

High Level Program Reasoning

Wessex Theory Seminar - Feb 2010

Mark Wheelhouse
Imperial College London

Introduction

Program Verification:

Introduction

Program Verification:

- Separation Logic 

C Programs

Device Drivers

Introduction

Program Verification:

- Separation Logic 

C Programs

Device Drivers

- Context Logic

Tree Update

DOM Specification



Introduction

Program Verification:

- Separation Logic

C Programs

Device Drivers



- Context Logic

Tree Update

DOM Specification



- Segment Logic

Fine-grained Update

Concurrent Update



Introduction

Program Verification:

- Separation Logic

C Programs

Device Drivers



—
Low
Level

- Context Logic

Tree Update

DOM Specification



- Segment Logic

Fine-grained Update

Concurrent Update



Introduction

Program Verification:

- Separation Logic

C Programs

Device Drivers



—
Low
Level

- Context Logic

Tree Update

DOM Specification



—
High
Level



- Segment Logic

Fine-grained Update

Concurrent Update

Programming Language

skip

x := Exp

C;C

if (B) then { C } else { C }

while (B) do { C }

C_{Basic}

Local Hoare Reasoning

Partial Correctness: $\{ P \} \subset \{ Q \}$

Local Hoare Reasoning

Partial Correctness: $\{ P \} \ C \ \{ Q \}$

Precondition

Local Hoare Reasoning

Partial Correctness: $\{ P \} \subset \{ Q \}$

The diagram illustrates the concept of partial correctness using set notation. It shows two sets, {P} and {Q}, with a subset symbol (\subset) indicating that the set {P} is a subset of the set {Q}. Below the set {P}, there is a green arrow pointing upwards, labeled "Precondition". Below the set {Q}, there is another green arrow pointing upwards, labeled "Program". This visual representation emphasizes that the precondition {P} must be satisfied before the program executes, and the program's behavior is guaranteed to result in a state that satisfies the postcondition {Q}.

Local Hoare Reasoning

Partial Correctness:

$$\text{PSS: } \{ P \} \quad C \quad \{ Q \}$$

The diagram illustrates the components of a program specification. It features three labels at the bottom: "Precondition" on the left, "Program" in the center, and "Postcondition" on the right. Three green arrows point upwards from these labels to the corresponding parts of the formula above: the left arrow points to the set $\{ P \}$, the middle arrow points to the symbol C , and the right arrow points to the set $\{ Q \}$.

Separation Logic

heap $h : \mathbb{N}^+ \xrightarrow{\text{fin}} \text{Val}$

Separation Logic

heap $h : \mathbb{N}^+ \xrightarrow{\text{fin}} \text{Val}$

emp

empty heap

$x \mapsto y$

heap of exactly one cell

$P * Q$

separating conjunction
(disjoint union)

Separation Logic

heap $h : \mathbb{N}^+ \xrightarrow{\text{fin}} \text{Val}$

emp

empty heap

$x \mapsto y$

heap of exactly one cell

$P * Q$

separating conjunction
(disjoint union)

$x \mapsto y * y \mapsto z * w \mapsto \emptyset$

Low-Level Reasoning

$\{ x \mapsto - \}$

dispose(x)

$\{ \text{emp} \}$

$\{ x \mapsto v \}$

$[x] := E$

$\{ x \mapsto E[v/x] \}$

Low-Level Reasoning

$$\{ x \mapsto - \}$$

dispose(x)

$$\{ \text{emp} \}$$
$$\{ x \mapsto v \}$$

[x] := E

$$\{ x \mapsto E[v/x] \}$$

Small Axioms

Separation Frame Rule

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

Separation Frame Rule

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

h

Separation Frame Rule

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

R * P

Separation Frame Rule

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



Separation Frame Rule

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



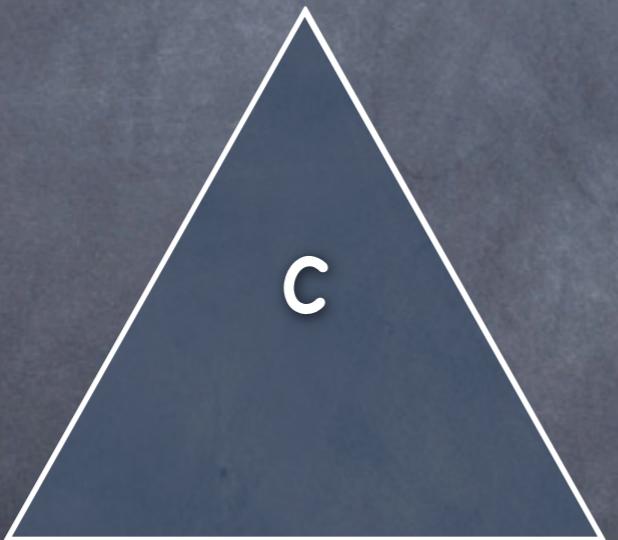
Separation Frame Rule

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

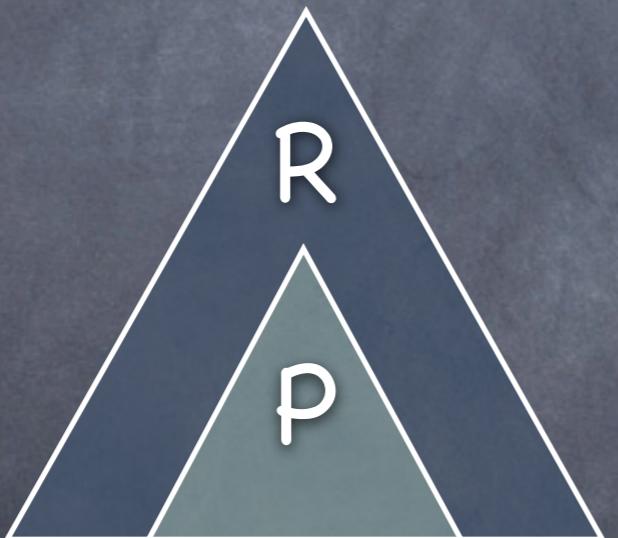


Context Logic in a Nutshell

Context Logic in a Nutshell



Context Logic in a Nutshell



Context Logic in a Nutshell



Context Logic in a Nutshell



Context Logic in a Nutshell



Context Logic in a Nutshell



Context Logic in a Nutshell



Concurrent Programs

⋮

$C \parallel C$

res r in { C }

with r when B do { C }

Conditional Critical
Region!!!

Disjoint Concurrency at the Low-Level

$$\frac{\{ P_1 \} C_1 \{ Q_1 \} \quad \{ P_2 \} C_2 \{ Q_2 \}}{\{ P_1 * P_2 \} C_1 || C_2 \{ Q_1 * Q_2 \}} \text{ PAR}$$

Disjoint Concurrency Example

[x] := 5 || dispose(y)

Disjoint Concurrency Example

$$\{ \ x \mapsto - * y \mapsto - \ }$$
$$[x] := 5 \quad \parallel \quad \text{dispose}(y)$$

Disjoint Concurrency Example

$$\begin{array}{c} \{ x \mapsto - * y \mapsto - \} \\[10pt] \{ x \mapsto - \} \quad \parallel \quad \{ y \mapsto - \} \\ [x] := 5 \qquad \qquad \qquad \text{dispose}(y) \end{array}$$

Disjoint Concurrency Example

$$\begin{array}{c} \{ x \mapsto - * y \mapsto - \} \\ \\ \{ x \mapsto - \} \quad \parallel \quad \{ y \mapsto - \} \\ [x] := 5 \quad \parallel \quad \text{dispose}(y) \\ \{ x \mapsto 5 \} \quad \parallel \quad \{ \text{emp} \} \end{array}$$

Disjoint Concurrency Example

$$\begin{array}{c} \{ x \mapsto - * y \mapsto - \} \\ \\ \{ x \mapsto - \} \parallel \{ y \mapsto - \} \\ [x] := 5 \qquad \text{dispose}(y) \\ \{ x \mapsto 5 \} \parallel \{ \text{emp} \} \\ \\ \{ x \mapsto 5 * \text{emp} \} \end{array}$$

Disjoint Concurrency Example

$$\begin{array}{c} \{ x \mapsto - * y \mapsto - \} \\[10pt] \{ x \mapsto - \} \parallel \{ y \mapsto - \} \\[10pt] [x] := 5 \qquad \text{dispose}(y) \\[10pt] \{ x \mapsto 5 \} \parallel \{ \text{emp} \} \\[10pt] \{ x \mapsto 5 * \text{emp} \} \\[10pt] \{ x \mapsto 5 \} \end{array}$$

Complex Concurrency at the Low-Level

$$\frac{\{ P \} \subset \{ Q \}}{\{ RI_r * P \} \text{ res } r \text{ in } \{ C \} \{ RI_r * Q \}} \text{ RES}$$

$$\frac{\{ RI_r * P \wedge B \} \subset \{ RI_r * Q \}}{\{ P \} \text{ with } r \text{ when } B \text{ do } \{ C \} \{ Q \}} \text{ CCR}$$

Complex Concurrency Example

```
with r when x=0 do { || with r when x=1 do {  
    c := cons(); x=1  
    }  
    }  
    }  
    }
```

Complex Concurrency Example

$$RI_r = (x=0 \wedge emp) \vee (x=1 \wedge c \mapsto -)$$

with r when x=0 do {

with r when x=1 do {

```
c := cons(); x=1
```

dispose(c); x=0

{}

Complex Concurrency Example

$$RI_r = (x=0 \wedge emp) \vee (x=1 \wedge c \mapsto -)$$

{ emp }

with r when $x=0$ do {

$c := \text{cons}(); x=1$

}

with r when $x=1$ do {

$\text{dispose}(c); x=0$

}

Complex Concurrency Example

$$RI_r = (x=0 \wedge emp) \vee (x=1 \wedge c \mapsto -)$$

{ emp }

with r when $x=0$ do {

{ $RI_r \wedge x=0$ }

$c := \text{cons}(); x=1$

}

with r when $x=1$ do {

$\text{dispose}(c); x=0$

}

Complex Concurrency Example

$$RI_r = (x=0 \wedge emp) \vee (x=1 \wedge c \mapsto -)$$

```
{ emp }  
with r when x=0 do {  
  { RI_r ∧ x=0 }  
  c := cons(); x=1  
  { RI_r ∧ x=1 }  
}  
|||  
with r when x=1 do {  
  dispose(c); x=0  
}  
}
```

Complex Concurrency Example

$$RI_r = (x=0 \wedge emp) \vee (x=1 \wedge c \mapsto -)$$

```
{ emp }  
with r when x=0 do {  
  { RI_r ∧ x=0 }  
  c := cons(); x=1  
  { RI_r ∧ x=1 }  
}  
{ emp }  
|||  
with r when x=1 do {  
  dispose(c); x=0  
}  
}
```

Complex Concurrency Example

$$RI_r = (x=0 \wedge emp) \vee (x=1 \wedge c \mapsto -)$$

```
{ emp }                                { emp }
with r when x=0 do {                   ||| with r when x=1 do {
  { RI_r \wedge x=0 }
  c := cons(); x=1
  { RI_r \wedge x=1 }
}
{ emp }                                }
```

{ emp }

Problems at the High-Level

$\{ n[t] \}$	\parallel	$\{ m[t'] \}$
deleteTree(n)		deleteTree(m)
$\{ \emptyset \}$		$\{ \emptyset \}$

Problems at the High-Level

{ n[t] -?- m[t'] }	
{ n[t] }	{ m[t'] }
deleteTree(n)	
{ Ø }	{ Ø }

Problems at the High-Level

{ n[t] -?- m[t'] }	
{ n[t] }	{ m[t'] }
deleteTree(n)	
{ Ø }	{ Ø }

⊗ – but what if not siblings?

Problems at the High-Level

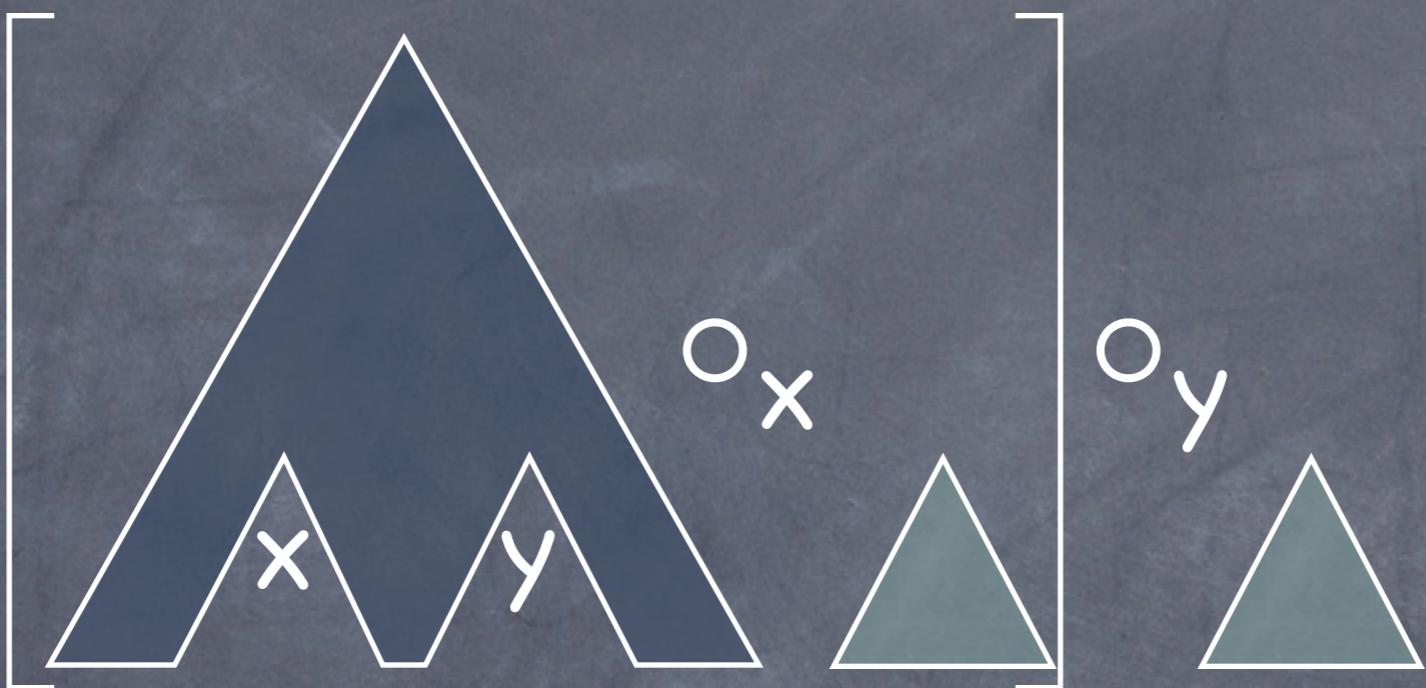
{ n[t] -?- m[t'] }	
{ n[t] }	{ m[t'] }
deleteTree(n)	
{ Ø }	{ Ø }

- ⊗ - but what if not siblings?
- ο_x - but neither is a context.

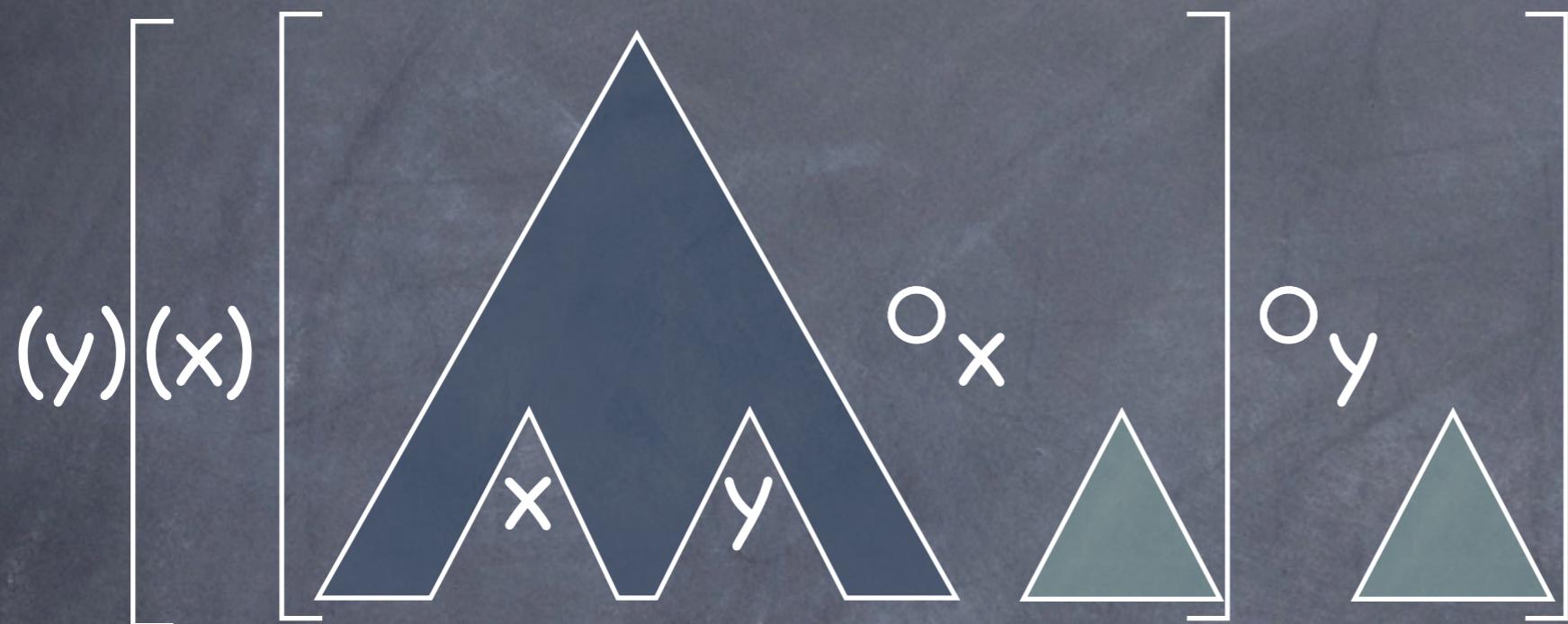
Segment Idea



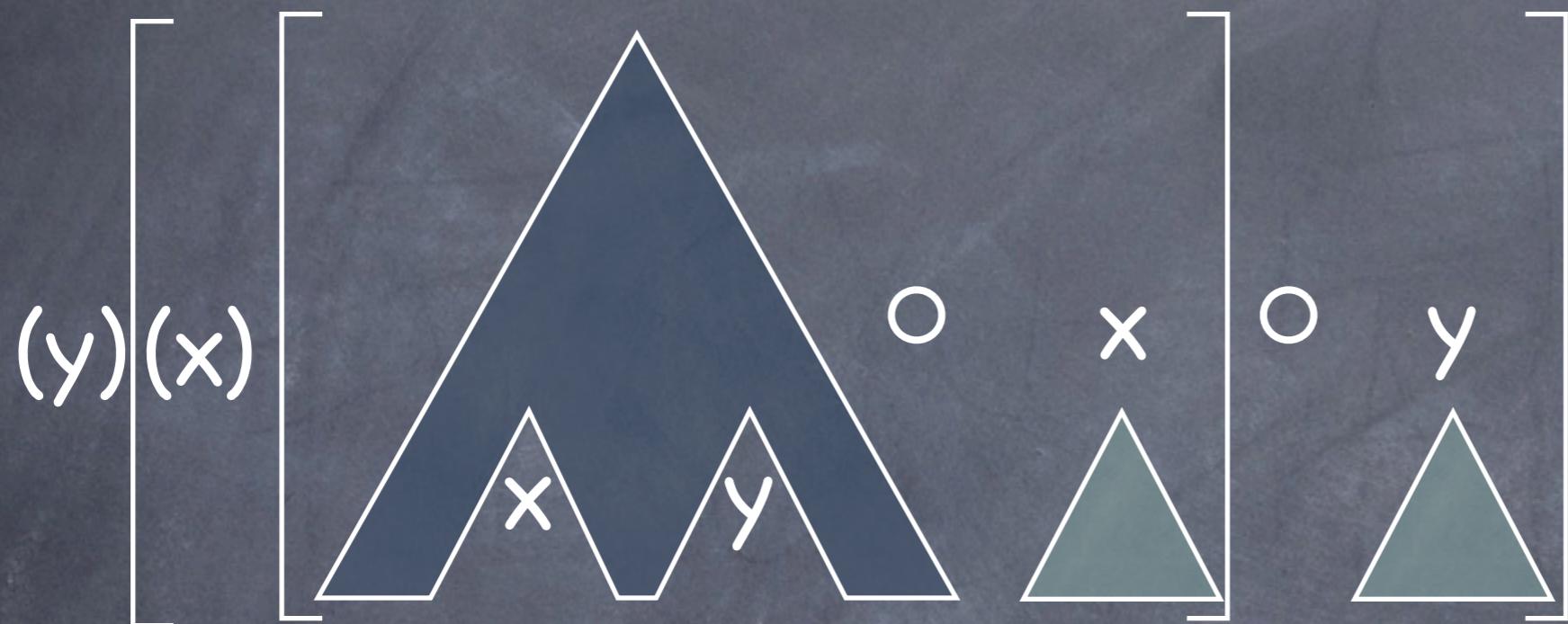
Segment Idea



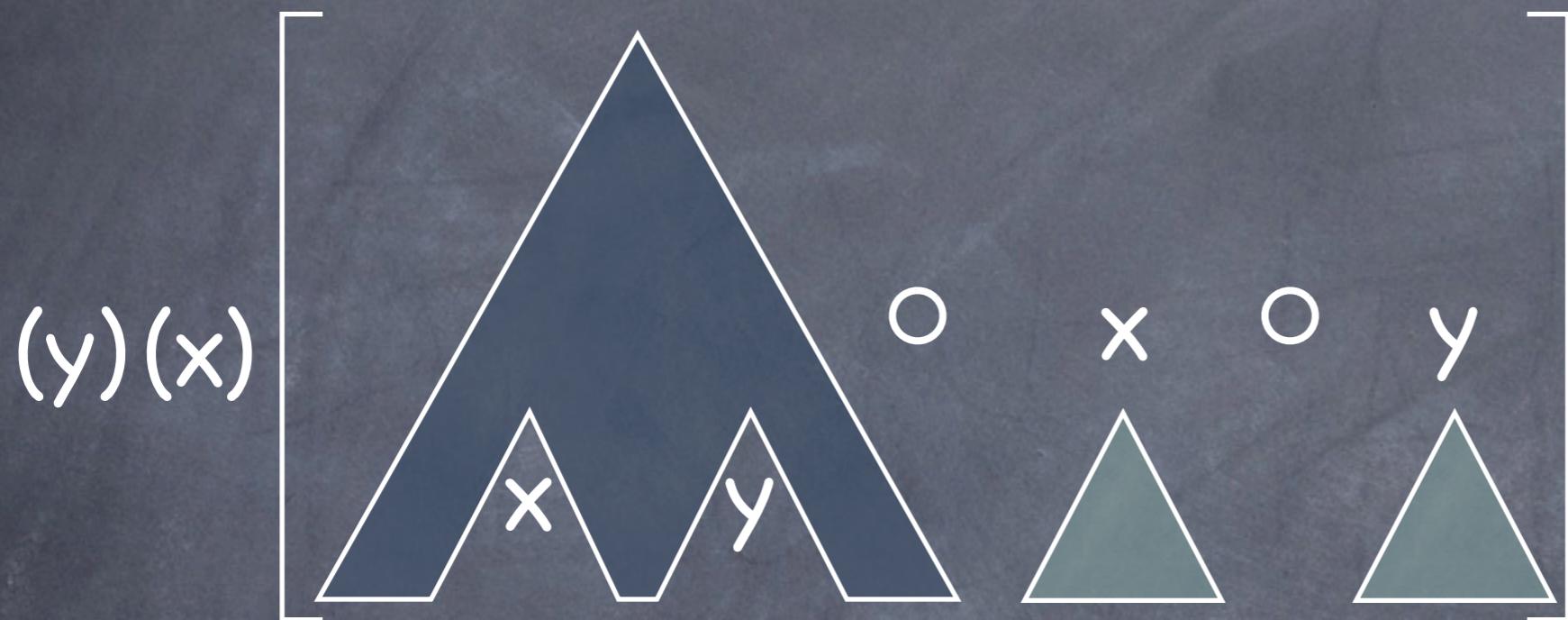
Segment Idea



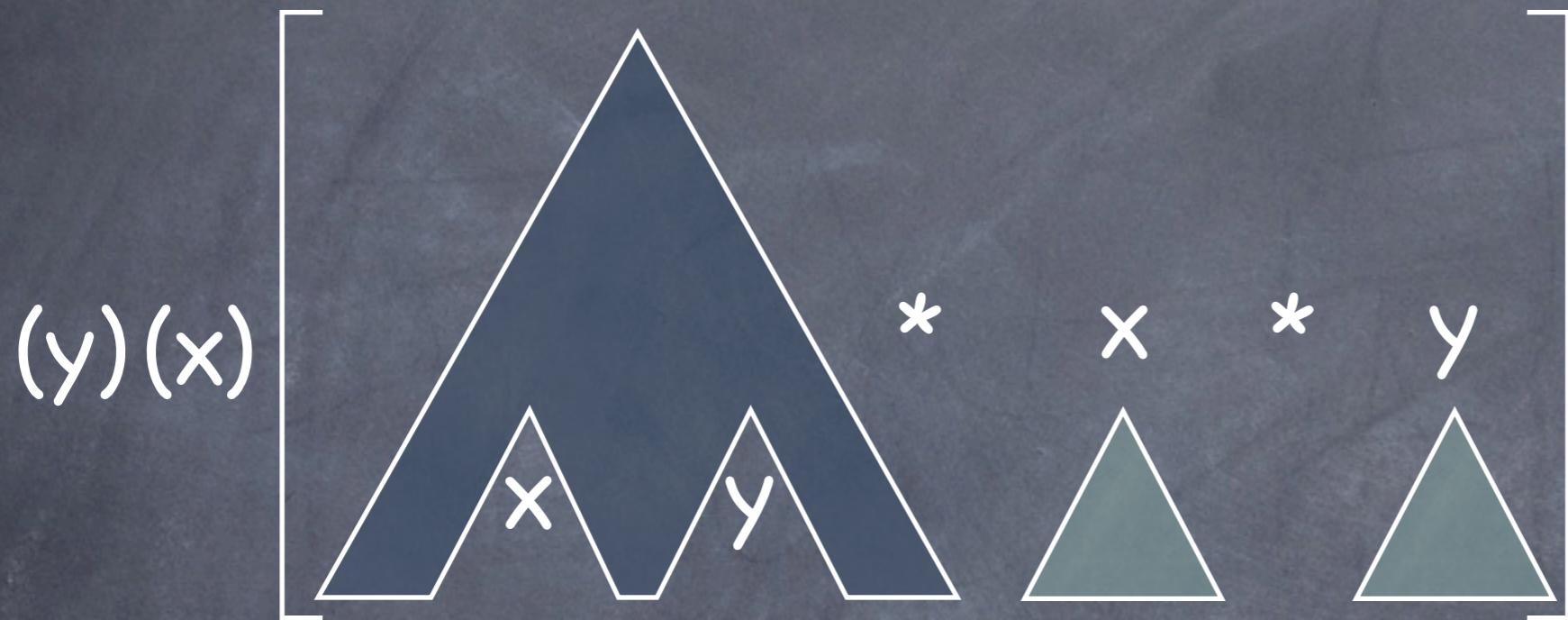
Segment Idea



Segment Idea



Segment Idea

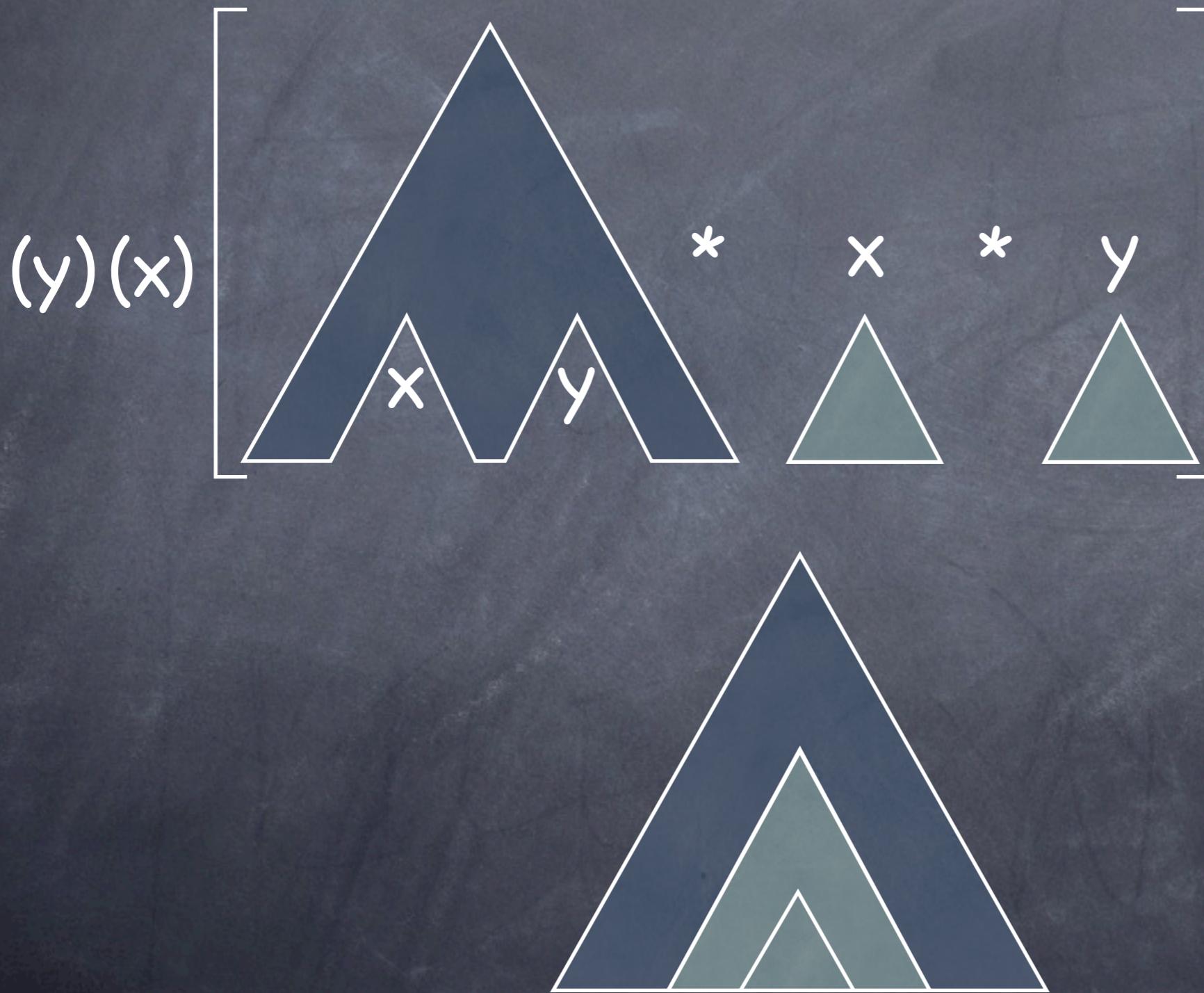


Segment Idea

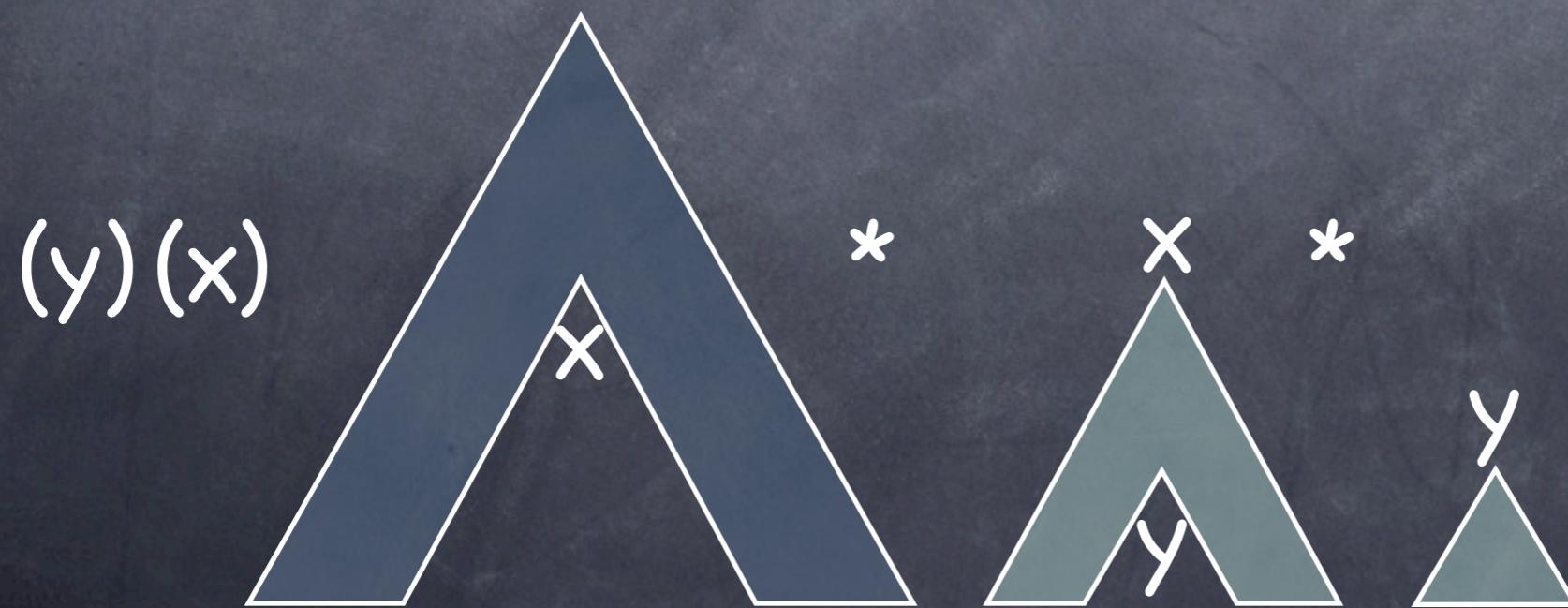
$$(y)(x) \left[\begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \end{array} \right] * \begin{array}{c} x \\ \text{---} \\ \text{---} \end{array} * \begin{array}{c} y \\ \text{---} \\ \text{---} \end{array} \right] = \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \end{array}$$

The diagram illustrates the concept of segments in a mathematical context. It shows a large triangle being partitioned into smaller triangles labeled x, y, and z. This partitioning is followed by two multiplication operators (*). To the right of the first operator is a single triangle labeled x, and to the right of the second operator is a single triangle labeled y. The entire sequence of partitions and operators is enclosed in brackets on both sides, indicating a product of segments. This is followed by an equals sign and a final large triangle.

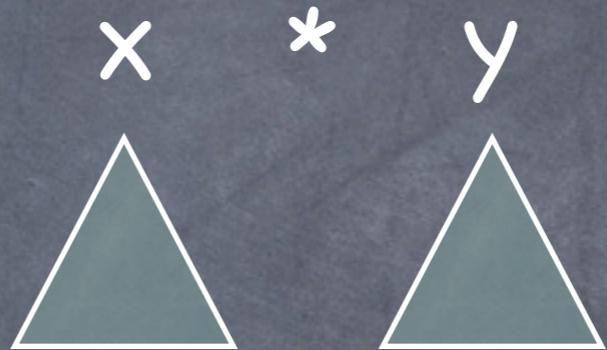
Segment Idea



Segment Idea



Segment Idea



Tree Segments

tree context $c ::= \emptyset \mid x \mid n[c] \mid c \otimes c$

tree segment $s ::= \emptyset_s \mid x \leftarrow c \mid s + s \mid (x)(s)$

Unique node identifiers n

Unique free hole addresses $x \leftarrow$

Unique free hole labels x

+ associative & commutative with unit \emptyset_s

\otimes associative with unit \emptyset

& no cycles!

Adjoints important for
Weakest Preconditions

Important Formulae

Structural formulae +

$P * Q$ separating conjunction

$P \multimap Q$ separation right adjoint

$\alpha @ P$ revelation

$\alpha -@ Q$ revelation right adjoint

$\text{H}\alpha.P$ fresh label

$\text{H}\alpha.P$ derived label hiding

Fine-grained High-Level Reasoning

$\{ \alpha \leftarrow n[t] \}$

deleteTree(n)

$\{ \alpha \leftarrow \emptyset \}$

$\{ \alpha \leftarrow n[\gamma] * \beta \leftarrow m[t] \}$

append(n, m)

$\{ \alpha \leftarrow n[\gamma \odot m[t]] * \beta \leftarrow \emptyset \}$

Fine-grained High-Level Reasoning

$\{ \alpha \leftarrow n[t] \}$

deleteTree(n)

$\{ \alpha \leftarrow \emptyset \}$

$\{ \alpha \leftarrow n[\gamma] * \beta \leftarrow m[t] \}$

append(n, m)

$\{ \alpha \leftarrow n[\gamma \odot m[t]] * \beta \leftarrow \emptyset \}$

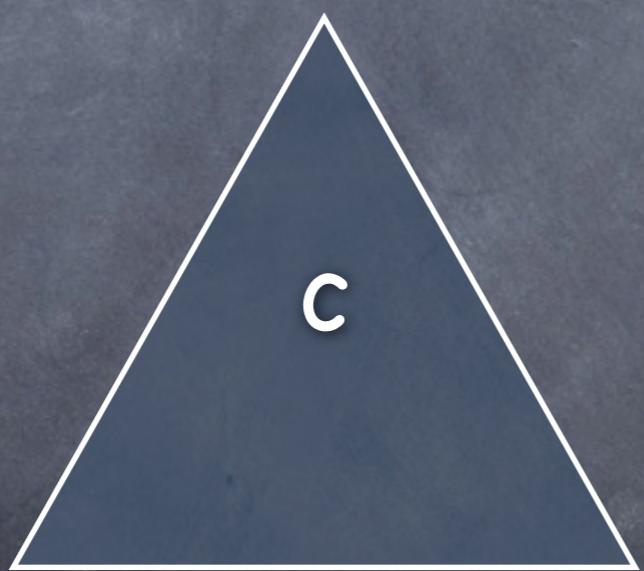
Small Axioms

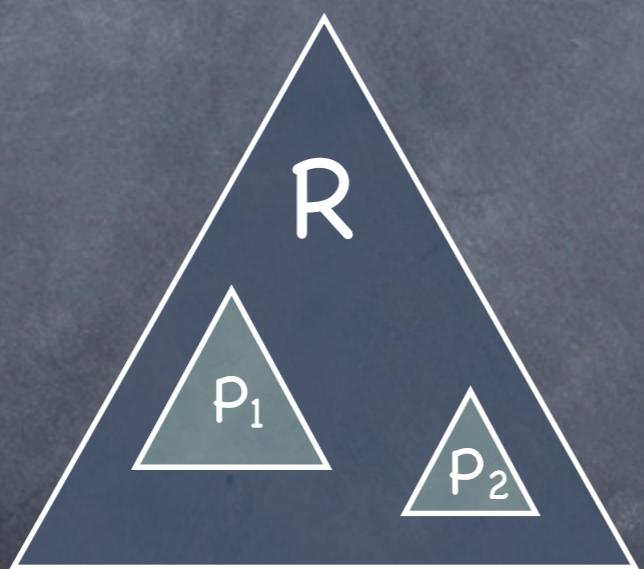
Hoare Rules

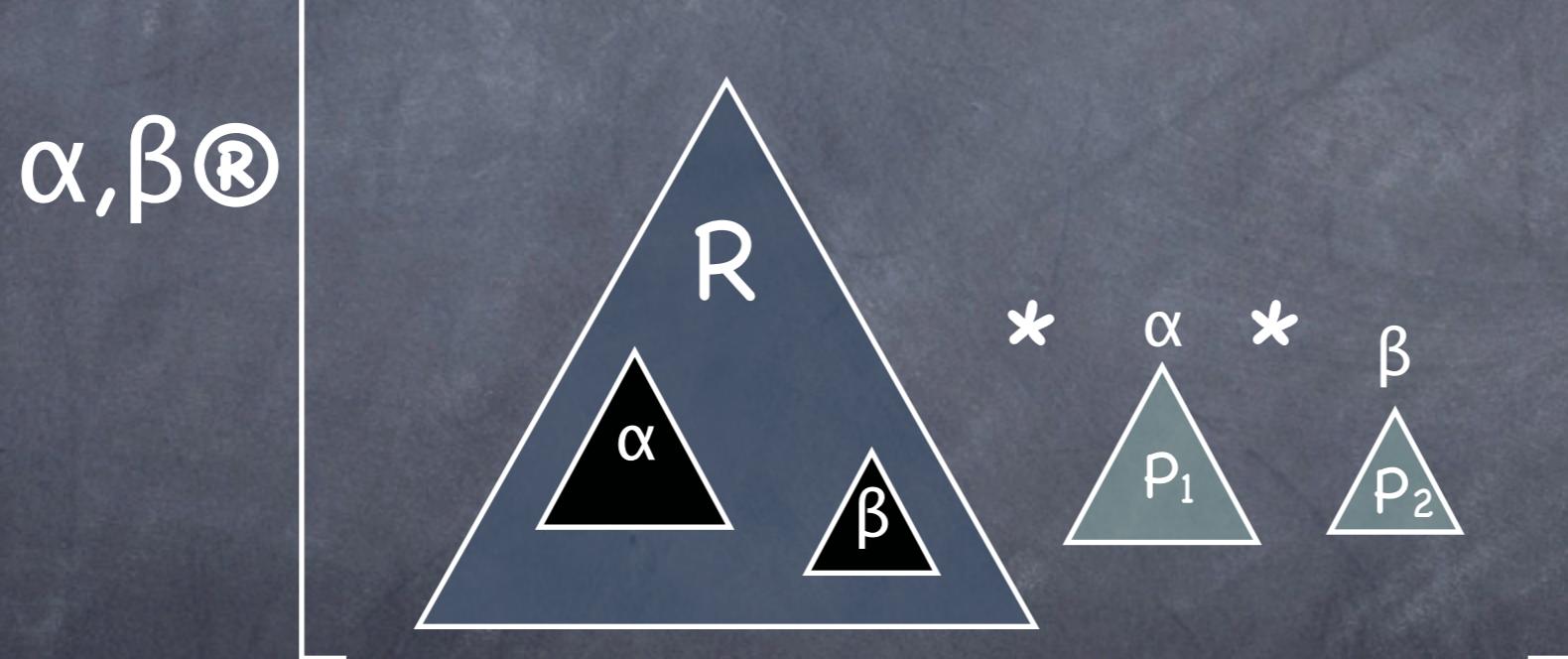
$$\frac{\{P\} \vdash \{Q\}}{\{R * P\} \vdash \{R * Q\}}$$

$$\frac{\{P\} \vdash \{Q\}}{\{\text{N}\alpha.P\} \vdash \{\text{N}\alpha.Q\}}$$

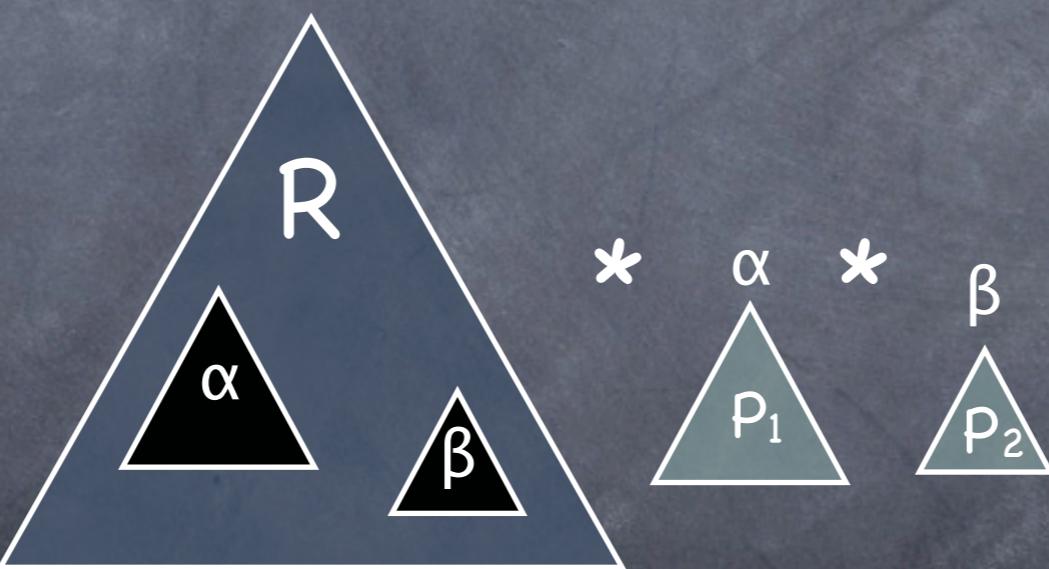
$$\frac{\{P\} \vdash \{Q\}}{\{\alpha @ R P\} \vdash \{\alpha @ R Q\}}$$







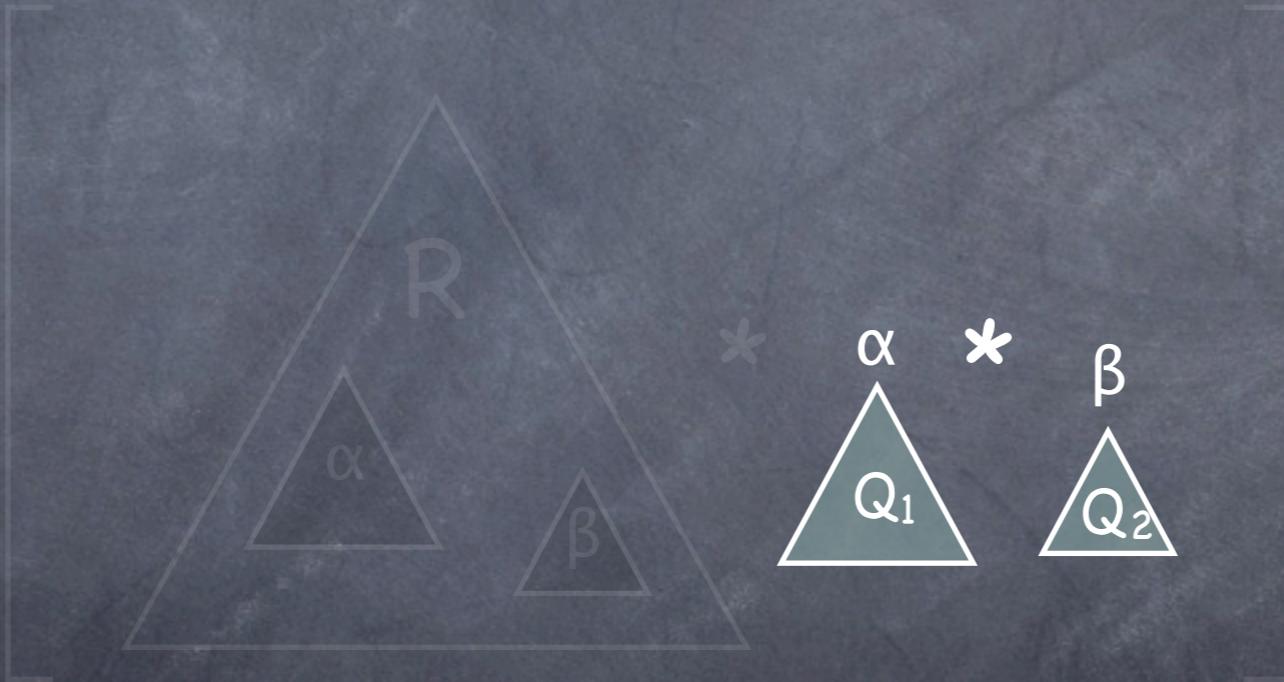
$\alpha, \beta \circledR$



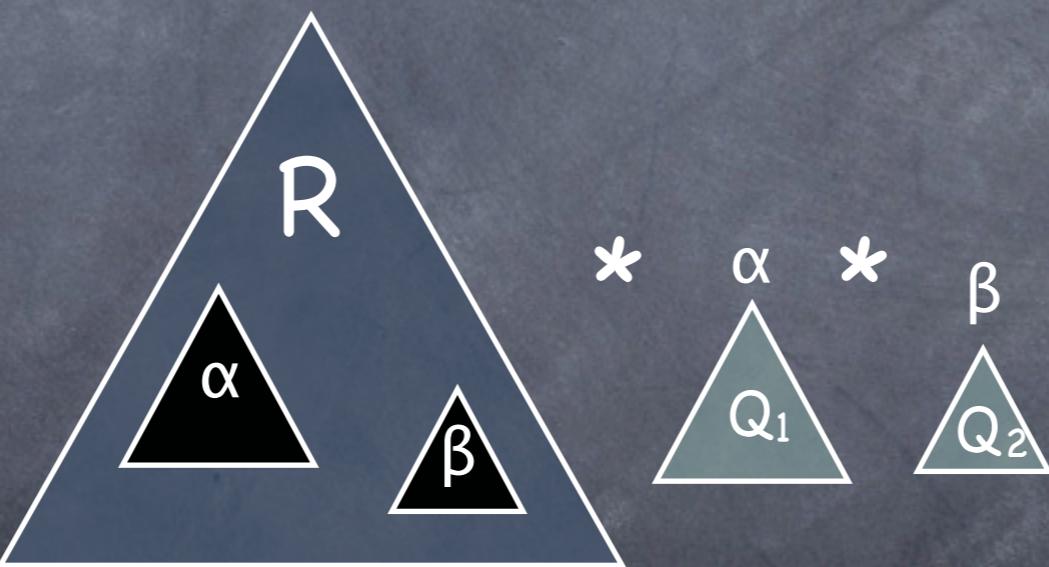
$\alpha, \beta \circledR$



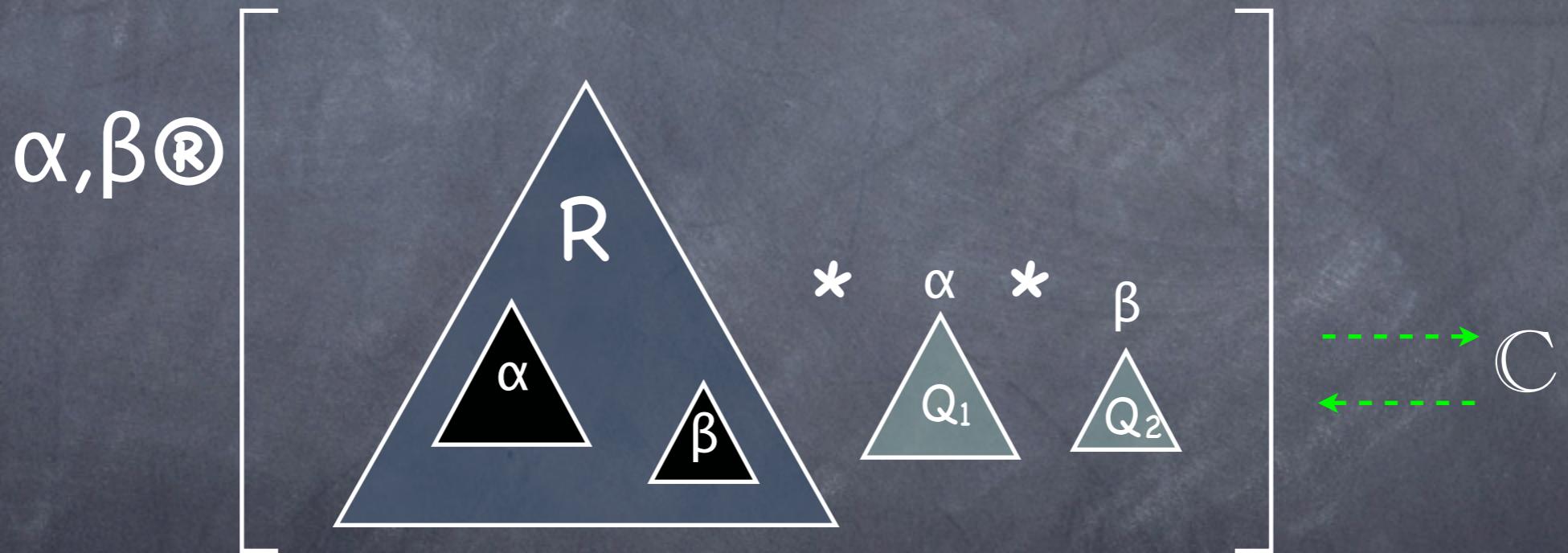
$\alpha, \beta \circledR$



$\alpha, \beta \circledR$



$\xleftarrow{\quad} \xrightarrow{\quad} C$





Back to Disjoint Concurrency

$$\frac{\{ P_1 \} C_1 \{ Q_1 \} \quad \{ P_2 \} C_2 \{ Q_2 \}}{\{ P_1 * P_2 \} C_1 || C_2 \{ Q_1 * Q_2 \}} \text{ PAR}$$

Disjoint Concurrency Example

deleteTree(n) || deleteTree(m)

Disjoint Concurrency Example

{ $\alpha \leftarrow n[t] * \beta \leftarrow m[t']$ }

deleteTree(n) || deleteTree(m)

Disjoint Concurrency Example

$\{ \alpha \leftarrow n[t] * \beta \leftarrow m[t'] \}$

$\{ \alpha \leftarrow n[t] \} \parallel \{ \beta \leftarrow m[t'] \}$

deleteTree(n) deleteTree(m)

Disjoint Concurrency Example

$$\{ \alpha \leftarrow n[t] * \beta \leftarrow m[t'] \}$$
$$\{ \alpha \leftarrow n[t] \} \quad \parallel \quad \{ \beta \leftarrow m[t'] \}$$
$$\text{deleteTree}(n) \quad \parallel \quad \text{deleteTree}(m)$$
$$\{ \alpha \leftarrow \emptyset \} \quad \parallel \quad \{ \beta \leftarrow \emptyset \}$$

Disjoint Concurrency Example

$$\begin{array}{c} \{ \alpha \leftarrow n[t] * \beta \leftarrow m[t'] \} \\ \\ \{ \alpha \leftarrow n[t] \} \quad \parallel \quad \{ \beta \leftarrow m[t'] \} \\ \text{deleteTree}(n) \quad \quad \quad \text{deleteTree}(m) \\ \{ \alpha \leftarrow \emptyset \} \quad \quad \quad \{ \beta \leftarrow \emptyset \} \\ \\ \{ \alpha \leftarrow \emptyset * \beta \leftarrow \emptyset \} \end{array}$$

More Than Just Disjoint

$$\frac{\{ P \} \subset \{ Q \}}{\{ \pi_r \circledR (RI_r * P) \} \text{ res } r \text{ in } \{ C \} \{ \pi_r \circledR (RI_r * Q) \}}^{\text{RES}}$$

$$\frac{\{ \pi_r \circledR (RI_r * P \wedge B) \} \subset \{ \pi_r \circledR (RI_r * Q) \}}{\{ P \} \text{ with } r \text{ when } B \text{ do } \{ C \} \{ Q \}}^{\text{CCR}}$$

Complex Concurrency Example

```
with r do{
```

```
    a := getLeft(n)  
}  
  
deleteTree(a)
```

```
||  
with r do{  
    b := getRight(n)  
}  
deleteTree(b)
```

Complex Concurrency Example

$$\begin{aligned} RI_r &= \alpha \leftarrow \beta \otimes n[\gamma] \otimes \delta \\ \pi_r &= \beta, \delta \end{aligned}$$

with r do{

```
a := getLeft(n)  
}  
deleteTree(a)
```

```
||  
with r do{  
    b := getRight(n)  
}  
deleteTree(b)
```

Complex Concurrency Example

{ $\beta \leftarrow p[t]$ }

with r do{

a := getLeft(n)

}

deleteTree(a)

$$\begin{aligned} RI_r &= \alpha \leftarrow \beta \otimes n[\gamma] \otimes \delta \\ \pi_r &= \beta, \delta \end{aligned}$$

with r do{

b := getRight(n)

}

deleteTree(b)

Complex Concurrency Example

{ $\beta \leftarrow p[t]$ }

with r do{

{ $\pi_r \circledR (RI_r * \beta \leftarrow p[t])$ }

a := getLeft(n)
}
deleteTree(a)

$$RI_r = \alpha \leftarrow \beta \otimes n[\gamma] \otimes \delta$$

$$\pi_r = \beta, \delta$$

with r do{
 b := getRight(n)
}
 deleteTree(b)

Complex Concurrency Example

{ $\beta \leftarrow p[t]$ }

with r do{

{ $\pi_r @ (RI_r * \beta \leftarrow p[t])$ }
{ $\delta @ (\alpha \leftarrow p[t] \otimes n[\gamma] \otimes \delta)$ }
a := getLeft(n)

}

deleteTree(a)

$$RI_r = \alpha \leftarrow \beta \otimes n[\gamma] \otimes \delta$$

$$\pi_r = \beta, \delta$$

with r do{

b := getRight(n)

}

deleteTree(b)

Complex Concurrency Example

{ $\beta \leftarrow p[t]$ }

with r do{

{ $\pi_r @ (RI_r * \beta \leftarrow p[t])$ }
{ $\delta @ (\alpha \leftarrow p[t] \otimes n[\gamma] \otimes \delta)$ }
a := getLeft(n)
{ $\delta @ (\alpha \leftarrow p[t] \otimes n[\gamma] \otimes \delta \wedge (a=p))$ }
}

deleteTree(a)

$$RI_r = \alpha \leftarrow \beta \otimes n[\gamma] \otimes \delta$$
$$\pi_r = \beta, \delta$$

with r do{
b := getRight(n)
}
deleteTree(b)

Complex Concurrency Example

{ $\beta \leftarrow p[t]$ }

with r do{

```
{  $\pi_r \circledR (RI_r * \beta \leftarrow p[t])$  }
{  $\delta \circledR (\alpha \leftarrow p[t] \otimes n[\gamma] \otimes \delta)$  }
a := getLeft(n)
{  $\delta \circledR (\alpha \leftarrow p[t] \otimes n[\gamma] \otimes \delta \wedge (a=p))$  }
{  $\pi_r \circledR (RI_r * \beta \leftarrow p[t] \wedge (a=p))$  }
}
```

deleteTree(a)

$$\begin{aligned} RI_r &= \alpha \leftarrow \beta \otimes n[\gamma] \otimes \delta \\ \pi_r &= \beta, \delta \end{aligned}$$

with r do{
 b := getRight(n)
}
deleteTree(b)

Complex Concurrency Example

{ $\beta \leftarrow p[t]$ }

with r do{

{ $\pi_r \circledR (RI_r * \beta \leftarrow p[t])$ }
{ $\delta \circledR (\alpha \leftarrow p[t] \otimes n[\gamma] \otimes \delta)$ }
 $a := \text{getLeft}(n)$
{ $\delta \circledR (\alpha \leftarrow p[t] \otimes n[\gamma] \otimes \delta \wedge (a=p))$ }
{ $\pi_r \circledR (RI_r * \beta \leftarrow p[t] \wedge (a=p))$ }
}
{ $\beta \leftarrow p[t] \wedge (a=p)$ }
deleteTree(a)

$$RI_r = \alpha \leftarrow \beta \otimes n[\gamma] \otimes \delta$$
$$\pi_r = \beta, \delta$$

with r do{
 $b := \text{getRight}(n)$
}
 deleteTree(b)

Complex Concurrency Example

{ $\beta \leftarrow p[t]$ }

with r do{

{ $\pi_r \circledR (RI_r * \beta \leftarrow p[t])$ }
{ $\delta \circledR (\alpha \leftarrow p[t] \otimes n[\gamma] \otimes \delta)$ }
a := getLeft(n)
{ $\delta \circledR (\alpha \leftarrow p[t] \otimes n[\gamma] \otimes \delta \wedge (a=p))$ }
{ $\pi_r \circledR (RI_r * \beta \leftarrow p[t] \wedge (a=p))$ }
}

{ $\beta \leftarrow p[t] \wedge (a=p)$ }

deleteTree(a)

{ $\beta \leftarrow \emptyset$ }

$$RI_r = \alpha \leftarrow \beta \otimes n[\gamma] \otimes \delta$$
$$\pi_r = \beta, \delta$$

with r do{
 b := getRight(n)
}
deleteTree(b)

Complex Concurrency Example

$\{ \beta \leftarrow p[t] \}$

with r do{

```

    {  $\pi_r \circledR (RI_r * \beta \leftarrow p[t])$  }
    {  $\delta \circledR (\alpha \leftarrow p[t] \otimes n[\gamma] \otimes \delta)$  }
    a := getLeft(n)
    {  $\delta \circledR (\alpha \leftarrow p[t] \otimes n[\gamma] \otimes \delta \wedge (a=p))$  }
    {  $\pi_r \circledR (RI_r * \beta \leftarrow p[t] \wedge (a=p))$  }
}

```

$\{ \beta \leftarrow p[t] \wedge (a=p) \}$

deleteTree(a)

$\{ \beta \leftarrow \emptyset \}$

$$RI_r = \alpha \leftarrow \beta \otimes n[\gamma] \otimes \delta$$

$$\pi_r = \beta, \delta$$

$\{ \delta \leftarrow q[t'] \}$

with r do{

b := getRight(n)

}

deleteTree(b)

$\{ \delta \leftarrow \emptyset \}$

Summary

- ⦿ Separation Logic allows low-level local reasoning for sequential and concurrent programs.
- ⦿ Context Logic allows high-level local reasoning for sequential programs.
- ⦿ Small axioms are necessary for local reasoning about concurrency.
- ⦿ Segment Logic allows high-level local reasoning about concurrent programs.

Ongoing Research

- ⦿ Abstract Local Reasoning : translating from specification to implementation.
(with Philippa Gardner and Thomas Dinsdale-Young - Imperial)
- ⦿ Concurrent XML Update : designing and specifying a language in the style of DOM.
(with James Kearney - Imperial)
- ⦿ Applying Rely-Guarantee and Deny-Guarantee techniques to BTrees.
(with Pedro da Rocha Pinto and Thomas Dinsdale-Young - Imperial)

Thanks for Listening
Any Questions ?