Methodologies for Software Processes

Lecture 8- Program Verification using Separation Logic

(The lecture slides and the notes are taken from Dino Distefano from Queen Mary University of London)

Study Separation Logic having automatic verification in mind

Learn how some notions of mathematical logic can be very helpful in reasoning about real world programs

```
void t1394Diag_CancelIrp(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
  KIRQL               Irql, CancelIrql;
  BUS_RESET_IRP       *BusResetIrp, *temp;
  PDEVICE_EXTENSION   deviceExtension;

  deviceExtension = DeviceObject->DeviceExtension;

  KeAcquireSpinLock(&deviceExtension->ResetSpinLock, &Irql);

  temp = (PBUS_RESET_IRP)deviceExtension;
  BusResetIrp = (PBUS_RESET_IRP)deviceExtension->Flink2;

  while (BusResetIrp) {

    if (BusResetIrp->Irp == Irp) {
      temp->Flink2 = BusResetIrp->Flink2;
      free(BusResetIrp);
      break;
    }
    else if (BusResetIrp->Flink2 == (PBUS_RESET_IRP)deviceExtension) {
      break;
    }
    else {
      temp = BusResetIrp;
      BusResetIrp = (PBUS_RESET_IRP)BusResetIrp->Flink2;
    }
  }

  KeReleaseSpinLock(&deviceExtension->ResetSpinLock, Irql);

  IoReleaseCancelSpinLock(Irp->CancelIrql);
  Irp->IoStatus.Status = STATUS_CANCELLED;
  IoCompleteRequest(Irp, IO_NO_INCREMENT);
} // t1394Diag_CancelIrp
```

A piece of a windows device driver.

Is this correct?

Or at least: does it have basic properties like it won't crash or leak memory?

Today's plan

- Motivation for Separation Logic
- Assertion language
- Mathematical model
- Data structures


Motivations...

Simple Imperative Language

- Safe commands:
  - S::= skip | x:=E | x:=new(E1,...,En)
- Heap accessing commands:
  - A(E) ::= dispose(E) | x:=[E] | [E]:=F

where E is an expression e.g., x, y, nil, etc.

- Command:
  - C::= S | A | C1;C2 | if B { C1 } else {C2} | while B do { C }

where B boolean guard E=E, E!=E, etc.

Example Program: List Reversal

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

Example Program: List Reversal

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

Example Program: List Reversal

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

Example Program: List Reversal

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

Some properties we would like to prove:

Does the program preserve acyclicity/cyclicity?

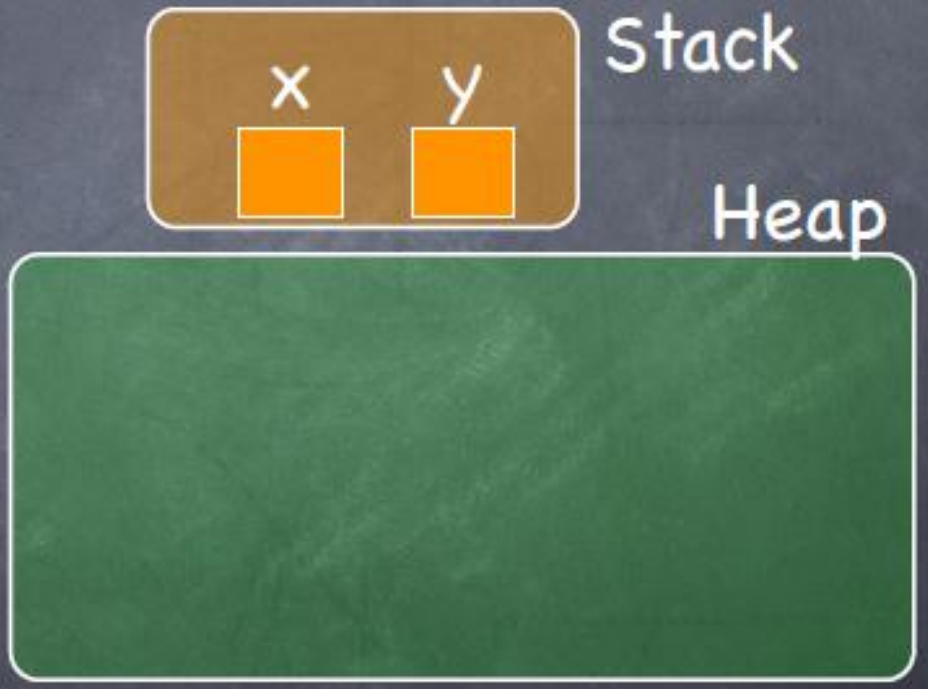
Does it core-dump?

Does it create garbage?

Example Program

We are interested in pointer manipulating programs

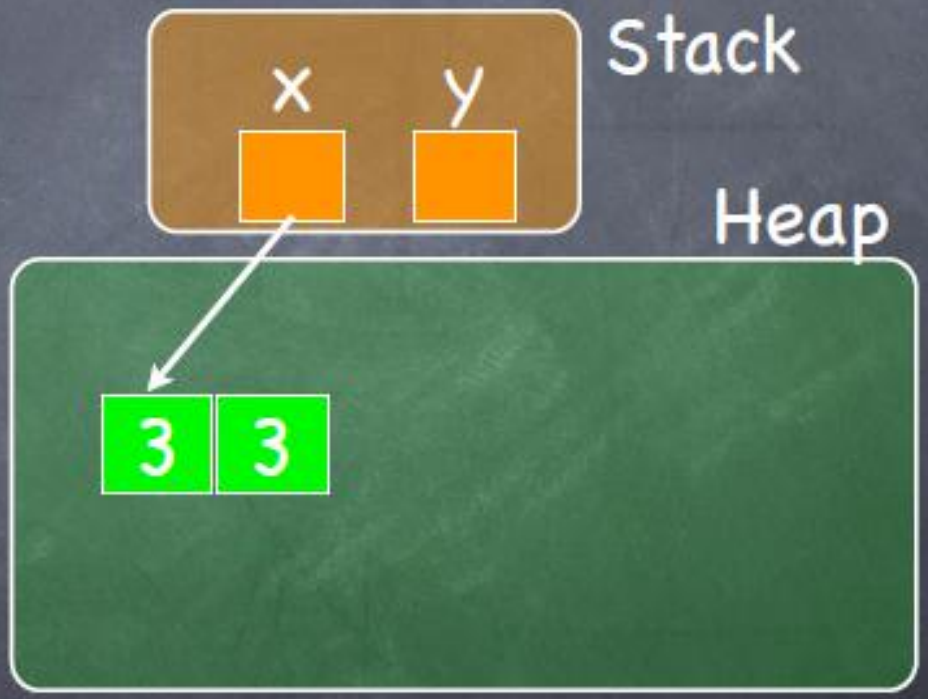
```
x = new(3,3);
y = new(4,4);
[x+1] = y;
[y+1] = x;
y = x+1;
dispose x;
y = [y];
```



Example Program

We are interested in pointer manipulating programs

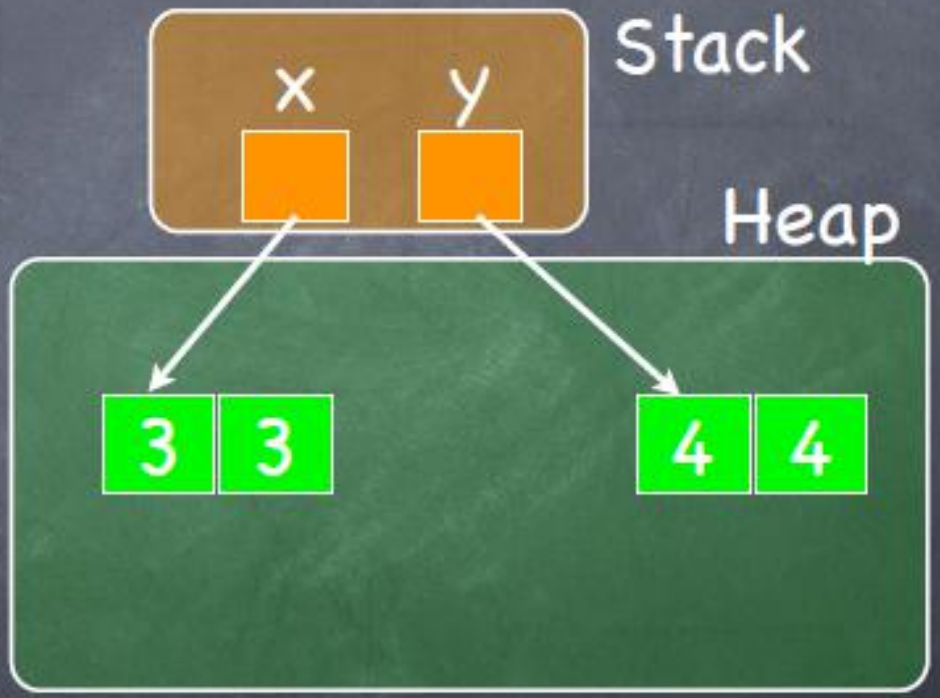
```
x = new(3,3);
y = new(4,4);
[x+1] = y;
[y+1] = x;
y = x+1;
dispose x;
y = [y];
```



Example Program

We are interested in pointer manipulating programs

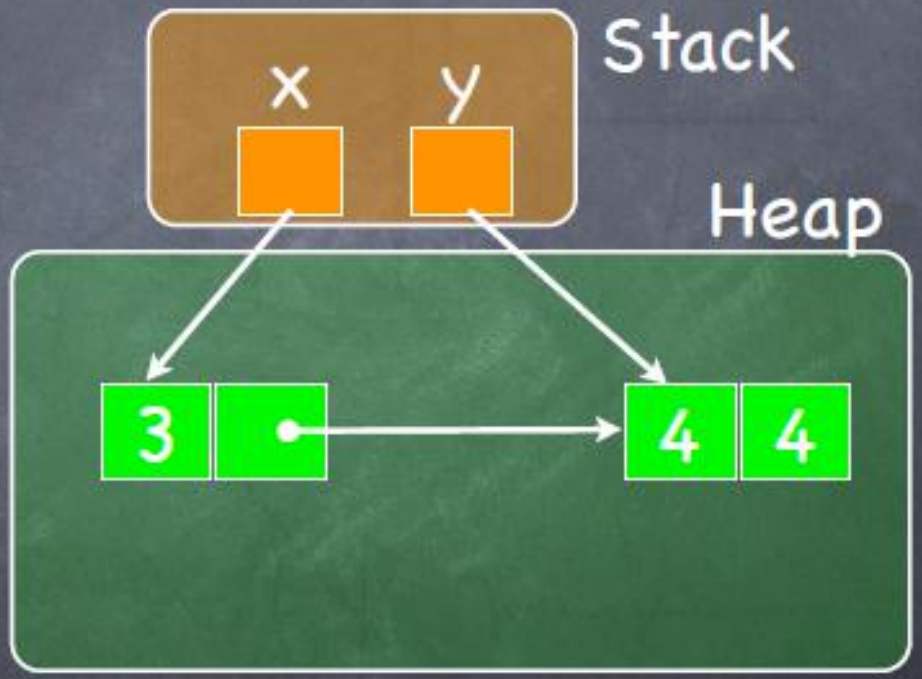
```
x = new(3,3);
y = new(4,4);
[x+1] = y;
[y+1] = x;
y = x+1;
dispose x;
y = [y];
```



Example Program

We are interested in pointer manipulating programs

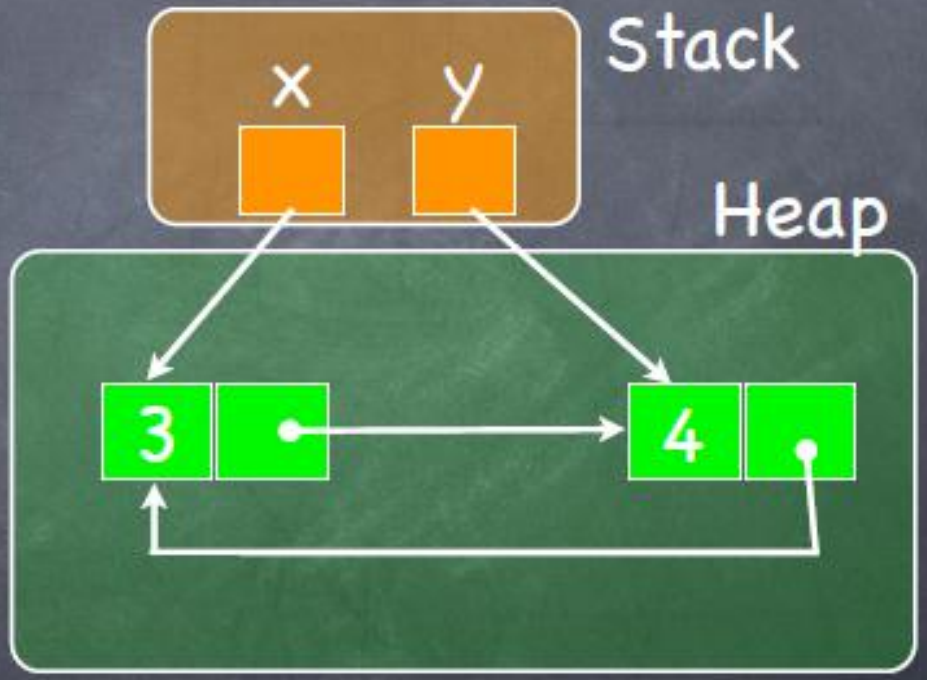
```
x = new(3,3);
y = new(4,4);
[x+1] = y;
[y+1] = x;
y = x+1;
dispose x;
y = [y];
```



Example Program

We are interested in pointer manipulating programs

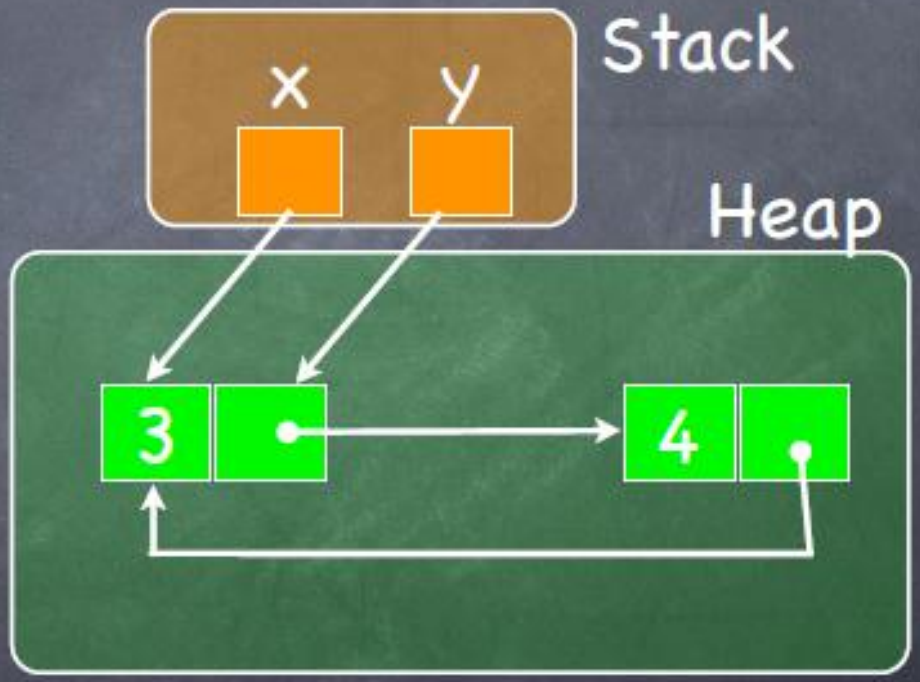
```
x = new(3,3);
y = new(4,4);
[x+1] = y;
[y+1] = x;
y = x+1;
dispose x;
y = [y];
```



Example Program

We are interested in pointer manipulating programs

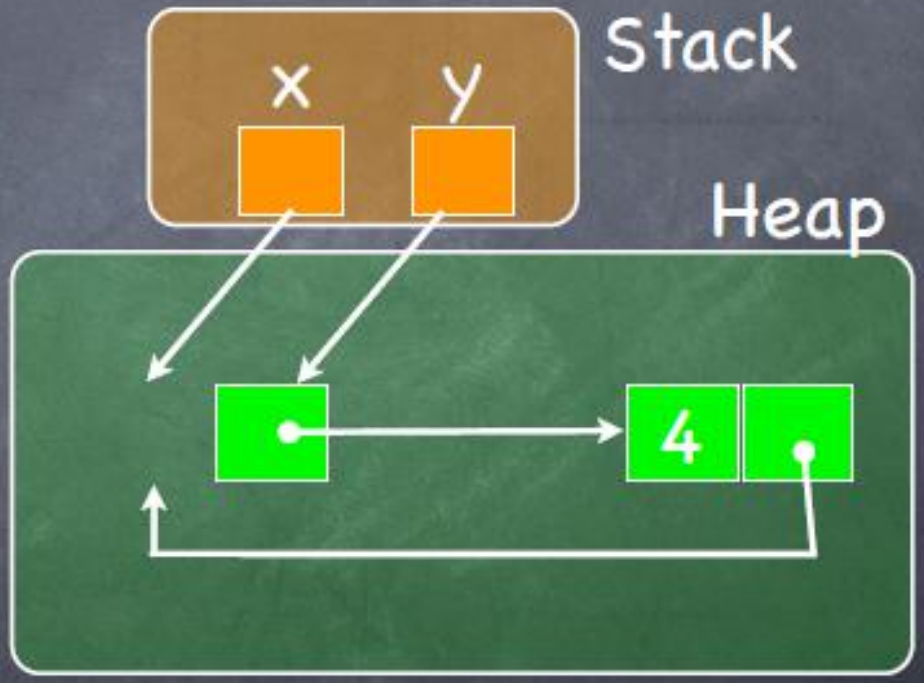
```
x = new(3,3);
y = new(4,4);
[x+1] = y;
[y+1] = x;
y = x+1;
dispose x;
y = [y];
```



Example Program

We are interested in pointer manipulating programs

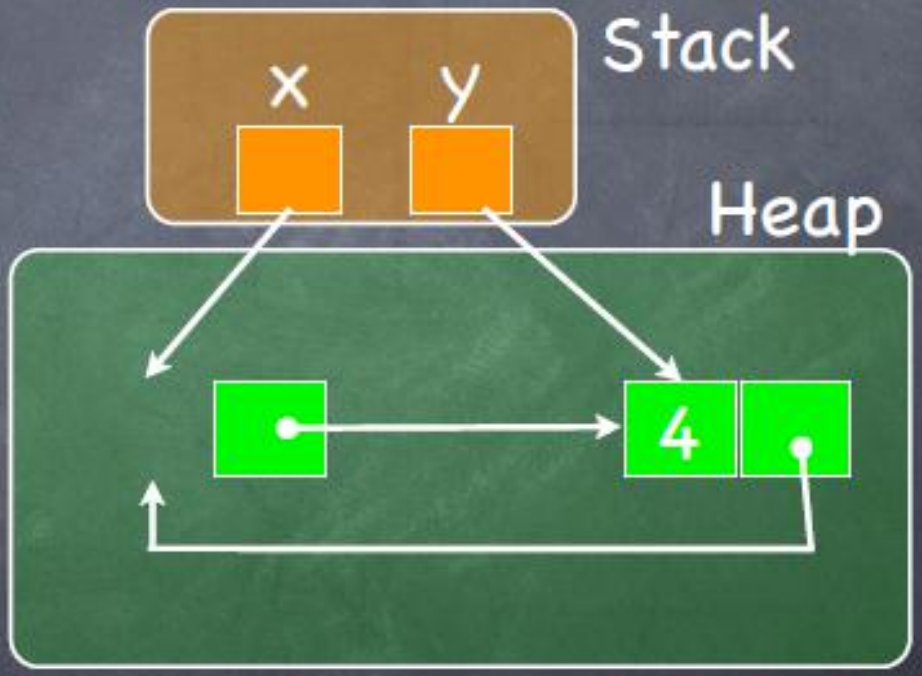
```
x = new(3,3);
y = new(4,4);
[x+1] = y;
[y+1] = x;
y = x+1;
dispose x;
y = [y];
```



Example Program

We are interested in pointer manipulating programs

```
x = new(3,3);
y = new(4,4);
[x+1] = y;
[y+1] = x;
y = x+1;
dispose x;
y = [y];
```



Why Separation Logic?

Consider this code:

```
[y] = 4;
[z] = 5;
```

Guarantee([y] != [z])

We need to know that things are different. How?

Why Separation Logic?

Consider this code:

```
Assume(y != z)
   [y] = 4;
   [z] = 5;
Guarantee([y] != [z])
```

Add assertion?

We need to know that things are different. How?

Why Separation Logic?

Consider this code:

```
Assume(y != z)
   [y] = 4;
   [z] = 5;
Guarantee([y] != [z])
```

Add assertion?

We need to know that things are different. How?

We need to know that things stay the same. How?

Why Separation Logic?

Consider this code:

```
Assume([x] = 3)
Assume(y != z)
   [y] = 4;
   [z] = 5;
Guarantee([y] != [z])
Guarantee([x] = 3)
```

Add assertion?

We need to know that things are different. How?

We need to know that things stay the same. How?

Why Separation Logic?

Consider this code:

```
Assume([x] = 3 && x!=y && x!=z)
Assume(y != z)
    [y] = 4;
    [z] = 5;
Guarantee([y] != [z])
Guarantee([x] = 3)
```

Add assertion?

Add assertion?

We need to know that things are different. How?

We need to know that things stay the same. How?

Framing

We want a general concept of things not being affected.

$$\frac{\{P\}\ C\ \{Q\}}{\{R\ \&\&\ P\}\ C\ \{Q\ \&\&\ R\}}$$

What are the conditions on C and R?

Hard to define if reasoning about a heap and aliasing

Framing

We want a general concept of things not being affected.

$$\frac{\{P\}\ C\ \{Q\}}{\{R\ \&\&\ P\ \}\ C\ \{Q\ \&\&\ R\ \}}$$

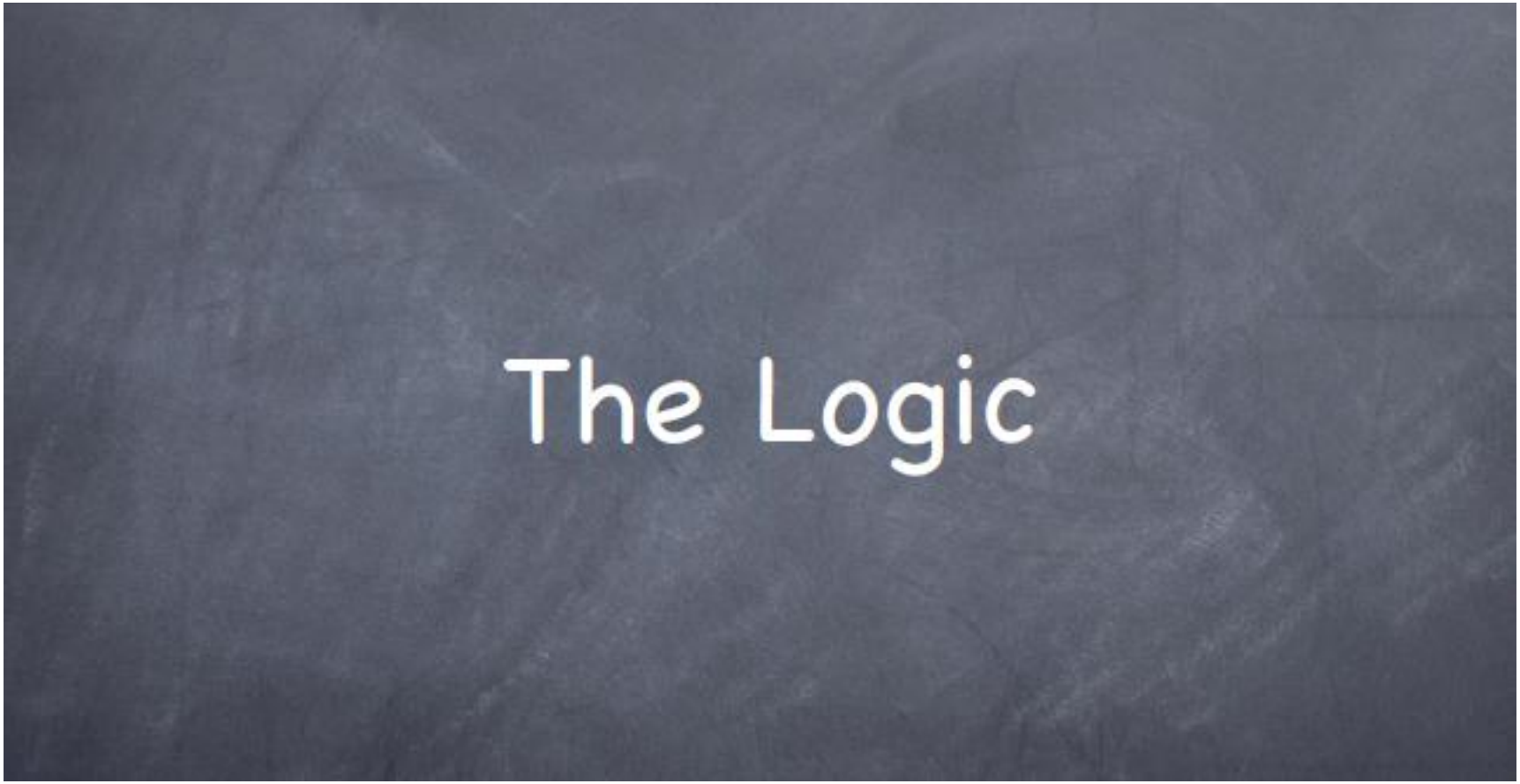
What are the conditions on C and R?

Hard to define if reasoning about a heap and aliasing

This is where separation logic comes in

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

Introduces new connective * used to separate state.


The Logic

Storage Model

$$\mathsf{Vars} \stackrel{def}{=} \{x, y, z, \ldots\}$$

$$\mathsf{Locs} \stackrel{def}{=} \{1, 2, 3, 4, \ldots\} \qquad \mathsf{Vals} \supseteq \mathsf{Locs}$$

$$\mathsf{Heaps} \stackrel{def}{=} \mathsf{Locs} \rightarrow_{\mathsf{fin}} \mathsf{Vals}$$

$$\mathsf{Stacks} \stackrel{def}{=} \mathsf{Vars} \rightarrow \mathsf{Vals}$$

$$\mathsf{States} \stackrel{def}{=} \mathsf{Stacks} \times \mathsf{Heaps}$$

Storage Model

$$\text{Vars} \stackrel{def}{=} \{x, y, z, \ldots\}$$

$$\text{Locs} \stackrel{def}{=} \{1, 2, 3, 4, \ldots\} \qquad \text{Vals} \supseteq \text{Locs}$$

$$\text{Heaps} \stackrel{def}{=} \text{Locs} \rightarrow_{\text{fin}} \text{Vals}$$

$$\text{Stacks} \stackrel{def}{=} \text{Vars} \rightarrow \text{Vals}$$

$$\text{States} \stackrel{def}{=} \text{Stacks} \times \text{Heaps}$$

Storage Model

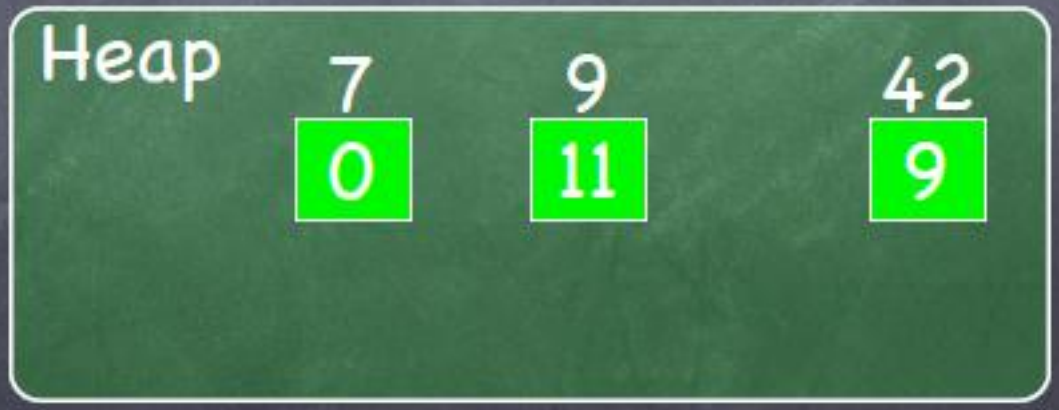
$$\mathsf{Vars} \stackrel{def}{=} \{x, y, z, \ldots\}$$

$$\mathsf{Locs} \stackrel{def}{=} \{1, 2, 3, 4, \ldots\} \qquad \mathsf{Vals} \supseteq \mathsf{Locs}$$

$$\mathsf{Heaps} \stackrel{def}{=} \mathsf{Locs} \rightarrow_{\mathsf{fin}} \mathsf{Vals}$$

$$\mathsf{Stacks} \stackrel{def}{=} \mathsf{Vars} \rightarrow \mathsf{Vals}$$

$$\mathsf{States} \stackrel{def}{=} \mathsf{Stacks} \times \mathsf{Heaps}$$



Storage Model

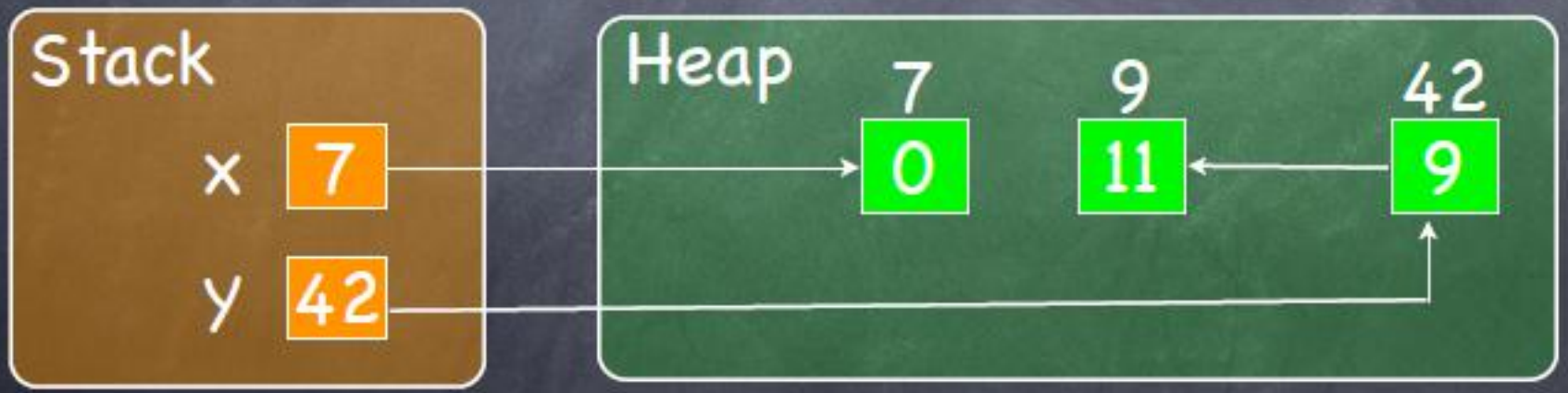
$$\text{Vars} \stackrel{def}{=} \{x, y, z, \ldots\}$$

$$\text{Locs} \stackrel{def}{=} \{1, 2, 3, 4, \ldots\} \qquad \text{Vals} \supseteq \text{Locs}$$

$$\text{Heaps} \stackrel{def}{=} \text{Locs} \rightarrow_{\text{fin}} \text{Vals}$$

$$\text{Stacks} \stackrel{def}{=} \text{Vars} \rightarrow \text{Vals}$$

$$\text{States} \stackrel{def}{=} \text{Stacks} \times \text{Heaps}$$



Mathematical Structure of Heap

$$\text{Heaps} \stackrel{def}{=} \text{Locs} \rightarrow_{\text{fin}} \text{Vals}$$

$$h_1 \# h_2 \stackrel{def}{\Longleftrightarrow} \text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset$$

$$h_1 * h_2 \stackrel{def}{=} \begin{cases} h_1 \cup h_2 & \text{if } h_1 \# h_2 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Mathematical Structure of Heap

$$\text{Heaps} \stackrel{def}{=} \text{Locs} \rightarrow_{\text{fin}} \text{Vals}$$

$$h_1 \# h_2 \stackrel{def}{\Longleftrightarrow} \text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset$$

$$h_1 * h_2 \stackrel{def}{=} \begin{cases} h_1 \cup h_2 & \text{if } h_1 \# h_2 \\ \text{undefined} & \text{otherwise} \end{cases}$$

1) * has a unit
2) * is associative and commutative
3) (Heap,*,{}) is a partial commutative monoid

Assertions

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

Informal Meaning

Assertions

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

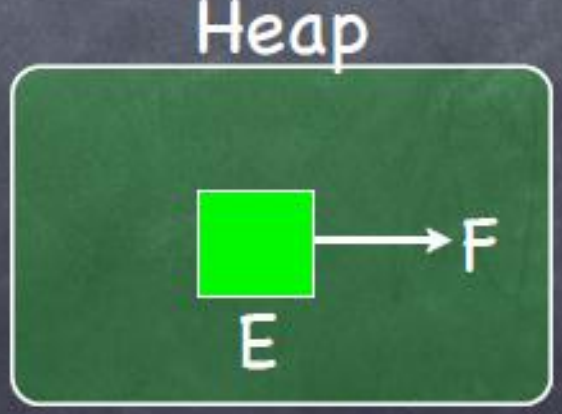
Informal Meaning

Heap

Assertions

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

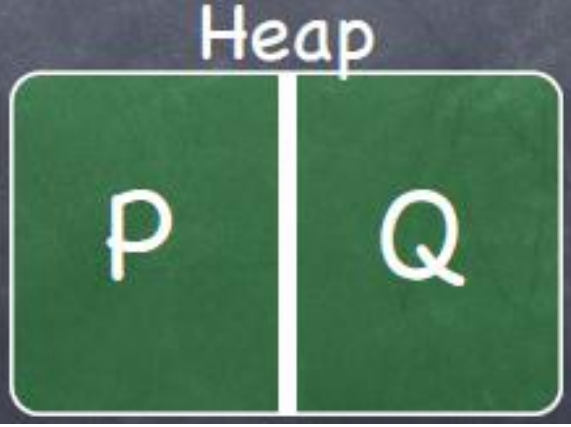
Informal Meaning



Assertions

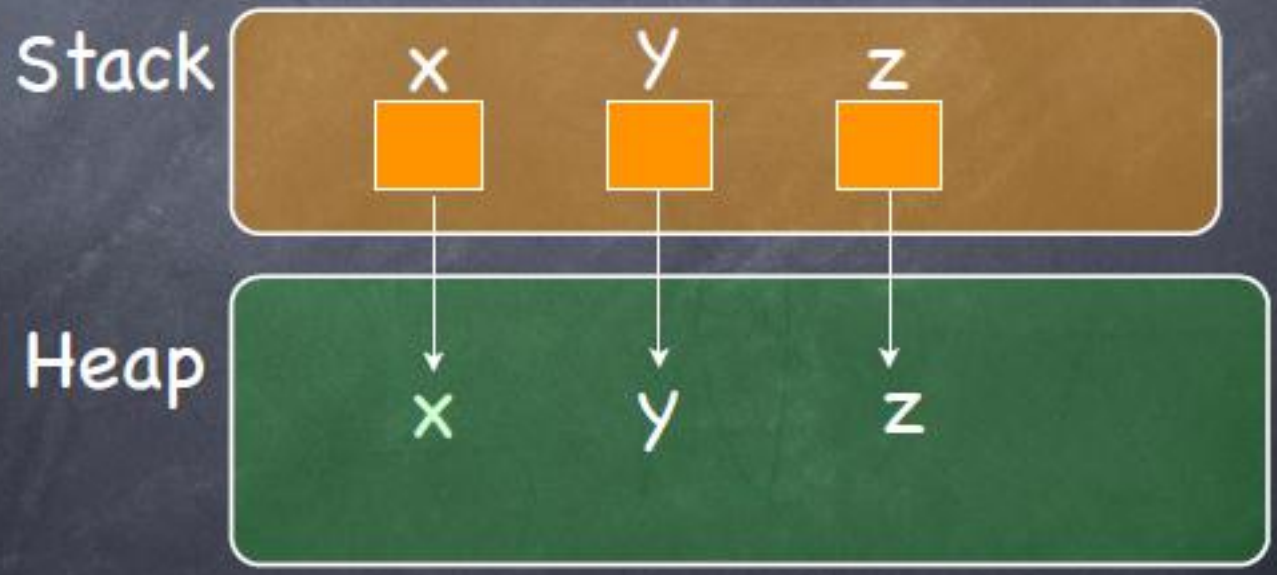
E, F	$::=$	$x \mid n \mid E+F \mid -E \mid \ldots$	Heap-independent Exprs
P, Q	$::=$	$E = F \mid E \geq F \mid E \mapsto F$	Atomic Predicates
	$\mid$	emp $\mid P * Q$	Separating Connectives
	$\mid$	true $\mid P \wedge Q \mid \neg P \mid \forall x.\, P$	Classical Logic

Informal Meaning



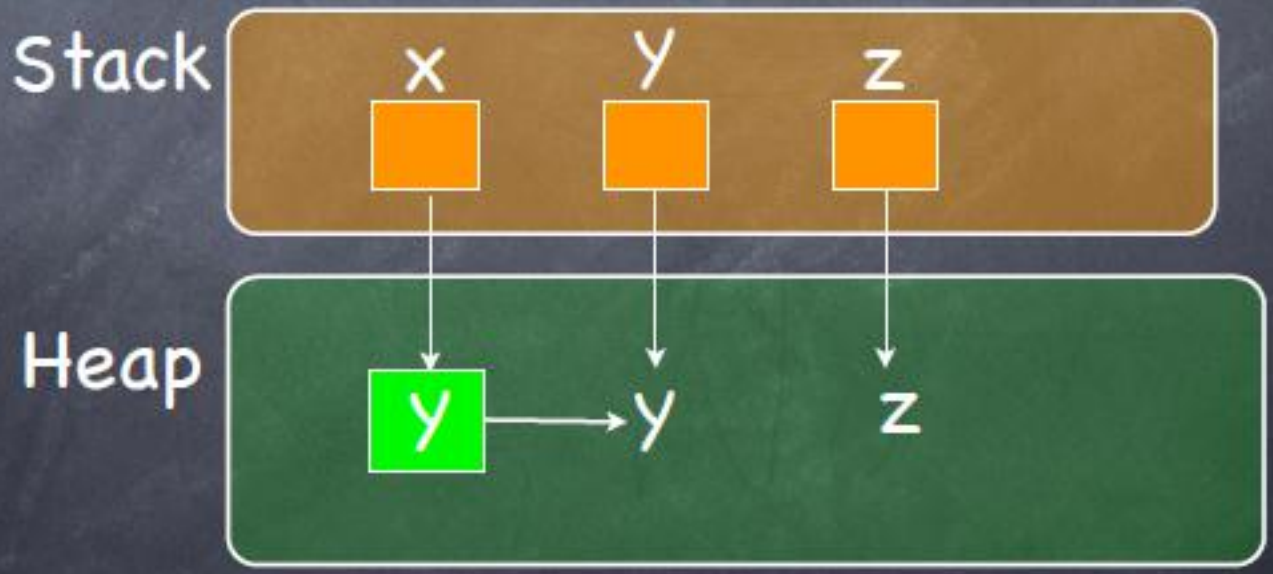
Examples

Formula: emp



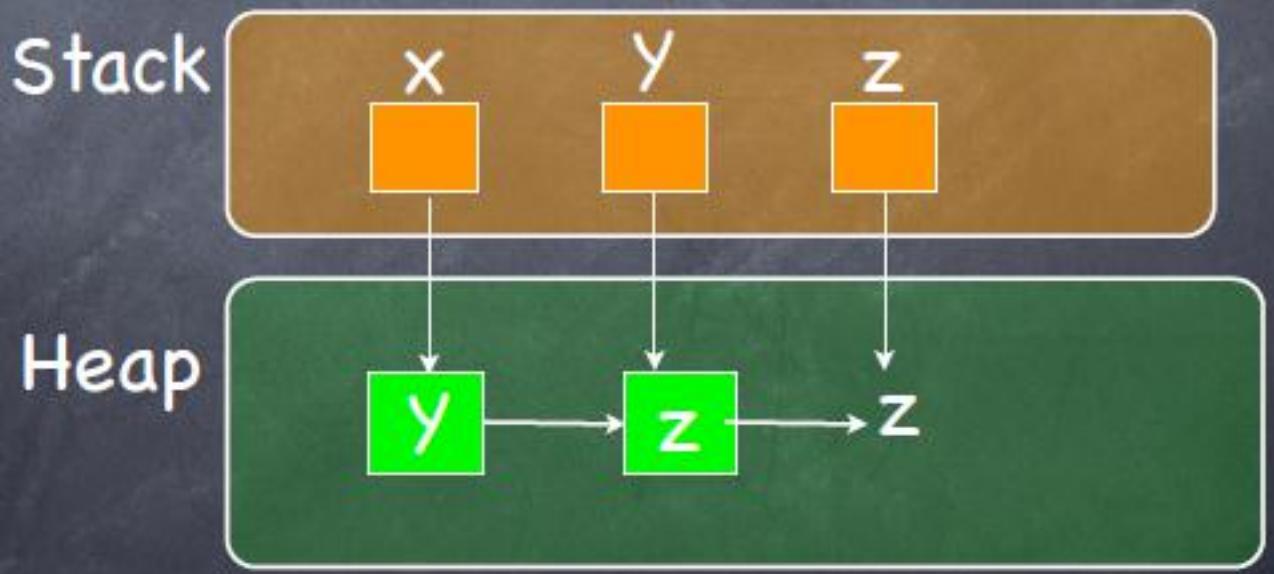
Examples

Formula: emp*x|->y



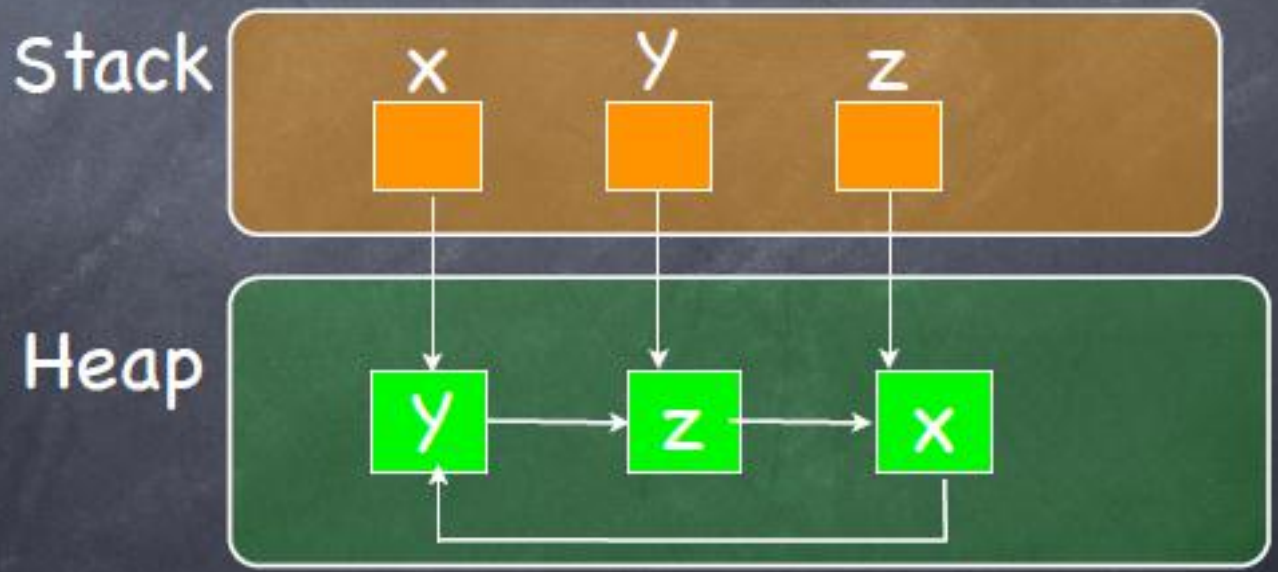
Examples

Formula: x|->y * y|->z



Examples

Formula: x|->y * y|->z * z|->x



Semantics of Assertions

- Expressions mean maps from stacks to integers.

$$[\![E]\!] \ : \ \text{Stacks} \rightarrow \text{Vals}$$

- Semantics of assertions given by satisfaction relation between states and assertions.

$$(s, h) \models P$$

Semantics of Assertions

- Expressions mean maps from stacks to integers.

$$[\![E]\!] \;:\; \text{Stacks} \to \text{Vals}$$

- Semantics of assertions given by satisfaction relation between states and assertions.

$$(s, h) \models P$$

Stack

Heap

Semantics of Assertions

$(s,h) \models E \geq F$	iff	$[\![E]\!]s, [\![F]\!]s \in$ Integers and $[\![E]\!]s \geq [\![F]\!]s$
$(s,h) \models E \mapsto F$	iff	$\mathsf{dom}(h) = \{[\![E]\!]s\}$ and $h([\![E]\!]s) = [\![F]\!]s$
$(s,h) \models \mathsf{emp}$	iff	$h = []$ (i.e., $\mathsf{dom}(h) = \emptyset$)
$(s,h) \models P * Q$	iff	$\exists h_0 h_1.\ h_0 * h_1 = h,\ (s,h_0) \models P$ and $(s,h_1) \models Q$
$(s,h) \models \mathsf{true}$		always
$(s,h) \models P \wedge Q$	iff	$(s,h) \models P$ and $(s,h) \models Q$
$(s,h) \models \neg P$	iff	not $((s,h) \models P)$
$(s,h) \models \forall x.\ P$	iff	$\forall v \in \mathsf{Vals}.\ (s[x \mapsto v], h) \models P$

Semantics of Assertions

$(s,h) \models E \geq F$ iff $[\![E]\!]s, [\![F]\!]s \in$ Integers and $[\![E]\!]s \geq [\![F]\!]s$

$(s,h) \models E \mapsto F$ iff $\mathsf{dom}(h) = \{[\![E]\!]s\}$ and $h([\![E]\!]s) = [\![F]\!]s$

$(s,h) \models \mathsf{emp}$ iff $h = []$ (i.e., $\mathsf{dom}(h) = \emptyset$)

$(s,h) \models P * Q$ iff $\exists h_0 h_1.\ h_0 * h_1 = h,\ (s,h_0) \models P$ and $(s,h_1) \models Q$

$(s,h) \models \mathsf{true}$ always

$(s,h) \models P \wedge Q$ iff $(s,h) \models P$ and $(s,h) \models Q$

$(s,h) \models \neg P$ iff not $((s,h) \models P)$

$(s,h) \models \forall x.\ P$ iff $\forall v \in \mathsf{Vals}.\ (s[x \mapsto v], h) \models P$

Semantics of Assertions

$$
\begin{array}{lll}
(s,h) \models E \geq F & \text{iff} & [\![E]\!]s, [\![F]\!]s \in \mathsf{Integers} \text{ and } [\![E]\!]s \geq [\![F]\!]s \\
(s,h) \models E \mapsto F & \text{iff} & \mathsf{dom}(h) = \{[\![E]\!]s\} \text{ and } h([\![E]\!]s) = [\![F]\!]s \\
(s,h) \models \mathsf{emp} & \text{iff} & h = [] \text{ (i.e., } \mathsf{dom}(h) = \emptyset) \\
(s,h) \models P * Q & \text{iff} & \exists h_0 h_1.\ h_0 * h_1 = h,\ (s,h_0) \models P \text{ and } (s,h_1) \models Q \\
(s,h) \models \mathsf{true} & & \text{always} \\
(s,h) \models P \wedge Q & \text{iff} & (s,h) \models P \text{ and } (s,h) \models Q \\
(s,h) \models \neg P & \text{iff} & \text{not } ((s,h) \models P) \\
(s,h) \models \forall x.\, P & \text{iff} & \forall v \in \mathsf{Vals}.\ (s[x \mapsto v], h) \models P)
\end{array}
$$

Semantics of Assertions

$(s,h) \models E \geq F$ iff $[\![E]\!]s, [\![F]\!]s \in$ Integers and $[\![E]\!]s \geq [\![F]\!]s$

$(s,h) \models E \mapsto F$ iff $\mathsf{dom}(h) = \{[\![E]\!]s\}$ and $h([\![E]\!]s) = [\![F]\!]s$

$(s,h) \models \mathsf{emp}$ iff $h = []$ (i.e., $\mathsf{dom}(h) = \emptyset$)

$(s,h) \models P * Q$ iff $\exists h_0 h_1.\ h_0 * h_1 = h,\ (s,h_0) \models P$ and $(s,h_1) \models Q$

$(s,h) \models \mathsf{true}$ always

$(s,h) \models P \wedge Q$ iff $(s,h) \models P$ and $(s,h) \models Q$

$(s,h) \models \neg P$ iff not $((s,h) \models P)$

$(s,h) \models \forall x.\, P$ iff $\forall v \in \mathsf{Vals}.\ (s[x \mapsto v], h) \models P$

Semantics of Assertions

$(s,h) \models E \geq F$	iff	$[\![E]\!]s, [\![F]\!]s \in$ Integers and $[\![E]\!]s \geq [\![F]\!]s$
$(s,h) \models E \mapsto F$	iff	$\mathsf{dom}(h) = \{[\![E]\!]s\}$ and $h([\![E]\!]s) = [\![F]\!]s$
$(s,h) \models \mathsf{emp}$	iff	$h = []$ (i.e., $\mathsf{dom}(h) = \emptyset$)
$(s,h) \models P * Q$	iff	$\exists h_0 h_1.\ h_0 * h_1 = h,\ (s,h_0) \models P$ and $(s,h_1) \models Q$
$(s,h) \models \mathsf{true}$		always
$(s,h) \models P \wedge Q$	iff	$(s,h) \models P$ and $(s,h) \models Q$
$(s,h) \models \neg P$	iff	not $((s,h) \models P)$
$(s,h) \models \forall x.\, P$	iff	$\forall v \in$ Vals. $(s[x \mapsto v], h) \models P$

Semantics of Assertions

$(s,h) \models E \geq F$ iff $[\![E]\!]s, [\![F]\!]s \in$ Integers and $[\![E]\!]s \geq [\![F]\!]s$

$(s,h) \models E \mapsto F$ iff $\mathsf{dom}(h) = \{[\![E]\!]s\}$ and $h([\![E]\!]s) = [\![F]\!]s$

$(s,h) \models \mathsf{emp}$ iff $h = []$ (i.e., $\mathsf{dom}(h) = \emptyset$)

$(s,h) \models P * Q$ iff $\exists h_0 h_1.\ h_0 * h_1 = h,\ (s,h_0) \models P$ and $(s,h_1) \models Q$

$(s,h) \models \mathsf{true}$ always

$(s,h) \models P \wedge Q$ iff $(s,h) \models P$ and $(s,h) \models Q$

$(s,h) \models \neg P$ iff not $((s,h) \models P)$

$(s,h) \models \forall x.\, P$ iff $\forall v \in \mathsf{Vals}.\ (s[x \mapsto v], h) \models P)$

Semantics of Assertions

$(s,h) \models E \geq F$ iff $[\![E]\!]s, [\![F]\!]s \in \mathsf{Integers}$ and $[\![E]\!]s \geq [\![F]\!]s$

$(s,h) \models E \mapsto F$ iff $\mathsf{dom}(h) = \{[\![E]\!]s\}$ and $h([\![E]\!]s) = [\![F]\!]s$

$(s,h) \models \mathsf{emp}$ iff $h = []$ (i.e., $\mathsf{dom}(h) = \emptyset$)

$(s,h) \models P * Q$ iff $\exists h_0 h_1.\ h_0 * h_1 = h,\ (s,h_0) \models P$ and $(s,h_1) \models Q$

$(s,h) \models \mathsf{true}$ always

$(s,h) \models P \wedge Q$ iff $(s,h) \models P$ and $(s,h) \models Q$

$(s,h) \models \neg P$ iff not $((s,h) \models P)$

$(s,h) \models \forall x.\, P$ iff $\forall v \in \mathsf{Vals}.\ (s[x \mapsto v], h) \models P$

Semantics of Assertions

$$
\begin{array}{lll}
(s,h) \models E \geq F & \text{iff} & [\![E]\!]s, [\![F]\!]s \in \mathsf{Integers} \text{ and } [\![E]\!]s \geq [\![F]\!]s \\
(s,h) \models E \mapsto F & \text{iff} & \mathsf{dom}(h) = \{[\![E]\!]s\} \text{ and } h([\![E]\!]s) = [\![F]\!]s \\
(s,h) \models \mathsf{emp} & \text{iff} & h = [] \text{ (i.e., } \mathsf{dom}(h) = \emptyset) \\
(s,h) \models P * Q & \text{iff} & \exists h_0 h_1.\ h_0 * h_1 = h,\ (s,h_0) \models P \text{ and } (s,h_1) \models Q \\
(s,h) \models \mathsf{true} & & \text{always} \\
(s,h) \models P \wedge Q & \text{iff} & (s,h) \models P \text{ and } (s,h) \models Q \\
(s,h) \models \neg P & \text{iff} & \text{not } ((s,h) \models P) \\
(s,h) \models \forall x.\, P & \text{iff} & \forall v \in \mathsf{Vals}.\ (s[x \mapsto v], h) \models P)
\end{array}
$$

Abbreviations

The address E is active:

$$E \mapsto - \triangleq \exists x'.E \mapsto x'$$

where x′ not free in E

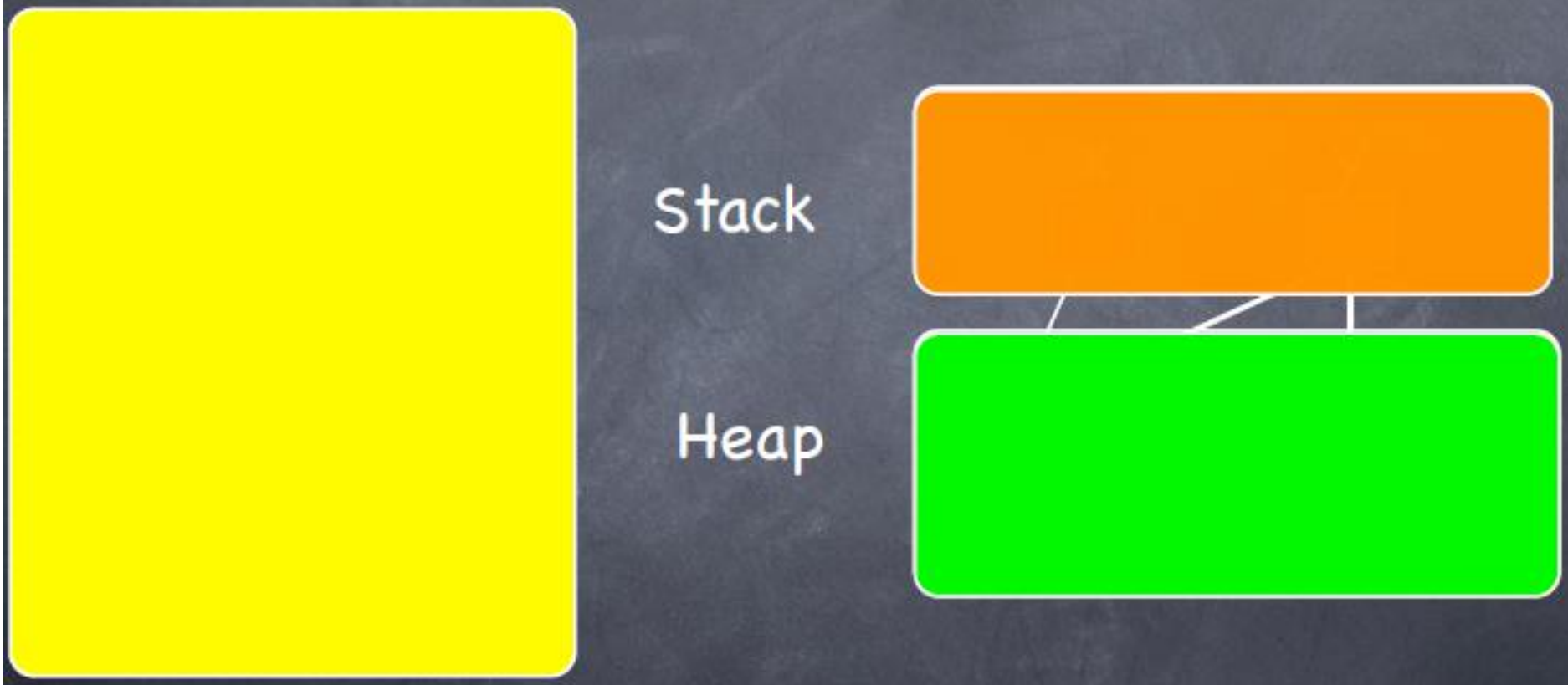
E points to F somewhere in the heap:

$$E \hookrightarrow F \triangleq E \mapsto F * \mathsf{true}$$

E points to a record of several fields:

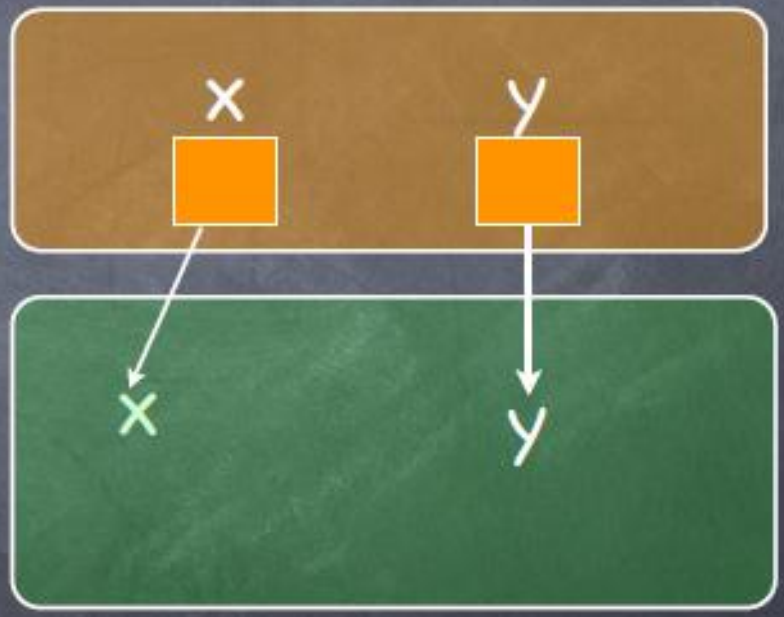
$$E \mapsto E_1, \ldots, E_n \triangleq E \mapsto E_1 * \cdots * E + n - 1 \mapsto E_n$$

Example



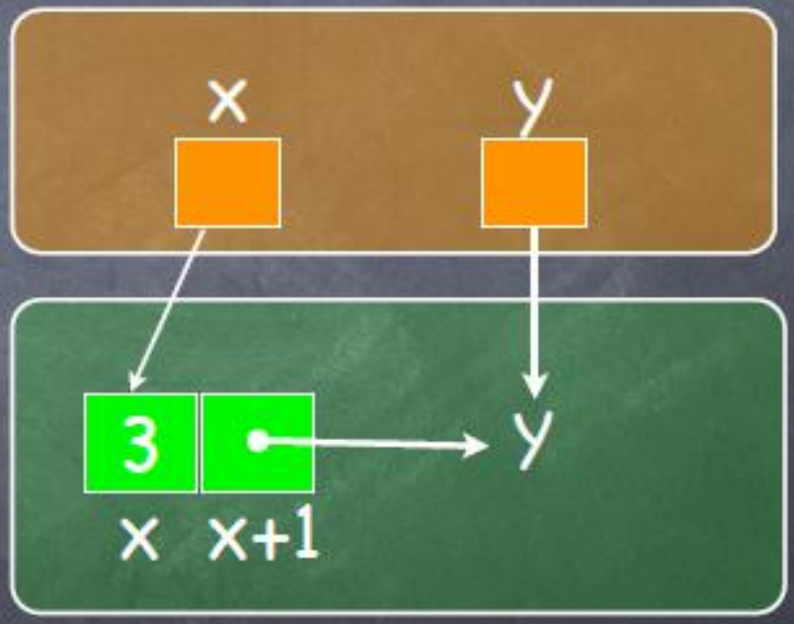
Example

$x \mapsto 3, y$



Example

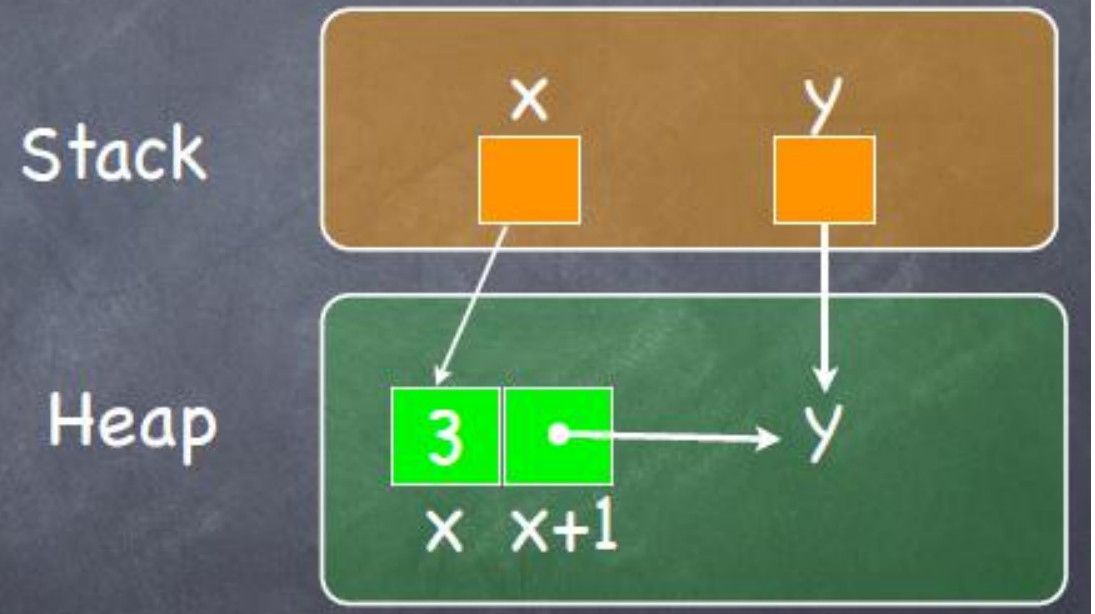
$x \mapsto 3, y$



Example

$x \mapsto 3, y$

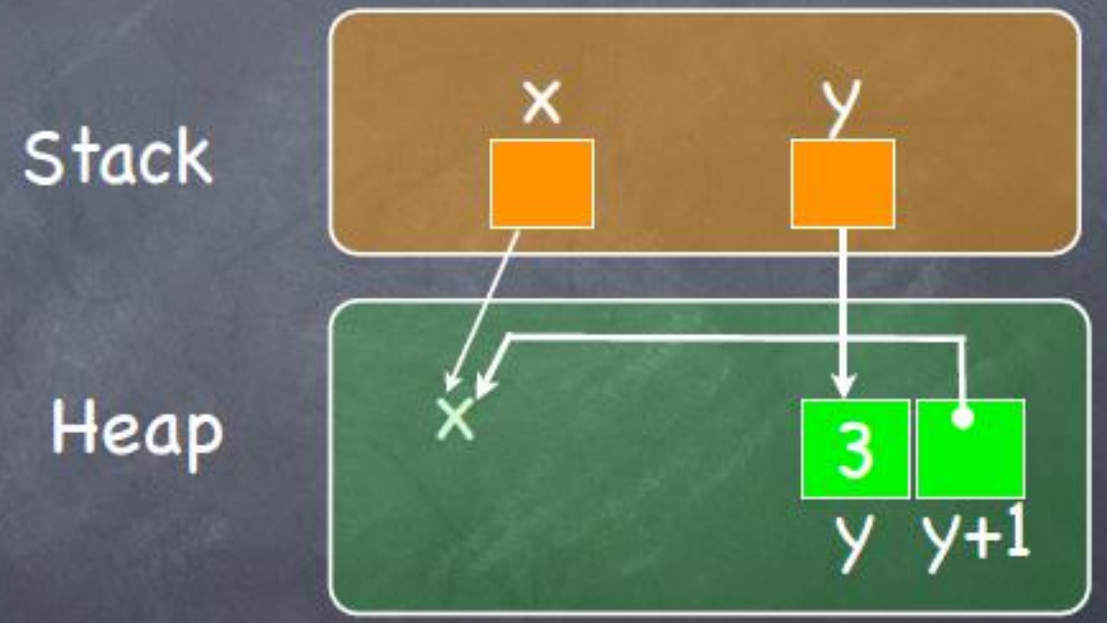
$y \mapsto 3, x$



Example

$x \mapsto 3, y$

$y \mapsto 3, x$

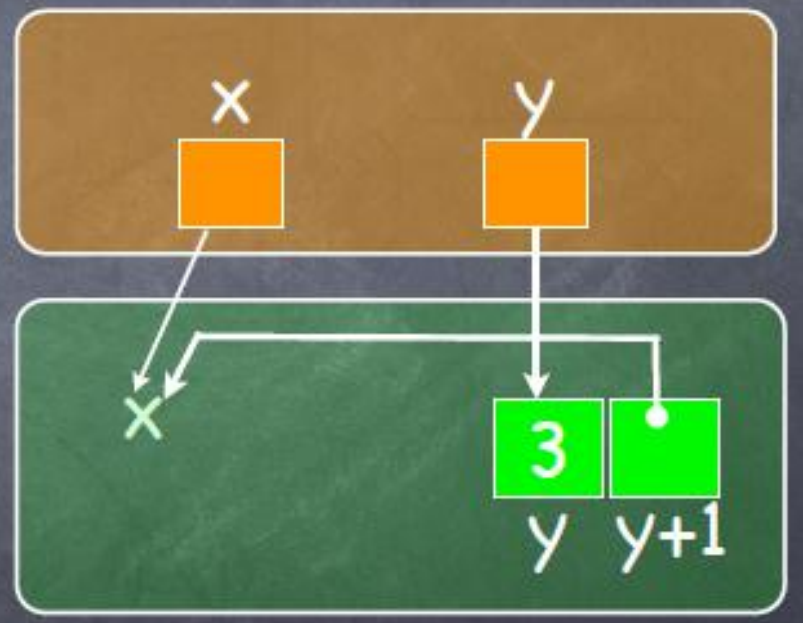


Example

$x \mapsto 3, y$

$y \mapsto 3, x$

$x \mapsto 3, y * y \mapsto 3, x$

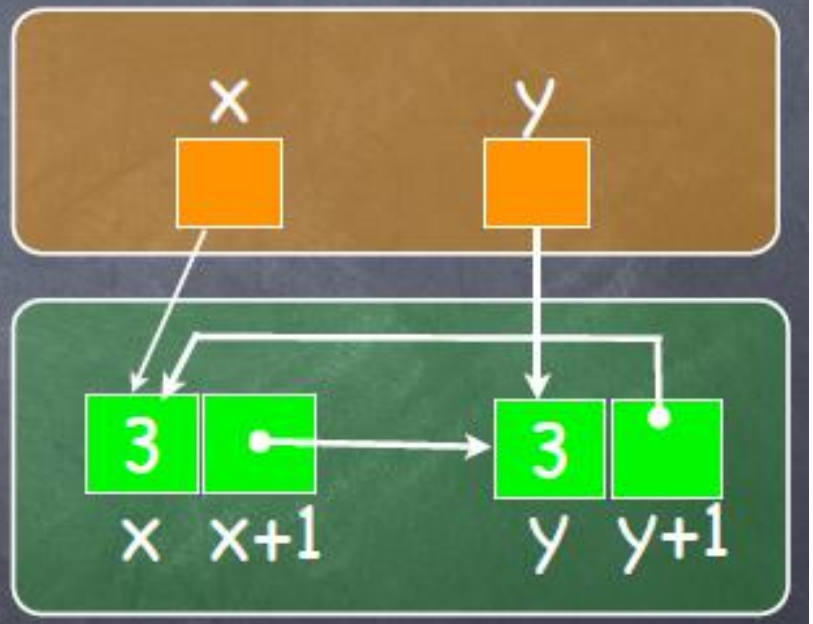


Example

$x \mapsto 3, y$

$y \mapsto 3, x$

$x \mapsto 3, y * y \mapsto 3, x$



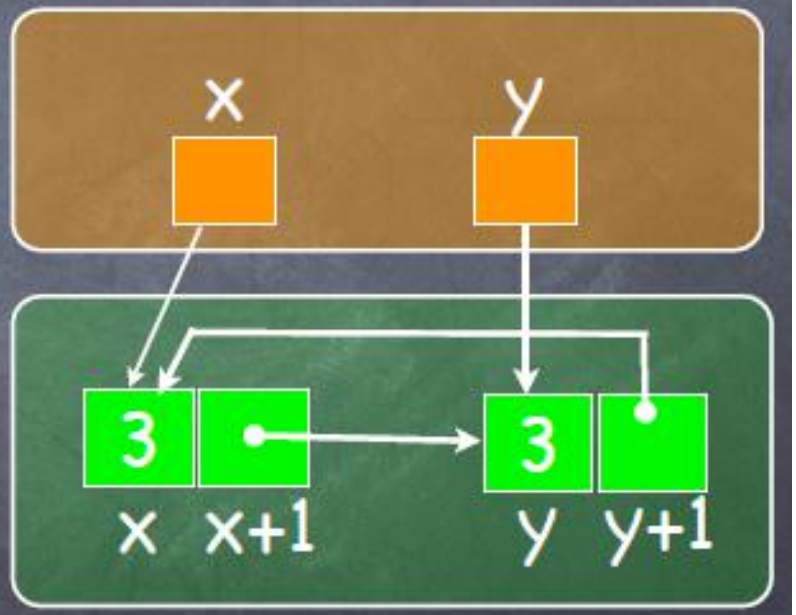
Example

$x \mapsto 3, y$

$y \mapsto 3, x$

$x \mapsto 3, y * y \mapsto 3, x$

$x \mapsto 3, y \wedge y \mapsto 3, x$



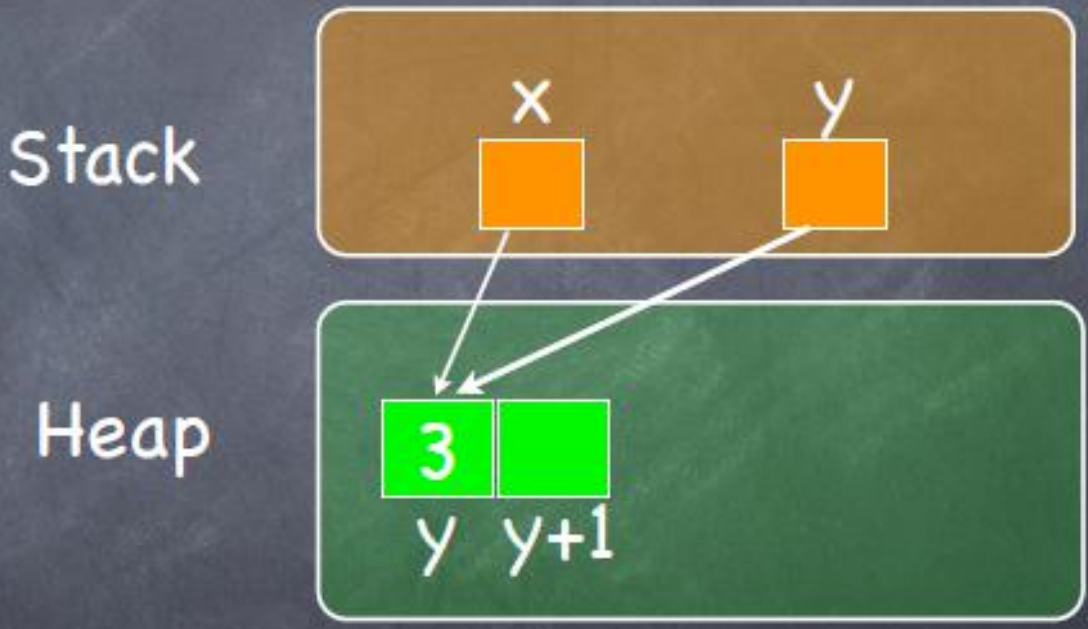
Example

$x \mapsto 3, y$

$y \mapsto 3, x$

$x \mapsto 3, y * y \mapsto 3, x$

$x \mapsto 3, y \wedge y \mapsto 3, x$

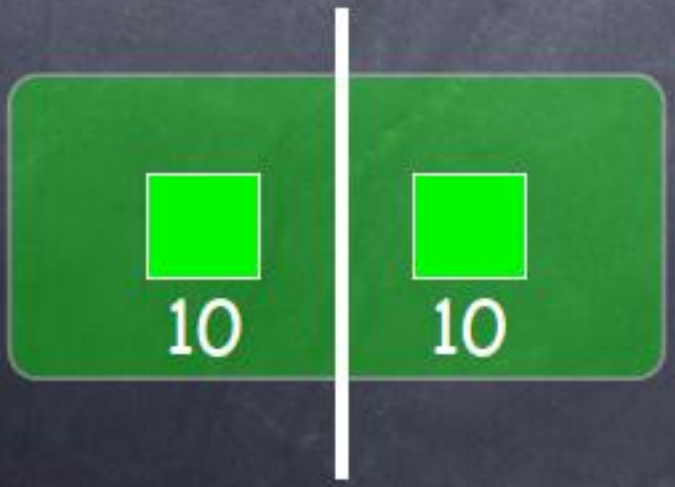


An inconsistency

- What's wrong with the following formula?
  - 10|–>3 * 10|–>3

An inconsistency

- What's wrong with the following formula?
  - 10|->3 * 10|->3



Try to be in two places at the same time

Exercise

what is h such that s,h|= p

h1={(s(x),1)}

h2={(s(y),2)}

with s(x)!=s(y)

Exercise

what is h such that s,h|= p

x|->1

h=h1

h1={(s(x),1)}

h2={(s(y),2)}

with s(x)!=s(y)

Exercise

what is h such that s,h|= p

x|->1 h=h1

y|->2 h=h2

h1={(s(x),1)}

h2={(s(y),2)}

with s(x)!=s(y)

Exercise

what is h such that s,h|= p

h1={(s(x),1)}

h2={(s(y),2)}

with s(x)!=s(y)

x|->1 h=h1

y|->2 h=h2

x|->1 * y|->2 h=h1 * h2

Exercise

what is h such that s,h|= p

h1={(s(x),1)}

h2={(s(y),2)}

with s(x)!=s(y)

x|->1	h=h1
y|->2	h=h2
x|->1 * y|->2	h=h1 * h2
x|->1 * true	h1 contained in h

Exercise

what is h such that s,h|= p

h1={(s(x),1)}

h2={(s(y),2)}

with s(x)!=s(y)

x|->1	h=h1
y|->2	h=h2
x|->1 * y|->2	h=h1 * h2
x|->1 * true	h1 contained in h
x|->1 * y|->2 * (x|->1 \/ y|->2)	Homework!

Validity

- P is valid if, for all s,h, s,h|=P
- Examples:
  - E|->3 => E>0 Valid!
  - E|-> - * E|-> -
  - E|-> - * F |-> - => E != F
  - E |-> 3 /\ F |-> 3 => E=F
  - E|->3 * F |->3 => E|->3 /\ F |->3

Validity

- P is valid if, for all s,h, s,h|=P
- Examples:
  - E|->3 => E>0 Valid!
  - E|-> - * E|-> - Invalid!
  - E|-> - * F |-> - => E != F
  - E |-> 3 /\ F |-> 3 => E=F
  - E|->3 * F |->3 => E|->3 /\ F |->3

Validity

- P is valid if, for all s,h, s,h|=P
- Examples:
  - E|->3 => E>0 Valid!
  - E|-> - * E|-> - Invalid!
  - E|-> - * F |-> - => E != F Valid!
  - E |-> 3 /\ F |-> 3 => E=F
  - E|->3 * F |->3 => E|->3 /\ F |->3

Validity

- P is valid if, for all s,h, s,h|=P
- Examples:
  - E|->3 => E>0 Valid!
  - E|-> - * E|-> - Invalid!
  - E|-> - * F |-> - => E != F Valid!
  - E |-> 3 /\ F |-> 3 => E=F Valid!
  - E|->3 * F |->3 => E|->3 /\ F |->3

Validity

- P is valid if, for all s,h, s,h|=P
- Examples:
  - E|->3 => E>0 Valid!
  - E|-> - * E|-> - Invalid!
  - E|-> - * F |-> - => E != F Valid!
  - E |-> 3 /\ F |-> 3 => E=F Valid!
  - E|->3 * F |->3 => E|->3 /\ F |->3 Invalid!

Some Laws and inference rules

$$p_1 * p_2 \iff p_2 * p_1$$

$$(p_1 * p_2) * p3 \iff p_1 * (p_2 * p_3)$$

$$p * \mathsf{emp} \iff p$$

$$(p_1 \vee p_2) * q \iff (p_1 * q) \vee (p_2 * q)$$

$$(\exists x.p_1) * p_2 \iff \exists x.(p_1 * p_2) \quad \text{when } x \text{ not in } p_2$$

$$(\forall x.p_1) * p_2 \iff \forall x.(p_1 * p_2) \quad \text{when } x \text{ not in } p_2$$

$$\frac{p_1 \implies p_2 \qquad q_1 \implies q_2}{p_1 * q_1 \implies p_2 * q_2}$$ Monotonicity

Some Laws and inference rules

...but

$$(p_1 \wedge p_2) * q \implies (p_1 * q) \wedge (p_2 * q)$$

Exercise: prove that the other direction does not hold

$$(\forall x.p_1) * p_2 \iff \forall x.(p_1 * p_2) \quad \text{when } x \text{ not in } p_2$$

$$\frac{p_1 \implies p_2 \qquad q_1 \implies q_2}{p_1 * q_1 \implies p_2 * q_2}$$ Monotonicity

Substructural logic

- Separation logic is a substructural logic:

No Contraction $A \not\vdash A * A$

No Weakening $A * B \not\vdash A$

Examples:

$10 \mapsto 3 \not\vdash 10 \mapsto 3 * 10 \mapsto 3$

$10 \mapsto 3 * 42 \mapsto 7 \not\vdash 42 \mapsto 7$

Lists

A non circular list can be defined with the following inductive predicate:

$$\text{list } [] \ i = \text{emp} \wedge i = \text{nil}$$
$$\text{list } (s::S) \ i = \exists j.\ i \mapsto s,j * \text{list } S \ j$$

List segment

Possibly empty list segment

lseg(x,y) = (emp /\ x=y) OR
exists j. x|->j * lseg(j,y)

Non-empty non-circular list segment

lseg(x,y) = x!=y /\
((x|->y) OR exists j. x|->j * lseg(j,y))

Trees

A tree can be defined with this inductive definition:

$$\text{tree } [] \; i = \text{emp} \wedge i = \text{nil}$$

$$\text{tree } (t1, a, t2) \; i = \exists j, k. \; i \mapsto j, a, k * (\text{tree } t1 \; j) * (\text{tree } t2 \; k)$$