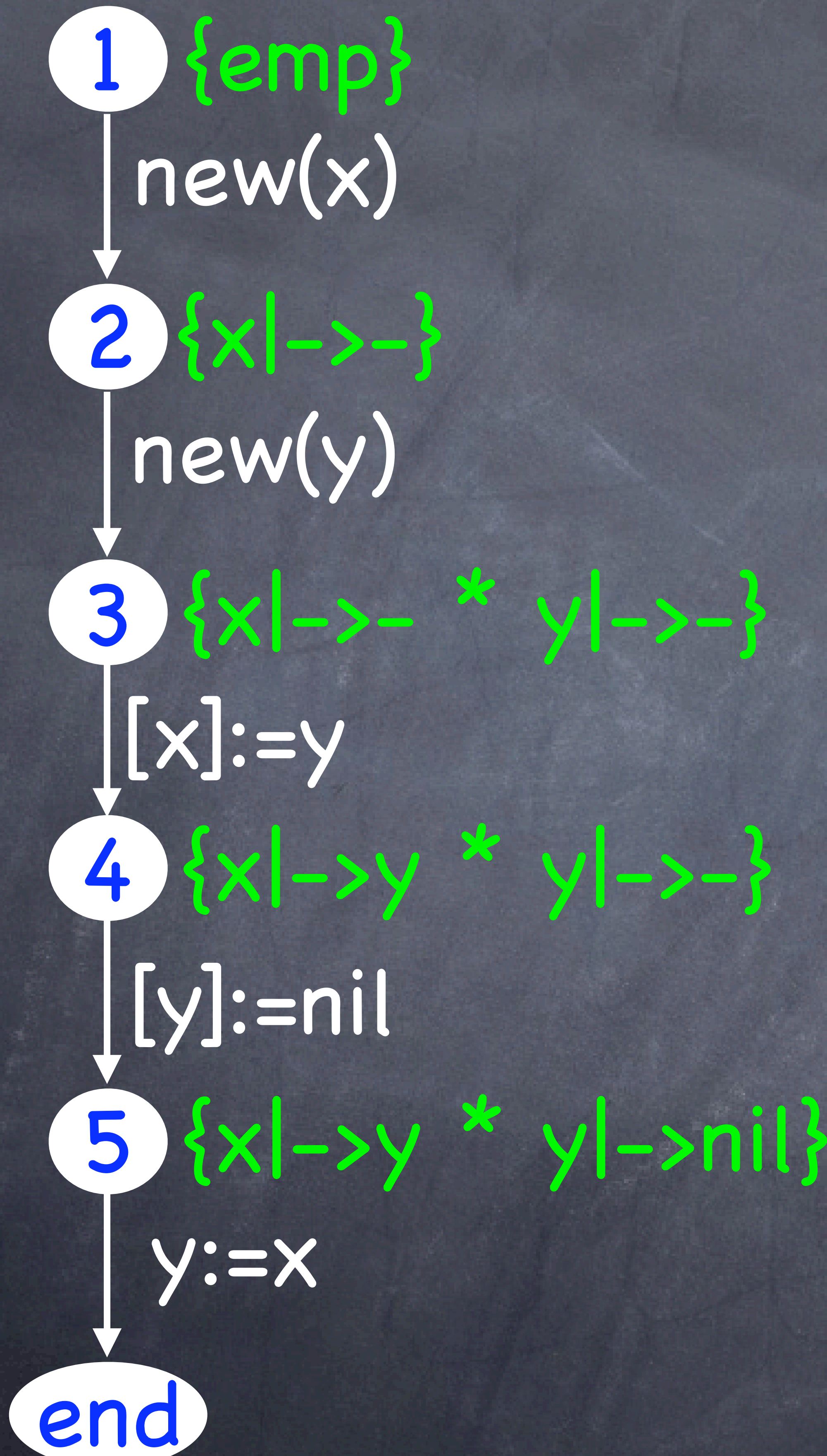


$\Pi\|\Sigma,$	$x := E$	$\Longrightarrow$	$x = E[x'/x] \wedge (\Pi\|\Sigma)[x'/x]$
$\Pi\|\Sigma * E \mapsto F,$	$x := [E]$	$\Longrightarrow$	$x = F[x'/x] \wedge (\Pi\|\Sigma * E \mapsto F)[x'/x]$
$\Pi\|\Sigma * E \mapsto F,$	$[E] := G$	$\Longrightarrow$	$\Pi\|\Sigma * E \mapsto G$
$\Pi;\Sigma,$	$\mathsf{new}(x)$	$\Longrightarrow$	$(\Pi\|\Sigma)[x'/x] * x \mapsto y'$
$\Pi\|\Sigma * E \mapsto F,$	$\mathsf{dispose}(E)$	$\Longrightarrow$	$\Pi\|\Sigma$

$$\frac{\Pi|\Sigma \nvdash \mathsf{Allocated}(E)}{\Pi|\Sigma,\ A(E) \ \Longrightarrow \top}$$

```
new(x);
new(y);
[x]:=y;
[y]:=nil;
y:=x;
```

Example 1



$\Pi\|\Sigma,$	$x := E$	$\Longrightarrow$	$x = E[x'/x] \wedge (\Pi\|\Sigma)[x'/x]$
$\Pi\|\Sigma * E \mapsto F,$	$x := [E]$	$\Longrightarrow$	$x = F[x'/x] \wedge (\Pi\|\Sigma * E \mapsto F)[x'/x]$
$\Pi\|\Sigma * E \mapsto F,$	$[E] := G$	$\Longrightarrow$	$\Pi\|\Sigma * E \mapsto G$
$\Pi; \Sigma,$	$\mathsf{new}(x)$	$\Longrightarrow$	$(\Pi\|\Sigma)[x'/x] * x \mapsto y'$
$\Pi\|\Sigma * E \mapsto F,$	$\mathsf{dispose}(E)$	$\Longrightarrow$	$\Pi\|\Sigma$

$$\frac{\Pi|\Sigma \nvdash \mathsf{Allocated}(E)}{\Pi|\Sigma,\ A(E)\ \Longrightarrow \top}$$

new(x);
new(y);
[x]:=y;
[y]:=nil;
y:=x;

Example 1

1 {emp}

new(x)

2 {x|->-}

new(y)

3 {x|->- * y|->-}

[x]:=y

4 {x|->y * y|->-}

[y]:=nil

5 {x|->y * y|->nil}

y:=x

end {x=y /\ x|->y' * y'|->nil}

$\Pi\|\Sigma,$	$x := E$	$\Longrightarrow$	$x = E[x'/x] \wedge (\Pi\|\Sigma)[x'/x]$
$\Pi\|\Sigma * E \mapsto F,$	$x := [E]$	$\Longrightarrow$	$x = F[x'/x] \wedge (\Pi\|\Sigma * E \mapsto F)[x'/x]$
$\Pi\|\Sigma * E \mapsto F,$	$[E] := G$	$\Longrightarrow$	$\Pi\|\Sigma * E \mapsto G$
$\Pi; \Sigma,$	$\mathsf{new}(x)$	$\Longrightarrow$	$(\Pi\|\Sigma)[x'/x] * x \mapsto y'$
$\Pi\|\Sigma * E \mapsto F,$	$\mathsf{dispose}(E)$	$\Longrightarrow$	$\Pi\|\Sigma$

$$\frac{\Pi|\Sigma \nvdash \mathsf{Allocated}(E)}{\Pi|\Sigma,\ A(E) \implies \top}$$

new(x);
new(y);
[x]:=y;
[y]:=nil;
y:=x;

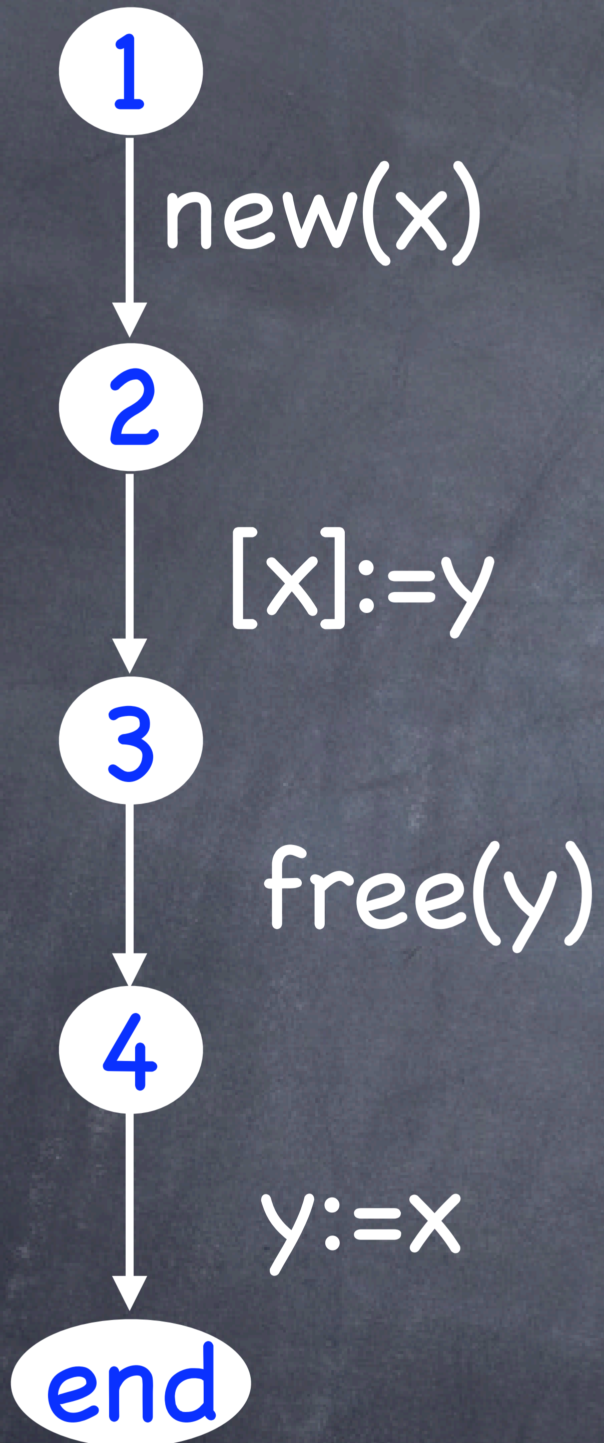
Example 2

$$
\begin{array}{llll}
\Pi|\Sigma, & x := E & \Longrightarrow & x = E[x'/x] \wedge (\Pi|\Sigma)[x'/x] \\
\Pi|\Sigma * E \mapsto F, & x := [E] & \Longrightarrow & x = F[x'/x] \wedge (\Pi|\Sigma * E \mapsto F)[x'/x] \\
\Pi|\Sigma * E \mapsto F, & [E] := G & \Longrightarrow & \Pi|\Sigma * E \mapsto G \\
\Pi; \Sigma, & \mathsf{new}(x) & \Longrightarrow & (\Pi|\Sigma)[x'/x] * x \mapsto y' \\
\Pi|\Sigma * E \mapsto F, & \mathsf{dispose}(E) & \Longrightarrow & \Pi|\Sigma
\end{array}
$$

$$
\frac{\Pi|\Sigma \nvdash \mathsf{Allocated}(E)}{\Pi|\Sigma,\ A(E) \ \Longrightarrow \top}
$$

```
new(x);
[x]:=y;
free(y);
y:=x;
```

Example 2



$$\Pi|\Sigma, \quad x := E \quad \Longrightarrow \quad x = E[x'/x] \wedge (\Pi|\Sigma)[x'/x]$$

$$\Pi|\Sigma * E \mapsto F, \quad x := [E] \quad \Longrightarrow \quad x = F[x'/x] \wedge (\Pi|\Sigma * E \mapsto F)[x'/x]$$

$$\Pi|\Sigma * E \mapsto F, \quad [E] := G \quad \Longrightarrow \quad \Pi|\Sigma * E \mapsto G$$

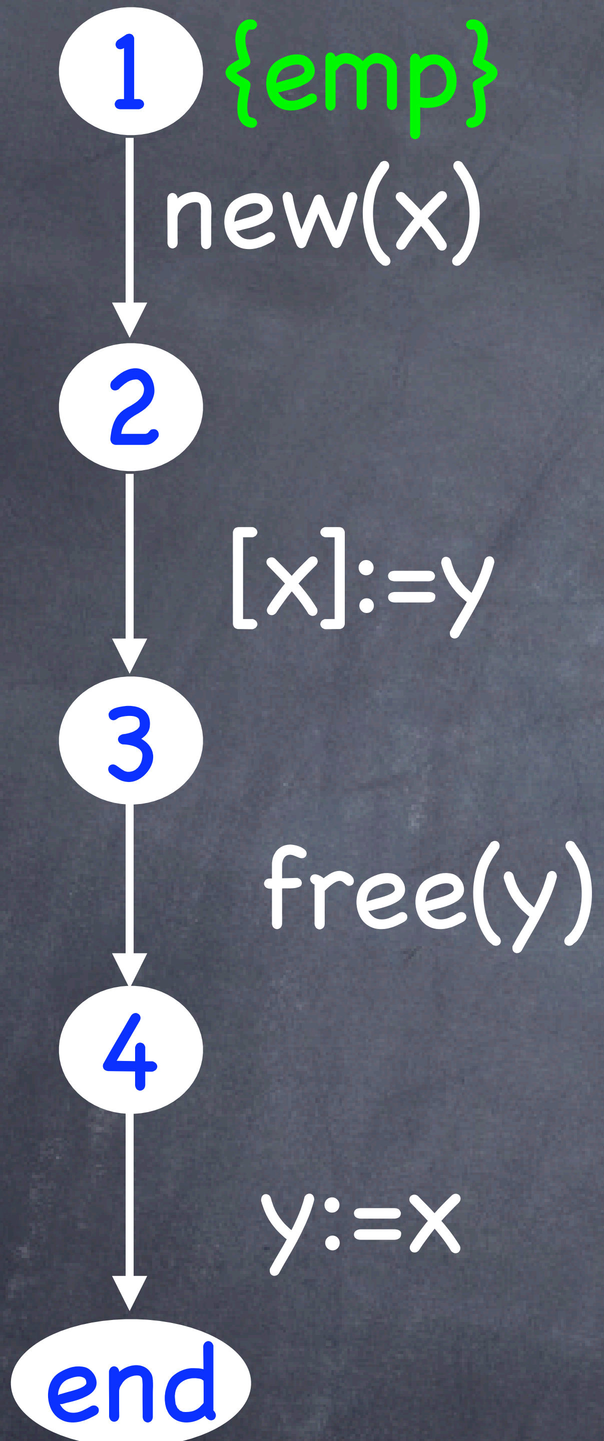
$$\Pi;\Sigma, \quad \mathsf{new}(x) \quad \Longrightarrow \quad (\Pi|\Sigma)[x'/x] * x \mapsto y'$$

$$\Pi|\Sigma * E \mapsto F, \quad \mathsf{dispose}(E) \quad \Longrightarrow \quad \Pi|\Sigma$$

$$\frac{\Pi|\Sigma \nvdash \mathsf{Allocated}(E)}{\Pi|\Sigma,\ A(E) \ \Longrightarrow \top}$$

```
new(x);
[x]:=y;
free(y);
y:=x;
```

Example 2

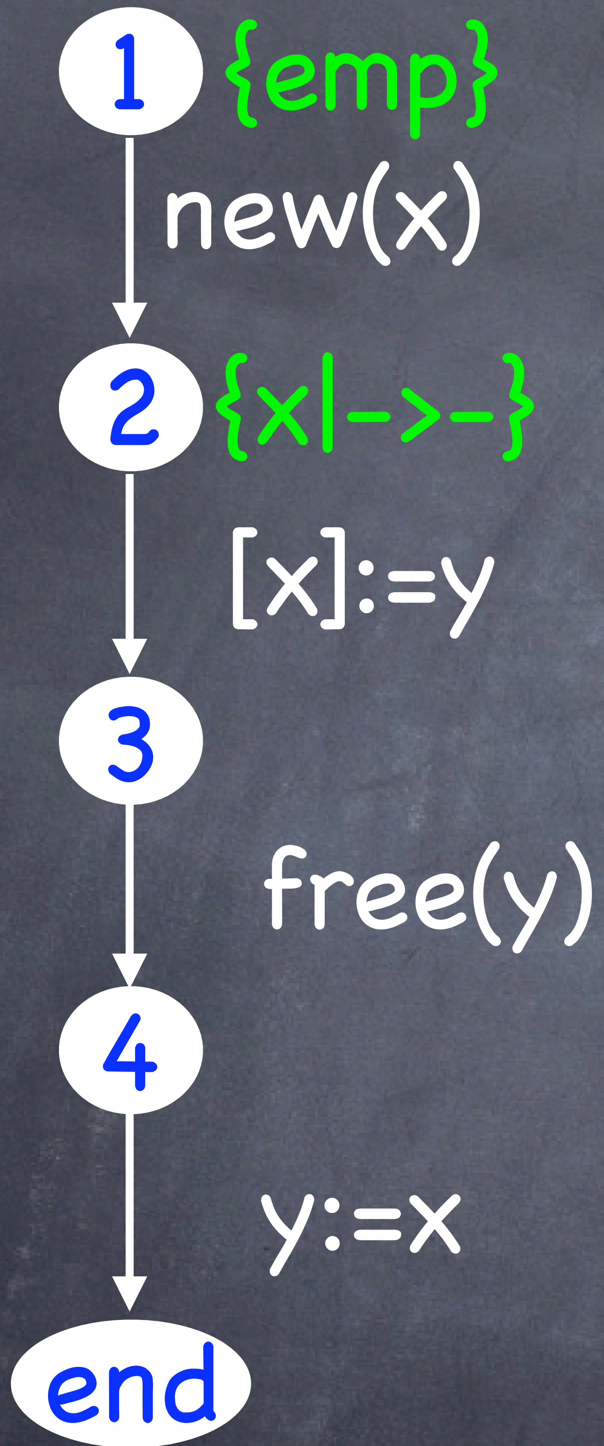


$\Pi\|\Sigma,$	$x := E$	$\Longrightarrow$	$x = E[x'/x] \wedge (\Pi\|\Sigma)[x'/x]$
$\Pi\|\Sigma * E \mapsto F,$	$x := [E]$	$\Longrightarrow$	$x = F[x'/x] \wedge (\Pi\|\Sigma * E \mapsto F)[x'/x]$
$\Pi\|\Sigma * E \mapsto F,$	$[E] := G$	$\Longrightarrow$	$\Pi\|\Sigma * E \mapsto G$
$\Pi; \Sigma,$	$\mathsf{new}(x)$	$\Longrightarrow$	$(\Pi\|\Sigma)[x'/x] * x \mapsto y'$
$\Pi\|\Sigma * E \mapsto F,$	$\mathsf{dispose}(E)$	$\Longrightarrow$	$\Pi\|\Sigma$

$$\frac{\Pi|\Sigma \nvdash \mathsf{Allocated}(E)}{\Pi|\Sigma,\ A(E) \ \Longrightarrow \top}$$

new(x);
[x]:=y;
free(y);
y:=x;

Example 2

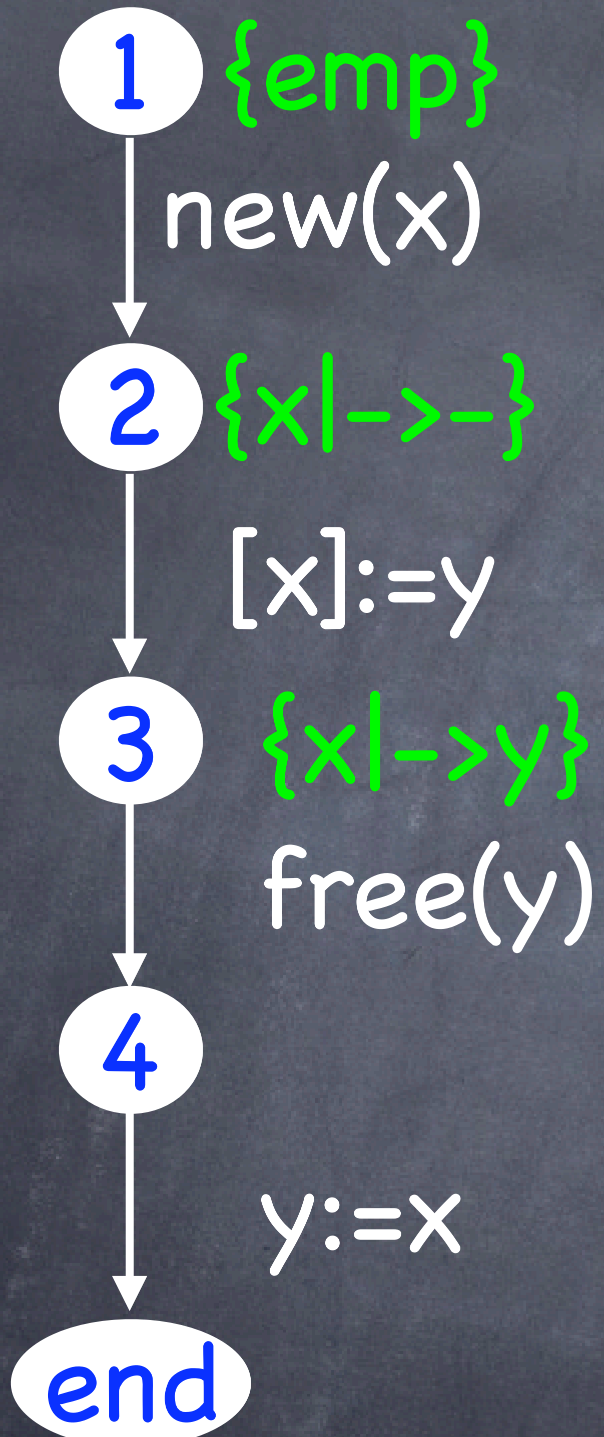


$\Pi \vert \Sigma,$	$x := E$	$\Longrightarrow$	$x = E[x'/x] \wedge (\Pi \vert \Sigma)[x'/x]$
$\Pi \vert \Sigma * E \mapsto F,$	$x := [E]$	$\Longrightarrow$	$x = F[x'/x] \wedge (\Pi \vert \Sigma * E \mapsto F)[x'/x]$
$\Pi \vert \Sigma * E \mapsto F,$	$[E] := G$	$\Longrightarrow$	$\Pi \vert \Sigma * E \mapsto G$
$\Pi; \Sigma,$	$\mathsf{new}(x)$	$\Longrightarrow$	$(\Pi \vert \Sigma)[x'/x] * x \mapsto y'$
$\Pi \vert \Sigma * E \mapsto F,$	$\mathsf{dispose}(E)$	$\Longrightarrow$	$\Pi \vert \Sigma$

$$\frac{\Pi|\Sigma \nvdash \mathsf{Allocated}(E)}{\Pi|\Sigma,\ A(E) \Longrightarrow \top}$$

new(x);
[x]:=y;
free(y);
y:=x;

Example 2

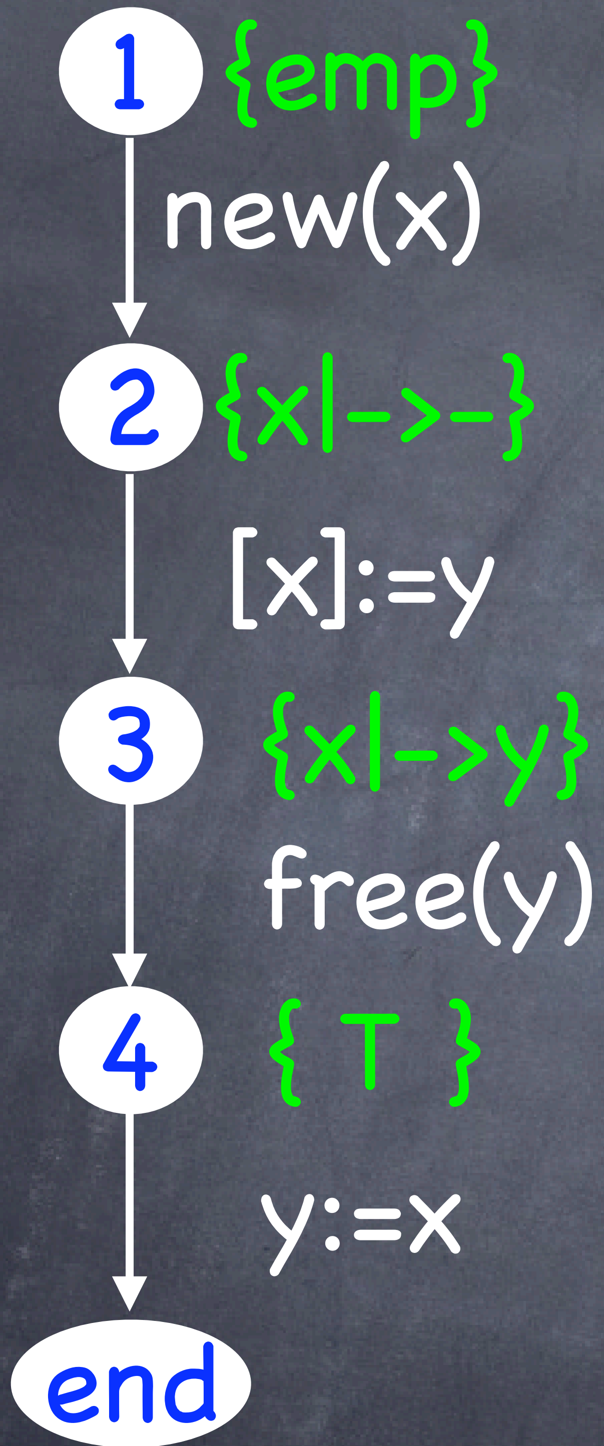


$\Pi \mid \Sigma,$	$x := E$	$\Longrightarrow$	$x = E[x'/x] \wedge (\Pi \mid \Sigma)[x'/x]$
$\Pi \mid \Sigma * E \mapsto F,$	$x := [E]$	$\Longrightarrow$	$x = F[x'/x] \wedge (\Pi \mid \Sigma * E \mapsto F)[x'/x]$
$\Pi \mid \Sigma * E \mapsto F,$	$[E] := G$	$\Longrightarrow$	$\Pi \mid \Sigma * E \mapsto G$
$\Pi; \Sigma,$	$\mathsf{new}(x)$	$\Longrightarrow$	$(\Pi \mid \Sigma)[x'/x] * x \mapsto y'$
$\Pi \mid \Sigma * E \mapsto F,$	$\mathsf{dispose}(E)$	$\Longrightarrow$	$\Pi \mid \Sigma$

$$\frac{\Pi \mid \Sigma \nvdash \mathsf{Allocated}(E)}{\Pi \mid \Sigma,\ A(E) \ \Longrightarrow \top}$$

```
new(x);
[x]:=y;
free(y);
y:=x;
```

Example 2



$$
\begin{array}{llll}
\Pi|\Sigma, & x := E & \Longrightarrow & x = E[x'/x] \wedge (\Pi|\Sigma)[x'/x] \\
\Pi|\Sigma * E \mapsto F, & x := [E] & \Longrightarrow & x = F[x'/x] \wedge (\Pi|\Sigma * E \mapsto F)[x'/x] \\
\Pi|\Sigma * E \mapsto F, & [E] := G & \Longrightarrow & \Pi|\Sigma * E \mapsto G \\
\Pi;\Sigma, & \mathsf{new}(x) & \Longrightarrow & (\Pi|\Sigma)[x'/x] * x \mapsto y' \\
\Pi|\Sigma * E \mapsto F, & \mathsf{dispose}(E) & \Longrightarrow & \Pi|\Sigma
\end{array}
$$

$$
\frac{\Pi|\Sigma \nvdash \mathsf{Allocated}(E)}{\Pi|\Sigma,\ A(E) \Longrightarrow \top}
$$

```
new(x);
[x]:=y;
free(y);
y:=x;
```

Entailment

- During symbolic execution we need to compute entailments P |- Q
  - e.g. P |- E=F ???
- In a tool we need to compute them automatically.

Entailments

Berdine/Calcagno proof theory

Subtraction rule

$$\frac{Q1 \vdash Q2}{Q1 * S \vdash Q2 * S}$$

Sample abstraction rule

lseg(x,t)*list(t) |- list(x)

Try to reduce to axiom:

$$\frac{}{B \wedge emp \vdash true \wedge emp}$$

Example

lseg(x,t) * t|->y * list(y) |- list(x)

Example

lseg(x,t) * t|->y * list(y) |- list(x)

Example

lseg(x,t) * t|->y * list(y) |- list(x)   (Abstraction Roll)

Example

lseg(x,t) * list(t) |- list(x)

lseg(x,t) * t|->y * list(y) |- list(x) (Abstraction Roll)

Example

lseg(x,t) * list(t) |- list(x)

lseg(x,t) * t|->y * list(y) |- list(x)   (Abstraction Roll)

Example

lseg(x,t) * list(t) |- list(x) (Abstraction Inductive)

lseg(x,t) * t|->y * list(y) |- list(x) (Abstraction Roll)

Example

list(x) |- list(x)

lseg(x,t) * list(t) |- list(x) (Abstraction Inductive)

lseg(x,t) * t|->y * list(y) |- list(x) (Abstraction Roll)

Example

list(x) |- list(x)

lseg(x,t) * list(t) |- list(x) (Abstraction Inductive)

lseg(x,t) * t|->y * list(y) |- list(x) (Abstraction Roll)

Example

list(x) |- list(x) (Substract)

lseg(x,t) * list(t) |- list(x) (Abstraction Inductive)

lseg(x,t) * t|->y * list(y) |- list(x) (Abstraction Roll)

Example

emp |- emp

list(x) |- list(x) (Substract)

lseg(x,t) * list(t) |- list(x) (Abstraction Inductive)

lseg(x,t) * t|->y * list(y) |- list(x) (Abstraction Roll)

Example

emp |- emp	(Axiom)
list(x) |- list(x)	(Substract)
lseg(x,t) * list(t) |- list(x)	(Abstraction Inductive)
lseg(x,t) * t|->y * list(y) |- list(x)	(Abstraction Roll)

Example

✓

emp |- emp (Axiom)

list(x) |- list(x) (Substract)

lseg(x,t) * list(t) |- list(x) (Abstraction Inductive)

lseg(x,t) * t|->y * list(y) |- list(x) (Abstraction Roll)

Example

lseg(x,t) * t|->nil * list(y) |- list(x)

Example

lseg(x,t) * t|->nil * list(y) |- list(x)

Example

lseg(x,t) * t|->nil * list(y) |- list(x) (Abstract inductive)

Example

list(x) * list(y) |- list(x)

lseg(x,t) * t|->nil * list(y) |- list(x)   (Abstract inductive)

Example

list(x) * list(y) |- list(x)

lseg(x,t) * t|->nil * list(y) |- list(x)   (Abstract inductive)

Example

list(x) * list(y) |- list(x)   (Subtract)

lseg(x,t) * t|->nil * list(y) |- list(x)   (Abstract inductive)

Example

list(y) |- emp

list(x) * list(y) |- list(x) (Subtract)

lseg(x,t) * t|->nil * list(y) |- list(x) (Abstract inductive)

Example

list(y) |- emp (No Axiom!)

list(x) * list(y) |- list(x) (Subtract)

lseg(x,t) * t|->nil * list(y) |- list(x) (Abstract inductive)

Example

list(y) |- emp (No Axiom!)

list(x) * list(y) |- list(x) (Subtract)

lseg(x,t) * t|->nil * list(y) |- list(x) (Abstract inductive)

Remarks

- Berdine/Calcagno gave a proof procedure that was cubic and complete on certain formulae (simple lists only).

- Abstract interpreters based on sep logic: Space Invader, SLAyer, THOR, jStar, use special versions of the abstraction rules to ensure convergence (we will see it in another lecture)

Automating proofs

Specification $\{tree(p)\}$ DisposeTree(p) $\{emp\}$

$\{tree(i)*tree(j)\}$
DisposeTree(i);

DisposeTree(j);

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

Automating proofs

Specification $\{tree(p)\}$ DisposeTree(p) $\{emp\}$

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

DisposeTree(i);

DisposeTree(j);

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

Automating proofs

Specification {tree(p)} DisposeTree(p) {emp}

{tree(i)*tree(j)}
DisposeTree(i);
{emp*tree(j)}
DisposeTree(j);

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

Automating proofs

Specification {tree(p)} DisposeTree(p) {emp}

{tree(i)*tree(j)}
DisposeTree(i);
{emp*tree(j)}
DisposeTree(j);

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

Automating proofs

Specification {tree(p)} DisposeTree(p) {emp}

{tree(i)*tree(j)}
DisposeTree(i);
{emp*tree(j)}
DisposeTree(j);
{emp*emp}

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

Automating proofs

Specification {tree(p)} DisposeTree(p) {emp}

{tree(i)*tree(j)}
DisposeTree(i);
{emp*tree(j)}
DisposeTree(j);
{emp*emp}
{emp}

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

Frame inference problem

Given A and B find X such that:

A |- B * X

Example:

tree(i)*tree(j) |- tree(i) *X

Frame inference problem

Given A and B find X such that:

A |- B * X

Example:

tree(i)*tree(j) |- tree(i) *tree(j)

How to infer the frame

lseg(x,t) * t|->nil * list(y) |- list(x)

How to infer the frame

lseg(x,t) * t|->nil * list(y) |- list(x)   (Abstract inductive)

How to infer the frame

list(x) * list(y) |- list(x)

lseg(x,t) * t|->nil * list(y) |- list(x)   (Abstract inductive)

How to infer the frame

list(x) * list(y) |- list(x) (Subtract)

lseg(x,t) * t|->nil * list(y) |- list(x) (Abstract inductive)

How to infer the frame

list(y) |- emp

list(x) * list(y) |- list(x) (Subtract)

lseg(x,t) * t|->nil * list(y) |- list(x) (Abstract inductive)

How to infer the frame

list(y) |- emp	(No Axiom!)
list(x) * list(y) |- list(x)	(Subtract)
lseg(x,t) * t|->nil * list(y) |- list(x)	(Abstract inductive)

How to infer the frame

list(y) |- emp	(No Axiom!)
list(x) * list(y) |- list(x)	(Subtract)
lseg(x,t) * t|->nil * list(y) |- list(x)	(Abstract inductive)

How to infer the frame

list(y) |- emp (No Axiom!)

list(x) * list(y) |- list(x) (Subtract)

lseg(x,t) * t|->nil * list(y) |- list(x) (Abstract inductive)

✓

emp |- emp

How to infer the frame

✗

list(y) |- emp (No Axiom!)

list(x) * list(y) |- list(x) (Subtract)

lseg(x,t) * t|->nil * list(y) |- list(x) (Abstract inductive)

✓

emp |- emp (Axiom)

How to infer the frame

✗

list(y) |- emp (No Axiom!)

list(x) * list(y) |- list(x) (Subtract)

lseg(x,t) * t|->nil * list(y) |- list(x) (Abstract inductive)

✓

emp |- emp (Axiom)

list(y) |- list(y)

How to infer the frame

✗

list(y) |- emp	(No Axiom!)
list(x) * list(y) |- list(x)	(Subtract)
lseg(x,t) * t|->nil * list(y) |- list(x)	(Abstract inductive)

✓

emp |- emp	(Axiom)
list(y) |- list(y)	(Substract)

How to infer the frame

✗

list(y) |- emp	(No Axiom!)
list(x) * list(y) |- list(x)	(Subtract)
lseg(x,t) * t|->nil * list(y) |- list(x)	(Abstract inductive)

✓

emp |- emp	(Axiom)
list(y) |- list(y)	(Substract)
list(x)*list(y) |- list(x)*list(y)	

How to infer the frame

✗

list(y) |- emp (No Axiom!)

list(x) * list(y) |- list(x) (Subtract)

lseg(x,t) * t|->nil * list(y) |- list(x) (Abstract inductive)

✓

emp |- emp (Axiom)

list(y) |- list(y) (Substract)

list(x)*list(y) |- list(x)*list(y) (Substract)

How to infer the frame

✗

list(y) |- emp (No Axiom!)

list(x) * list(y) |- list(x) (Subtract)

lseg(x,t) * t|->nil * list(y) |- list(x) (Abstract inductive)

✓

emp |- emp (Axiom)

list(y) |- list(y) (Substract)

list(x)*list(y) |- list(x)*list(y) (Substract)

(Abstract inductive)

How to infer the frame

✗

list(y) |- emp (No Axiom!)

list(x) * list(y) |- list(x) (Subtract)

lseg(x,t) * t|->nil * list(y) |- list(x) (Abstract inductive)

✓

emp |- emp (Axiom)

list(y) |- list(y) (Substract)

list(x)*list(y) |- list(x)*list(y) (Substract)

lseg(x,t) * t|->nil * list(y) |- list(x)*list(y) (Abstract inductive)

General rule for inferring frame

We need to compute X in A |- B *X

Strategy: apply subtration and abstraction to srink the goal as much as you can

When we get X |- emp then X is our frame

X |- emp

⋮

A |- B

Homework

Compute the frame of the following entailment:

z|->nil * x|->y * y|->nil |- z|->nil

Using symbolic heaps, compute the symbolic execution of the program.

```
t:=p;
p:=c;
c:=[c];
[p]:=t;
```

starting with precondition c|->c'*c'|->nil

References

- J.Berdine and C. Calcago: Symbolic Execution with Separation Logic. APLAS 2005
- D. Distefano, P. O'Hearn, H. Yang: A Local Shape Analysis Based on Separation Logic. TACAS 2006.