Small Specifications for Tree Update

(Cutting up trees any which way)

Mark Wheelhouse
Philippa Gardner

Imperial College London

with thanks to Thomas Dinsdale-Young

Overview

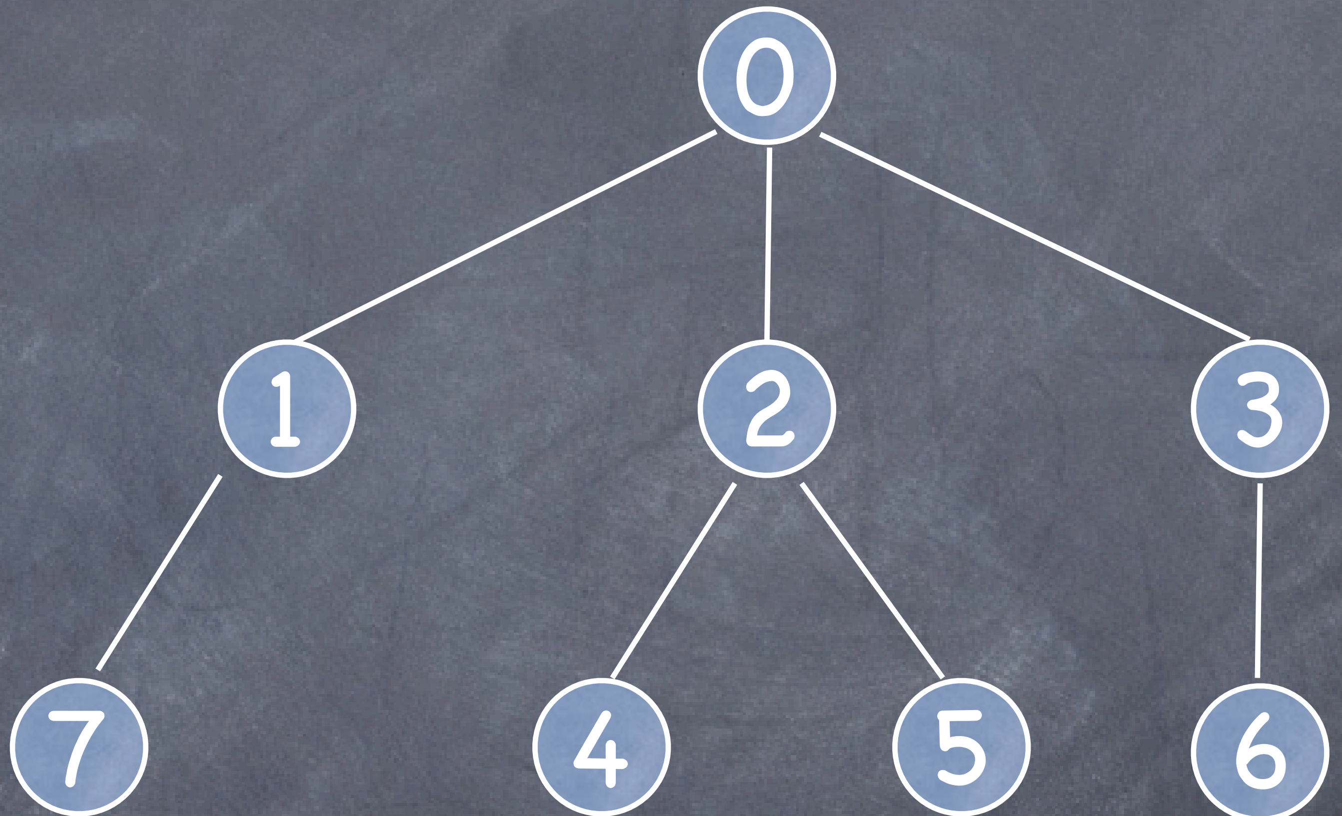
- Append - the command and its problems
- Our Model - a new data structure
- Tree Update Language - commands and OS
- The Logic - syntax and semantics
- Local Hoare Reasoning - small axioms
- Concluding Remarks

Overview

- Append - the command and its problems
- Our Model - a new data structure
- Tree Update Language - commands and OS
- The Logic - syntax and semantics
- Local Hoare Reasoning - small axioms
- Concluding Remarks

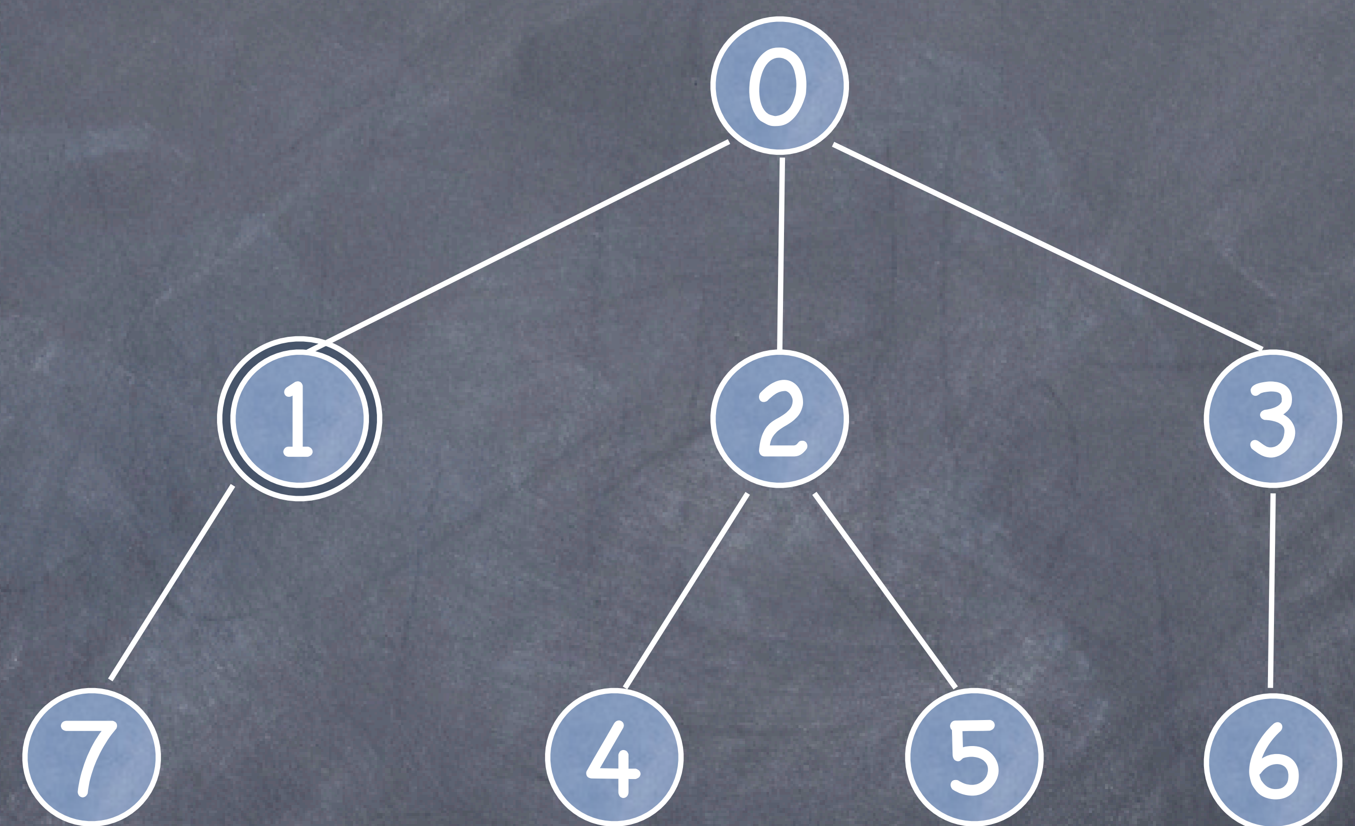
Append

append(1 , 3)



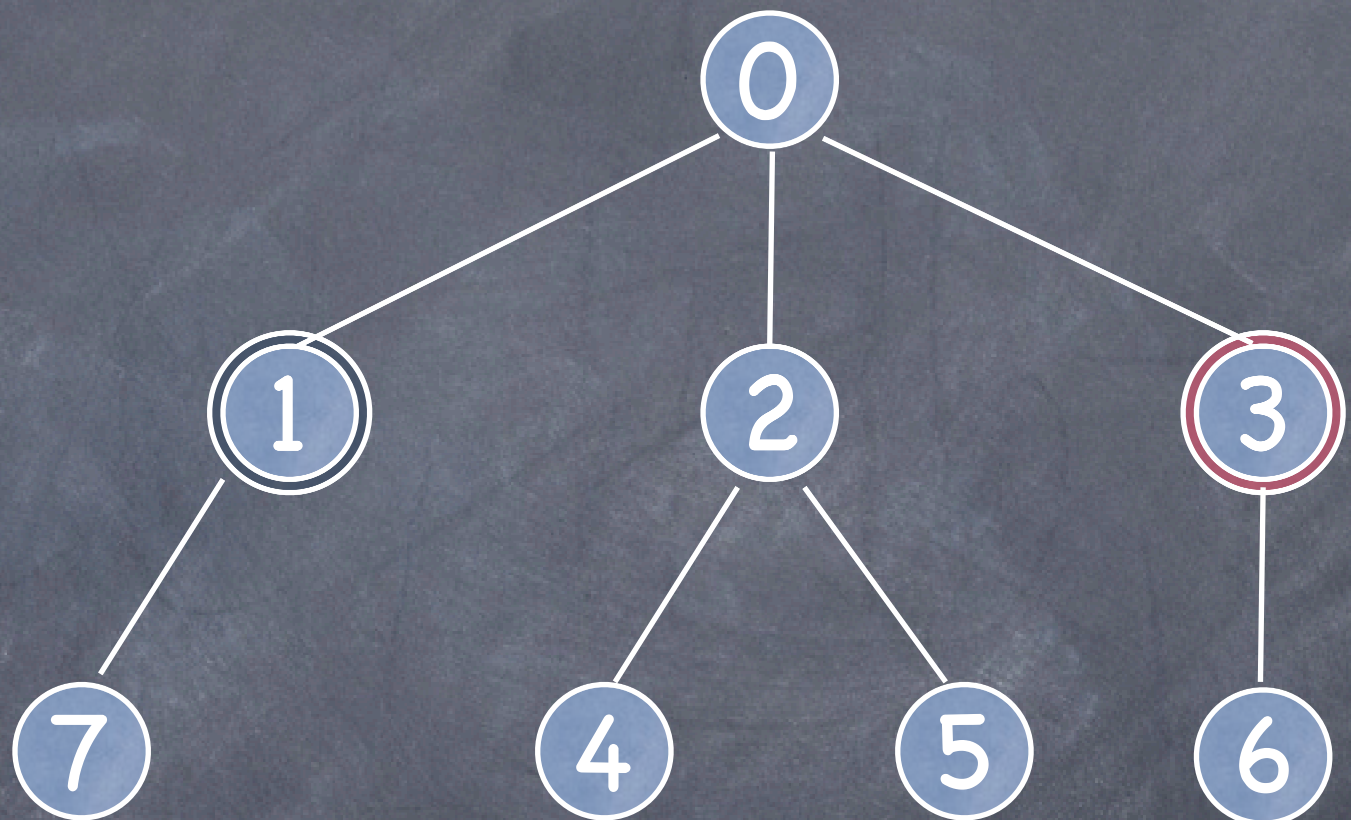
Append

append(①, 3)



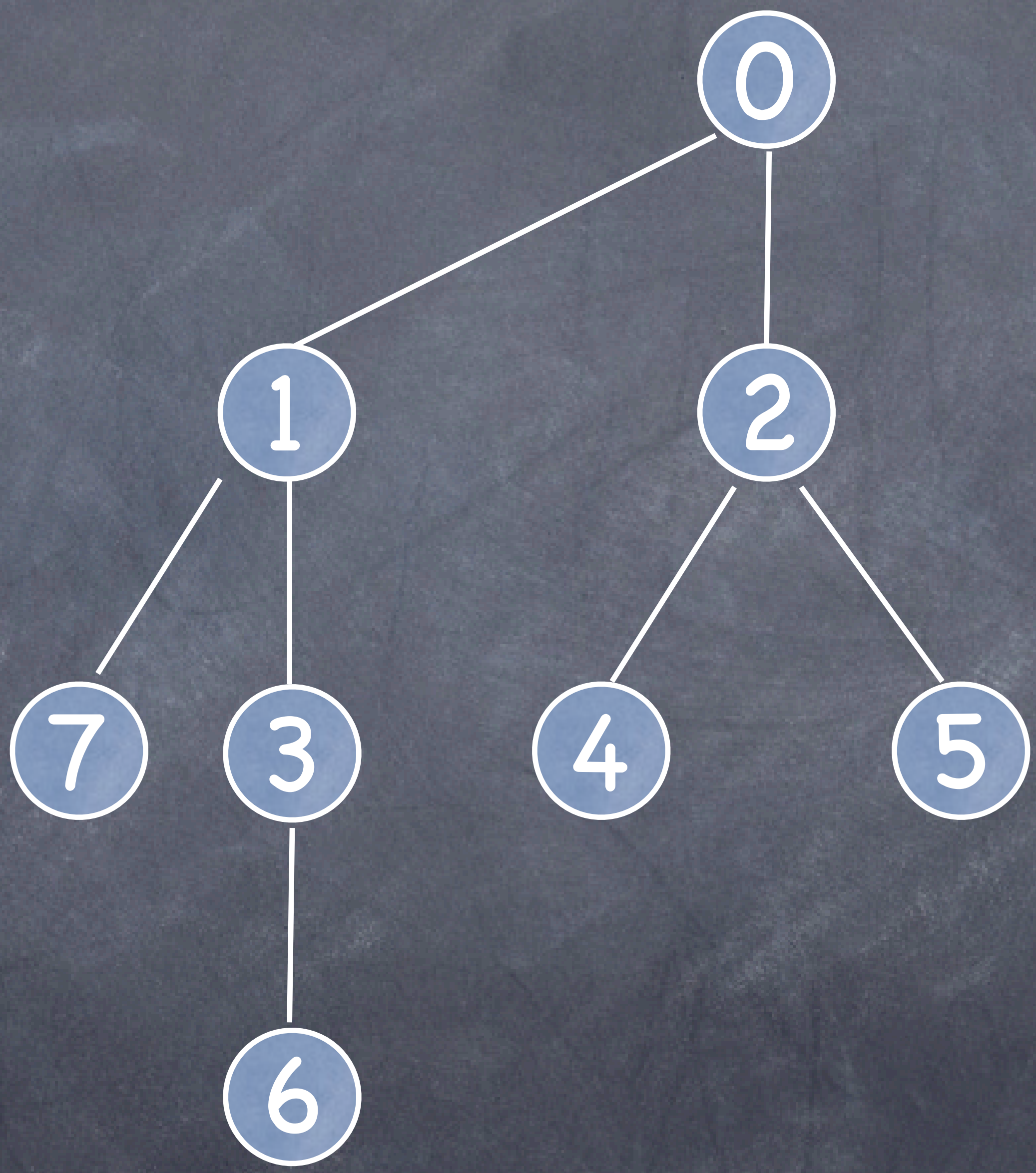
Append

append(①, ③)



Append

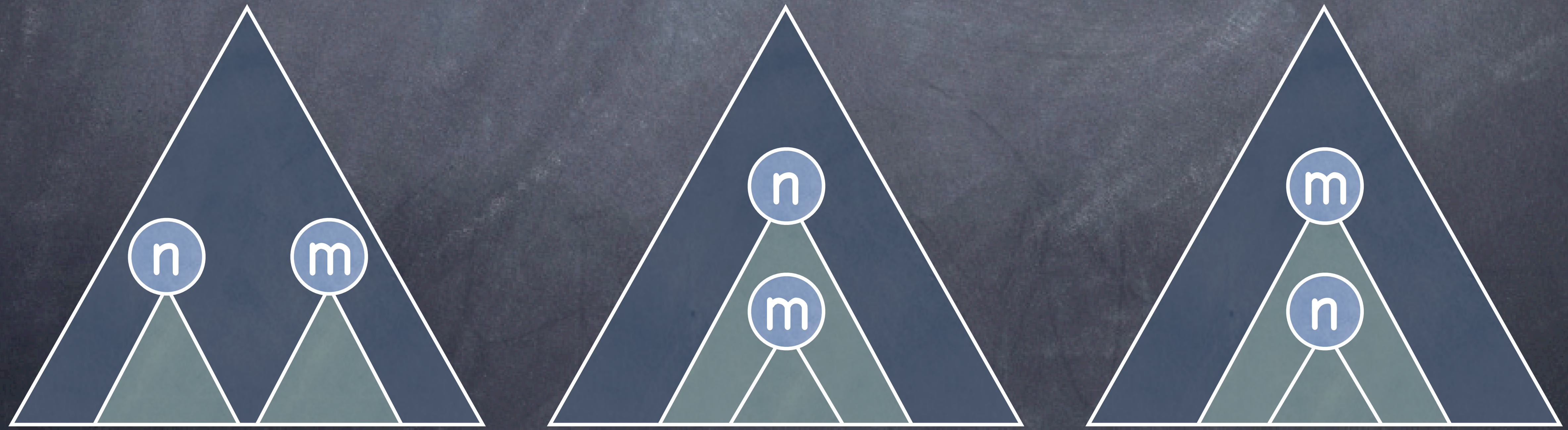
append(1 , 3)



3 Cases of Append

append(n,m)

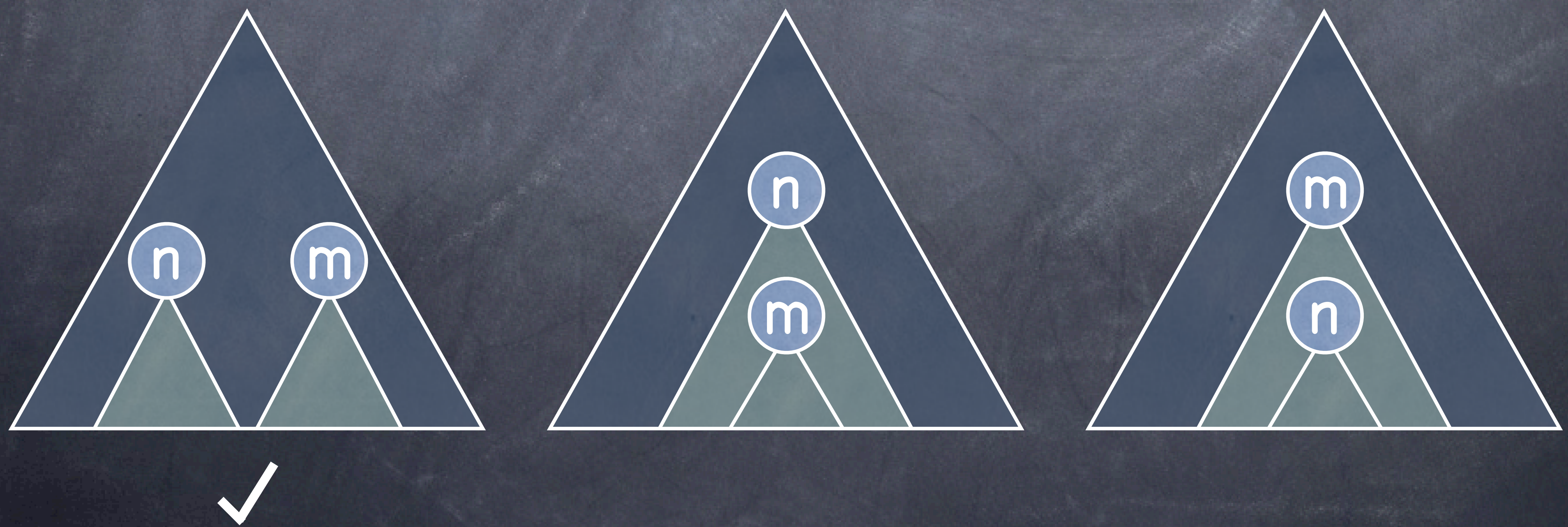
subtrees disjoint | n ancestor of m | m ancestor of n



3 Cases of Append

append(n,m)

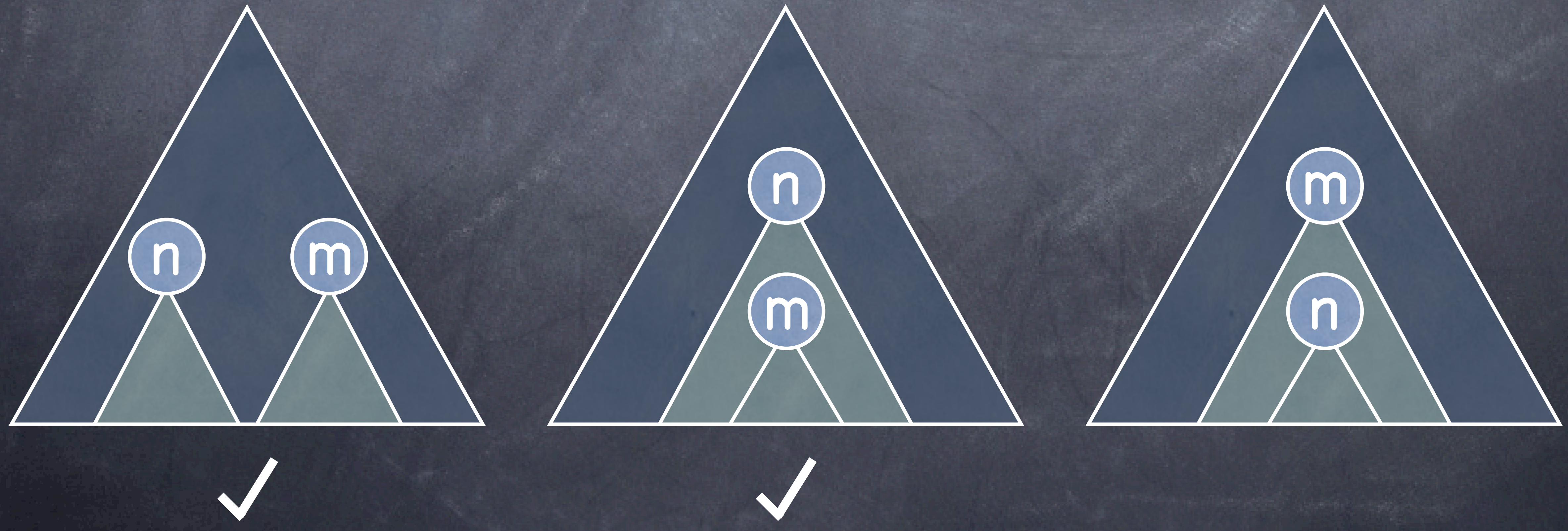
subtrees disjoint | n ancestor of m | m ancestor of n



3 Cases of Append

append(n,m)

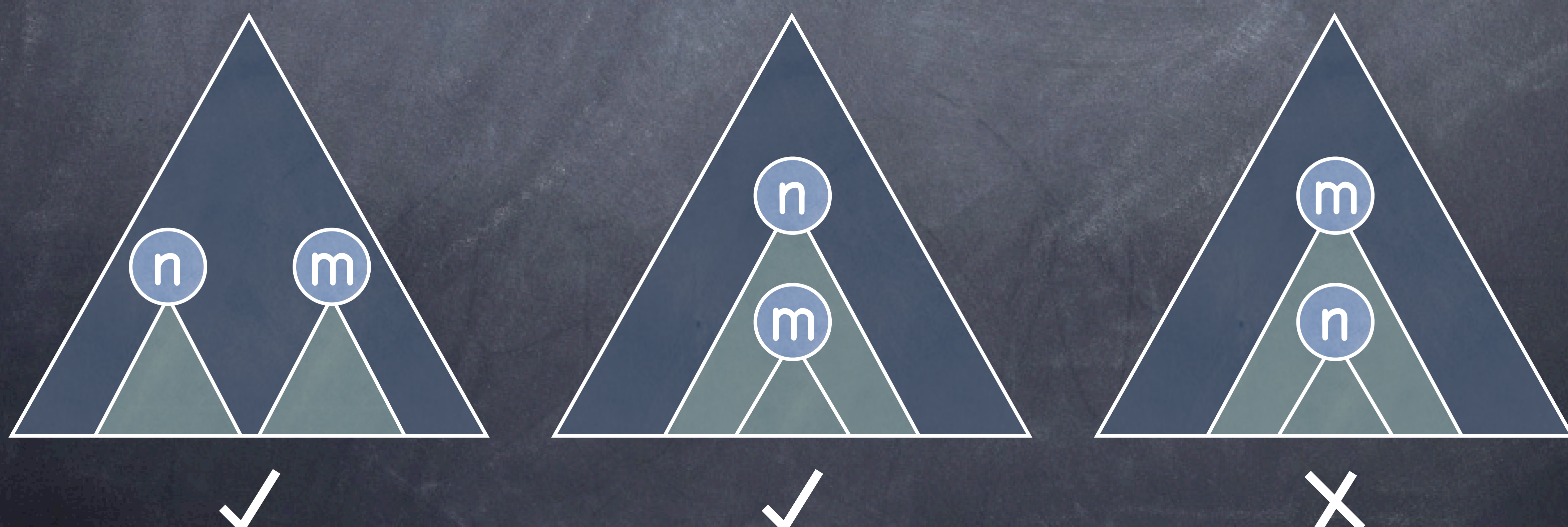
subtrees disjoint	n ancestor of m	m ancestor of n



3 Cases of Append

append(n,m)

subtrees disjoint | n ancestor of m | m ancestor of n

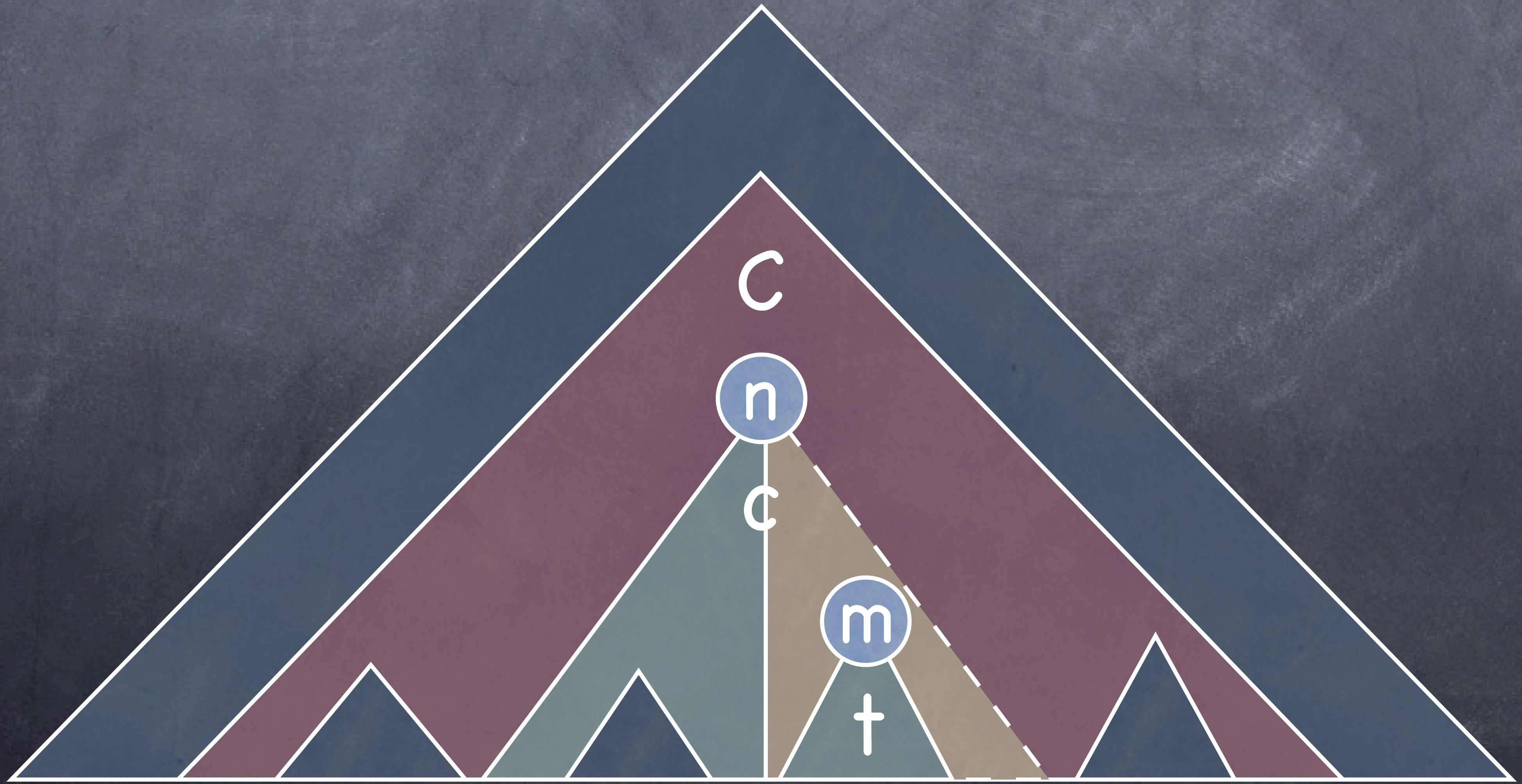


Current Specification

$$\{ (C \circ_y n[c]) \circ_x m[t] \}$$

append(m,n)

$$\{ (C \circ_y n[\, c \otimes m[t] \,]) \circ_x \emptyset_C \}$$

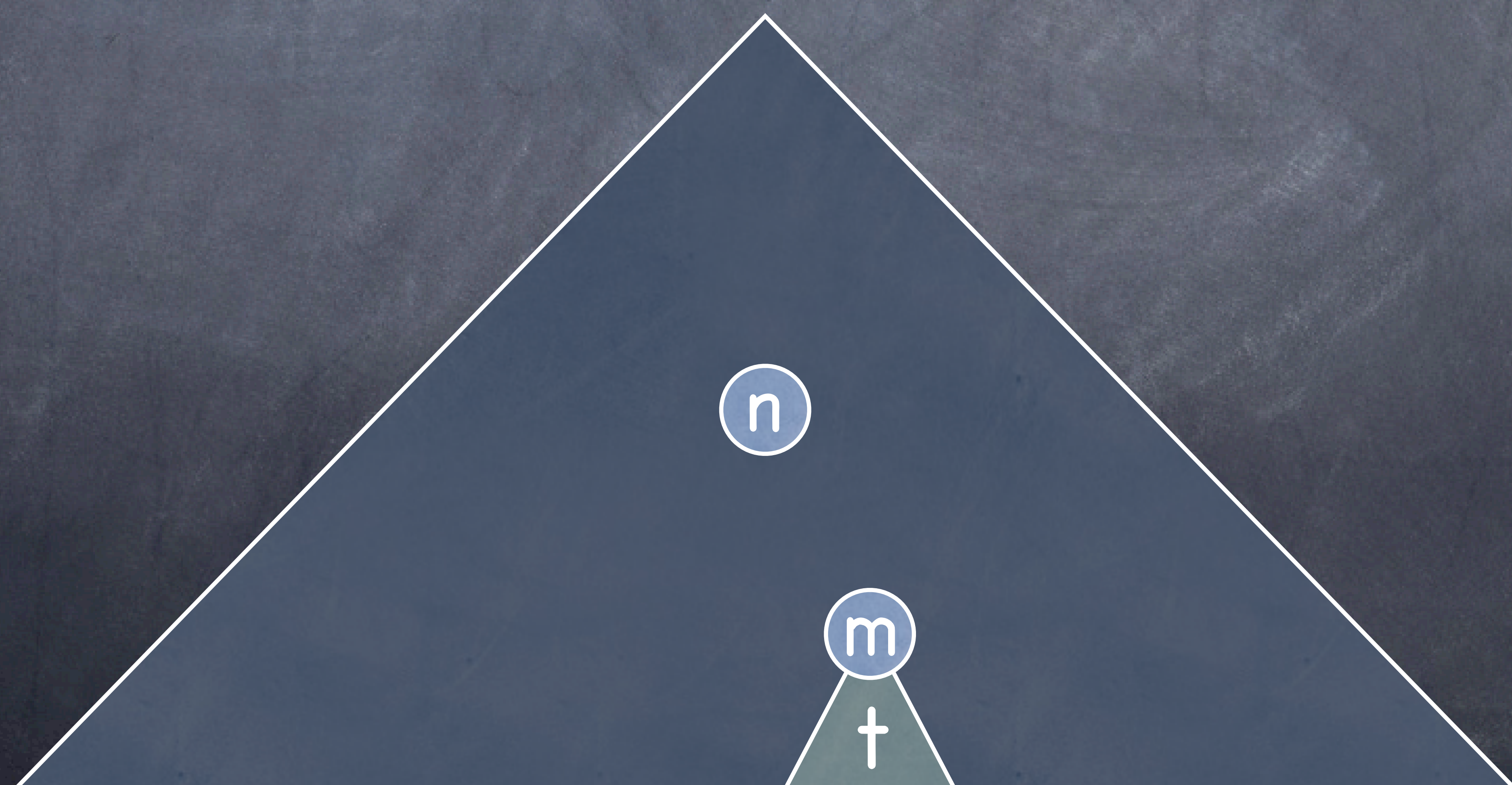


What We Would Like

$$\{\ n[z] \bullet_{?} m[t]\ \}$$

append(m,n)

$$\{\ n[\ z \otimes m[t]\ ] \bullet_{?} \emptyset_C\ \}$$



What We Would Like

From Separation Logic:

$$x \mapsto y,z \ast y \mapsto \_ \;=\; y \mapsto \_ \ast x \mapsto y,z$$

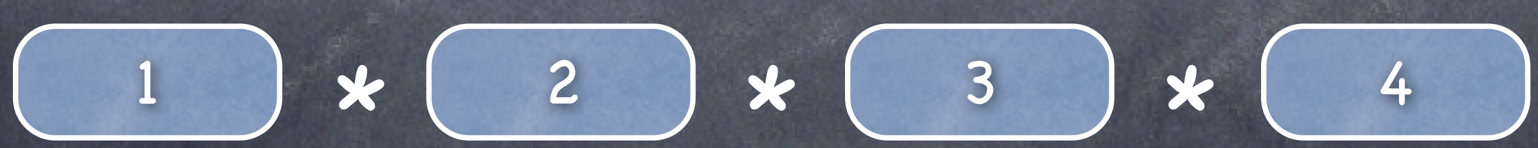
associativity and commutativity of $\ast$

What We Would Like

From Separation Logic:

$$x \mapsto y,z \ast y \mapsto \_ \;=\; y \mapsto \_ \ast x \mapsto y,z$$

<u>associativity</u> and <u>commutativity</u> of $\ast$



What We Would Like

From Separation Logic:

$$x \mapsto y,z * y \mapsto \_ \;=\; y \mapsto \_ * x \mapsto y,z$$

<u>associativity</u> and <u>commutativity</u> of *

2 * 1 * 3 * 4

What We Would Like

From Separation Logic:

$$x \mapsto y,z * y \mapsto \_ = y \mapsto \_ * x \mapsto y,z$$

associativity and commutativity of *

2 * 1 * 4 * 3

What We Would Like

From Separation Logic:

$$x \mapsto y,z \ast y \mapsto \_ \;=\; y \mapsto \_ \ast x \mapsto y,z$$

<u>associativity</u> and <u>commutativity</u> of *

2 * 4 * 1 * 3

What We Would Like

From Multi-holed Context Logic:

$$p[\ x\ ] \circ_x n[t] = p[\ n[t]\ ]$$

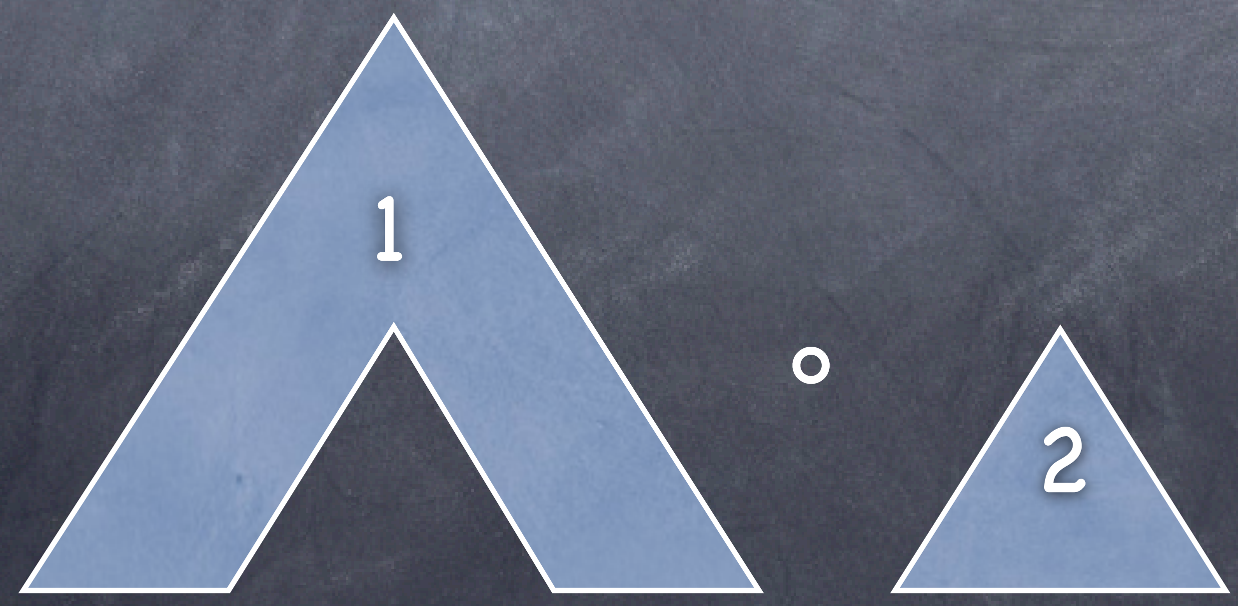
reduction of application $\circ$

What We Would Like

From Multi-holed Context Logic:

$$p[\ x\ ] \circ_x n[t] = p[\ n[t]\ ]$$

reduction of application ∘

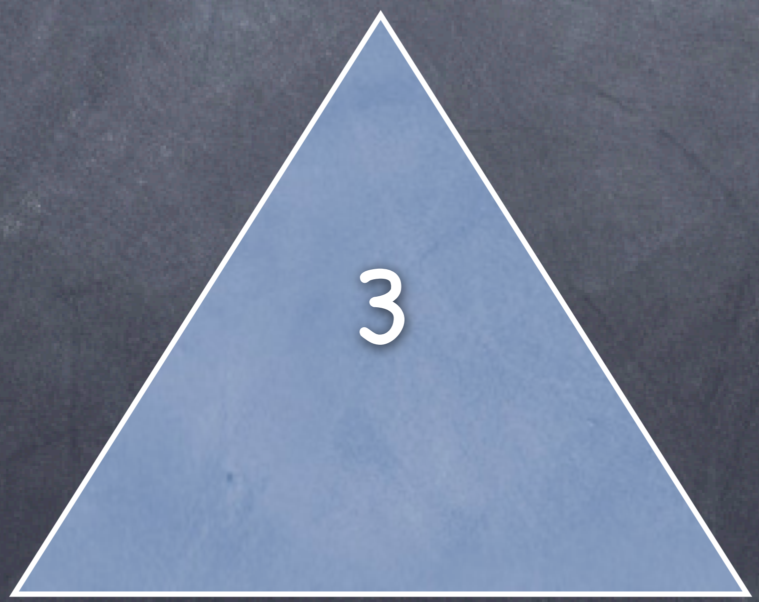


What We Would Like

From Multi-holed Context Logic:

$$p[\ x\ ] \circ_x n[t] = p[\ n[t]\ ]$$

reduction of application ∘



The Idea

Instead of labeling the application function...

$$p[\ x\ ] \circ_x n[t] = p[\ n[t]\ ]$$

...we label the data going into the context hole

$$p[\ x\ ] * x \leftarrow n[t] = p[\ n[t]\ ]$$

We Have to be Careful

$*$ is <u>not</u> associative under reduction.

$$(x \leftarrow n[y] * y \leftarrow m[\emptyset]) * y \leftarrow \emptyset = x \leftarrow n[m[\emptyset]] * y \leftarrow \emptyset$$

but,

$$x \leftarrow n[y] * (y \leftarrow m[\emptyset] * y \leftarrow \emptyset) = \ ??$$

so we need something new...

Overview

- Append - the command and its problems
- Our Model - a new data structure
- Tree Update Language - commands and OS
- The Logic - syntax and semantics
- Local Hoare Reasoning - small axioms
- Concluding Remarks

The Model

We have countably infinite and <u>disjoint</u> sets:

Location Names	Context Labels
ID = {m, n, p, q, ...}	X = {x, y, z, ...}

The Model

We have countably infinite and <u>disjoint</u> sets:

Location Names	Context Labels
ID = {m, n, p, q, ...}	X = {x, y, z, ...}

We define:

Tree Contexts	Tree Fragments
$c \in C_{ID,X}$	$f \in F_{ID,X}$

Data Structure

Data Structure

Tree Context $c ::= \emptyset_C \mid x \mid n[c] \mid c \otimes c$

Data Structure

Tree Context $c ::= \emptyset_C \mid x \mid n[c] \mid c \otimes c$

Tree Fragments $f ::= \emptyset_F \mid x \leftarrow c \mid f + f \mid (\nu x)(f)$

Data Structure

Tree Context $c ::= \emptyset_C \mid x \mid n[c] \mid c \otimes c$

Tree Fragments $f ::= \emptyset_F \mid x \leftarrow c \mid f + f \mid (\nu x)(f)$

unique location names n

unique free hole addresses $x \leftarrow$

unique free hole labels x

+ associative & commutative with id $\emptyset_F$

$\otimes$ associative with id $\emptyset_C$ - i.e. ordered trees

Data Structure

Tree Context $c ::= \emptyset_C \mid x \mid n[c] \mid c \otimes c$

Tree Fragments $f ::= \emptyset_F \mid x \leftarrow c \mid f + f \mid (\nu x)(f)$

unique location names n

unique free hole addresses $x\leftarrow$

unique free hole labels x

+ associative & commutative with id $\emptyset_F$

$\otimes$ associative with id $\emptyset_C$ - i.e. ordered trees

+ no cycles!

Data Structure

Tree Context $c ::= \emptyset_C \mid x \mid n[c] \mid c \otimes c$

Tree Fragments $f ::= \emptyset_F \mid x \leftarrow c \mid f + f \mid (\nu x)(f)$

unique location names n

unique free hole addresses $x \leftarrow$

unique free hole labels x

+ no cycles!

+ associative & commutative with id $\emptyset_F$

$\otimes$ associative with id $\emptyset_C$ - i.e. ordered trees

The set of hole labels that occur free in c is denoted fn(c)

The set of hole labels and addresses that occur free in f is denoted fn(f)

Examples - Syntax

$n[\emptyset] \otimes n[\emptyset]$ is undefined!

$x \leftarrow n[\emptyset] + y \leftarrow n[\emptyset]$ is undefined!

$x \leftarrow \emptyset + x \leftarrow \emptyset$ is undefined!

$x \leftarrow y + z \leftarrow y$ is undefined!

$x \leftarrow y + y \leftarrow z + z \leftarrow x$ is undefined!

$(\nu x)(x \leftarrow \emptyset) + x \leftarrow \emptyset$ is well defined

$(\nu y)(x \leftarrow y) + z \leftarrow y$ is well defined

Tree Context Application

$$ap_x(c_1, c_2) = \begin{cases} c_1[c_2/x] & \text{if } x \in fn(c_1) \\ & \text{and } fn(c_1) \cup fn(c_2) \subseteq \{x\} \\ \text{undefined} & \text{otherwise} \end{cases}$$

shorthand: $ap_x(c_1, c_2)$ written as $c_1 \circ_x c_2$

Tree Fragment Equivalence

disjoint union axioms:

$$f + \emptyset_F \equiv f$$

$$f_1 + f_2 \equiv f_2 + f_1$$

$$f_1 + (f_2 + f_3) \equiv (f_1 + f_2) + f_3$$

alpha conversion:

$$(\nu x)(f) \equiv (\nu y)(f[y/x]) \quad \text{if} \quad y \notin fn(f)$$

Tree Fragment Equivalence

restriction axioms:

$$(\nu x)(\emptyset_F) \equiv \emptyset_F$$

$$(\nu x)(\nu y)(f) \equiv (\nu y)(\nu x)(f)$$

$$(\nu x)(y \leftarrow c + f) \equiv y \leftarrow c + (\nu x)(f) \quad \text{if } x \neq y \text{ and } x \notin fn(c)$$

$$(\nu x)(y \leftarrow c_1 + x \leftarrow c_2) \equiv y \leftarrow c_1 \circ_x c_2 \quad \text{if } x \in fn(c_1)$$

Overview

- Append - the command and its problems
- Our Model - a new data structure
- Tree Update Language - commands and OS
- The Logic - syntax and semantics
- Local Hoare Reasoning - small axioms
- Concluding Remarks

Tree Shapes

$$t_\bullet \in T_\bullet$$

$$t_\bullet ::= \emptyset_T \mid \bullet[t_\bullet] \mid t_\bullet \otimes t_\bullet$$

Tree Shapes

$$t_\bullet \in T_\bullet$$

$$t_\bullet ::= \emptyset_T \mid \bullet[t_\bullet] \mid t_\bullet \otimes t_\bullet$$

Note: tree shapes have <u>no</u> holes

Tree Shapes

$$t_\bullet \in T_\bullet$$

$$t_\bullet ::= \emptyset_T \mid \bullet[t_\bullet] \mid t_\bullet \otimes t_\bullet$$

Note: tree shapes have <u>no</u> holes

write < t > for the shape of tree t, where:

$$\langle \emptyset_C \rangle = \emptyset_T$$

$$\langle n[t] \rangle = \bullet[\langle t \rangle]$$

$$\langle t \otimes t' \rangle = \langle t \rangle \otimes \langle t' \rangle$$

Program State

s,f

working tree fragment f

variable store s

$$s:(Var_{ID} \rightharpoonup_{fin} ID \cup \{null\}) \times (Var_{T_\bullet} \rightharpoonup_{fin} T_\bullet)$$

location name variables

tree shape variables

Tree Update Language

Tree Update Language

Node Commands

$n' := getUp(n)$

$n' := getRight(n)$

$n' := getLast(n)$

deleteNode(n)

insertNodeAbove(n,m)

moveNodeAbove(n,m)

appendNode(n,m)

Tree Update Language

Node Commands

n′ := getUp(n)

n′ := getRight(n)

n′ := getLast(n)

deleteNode(n)

insertNodeAbove(n,m)

moveNodeAbove(n,m)

appendNode(n,m)

Subtree Commands

t := copy(n)

deleteSubtree(n)

insertRight(n,T)

appendSub(n,m)

Tree Update Language

Node Commands

$n' := getUp(n)$

$n' := getRight(n)$

$n' := getLast(n)$

deleteNode(n)

insertNodeAbove(n,m)

moveNodeAbove(n,m)

appendNode(n,m)

Subtree Commands

t := copy(n)

deleteSubtree(n)

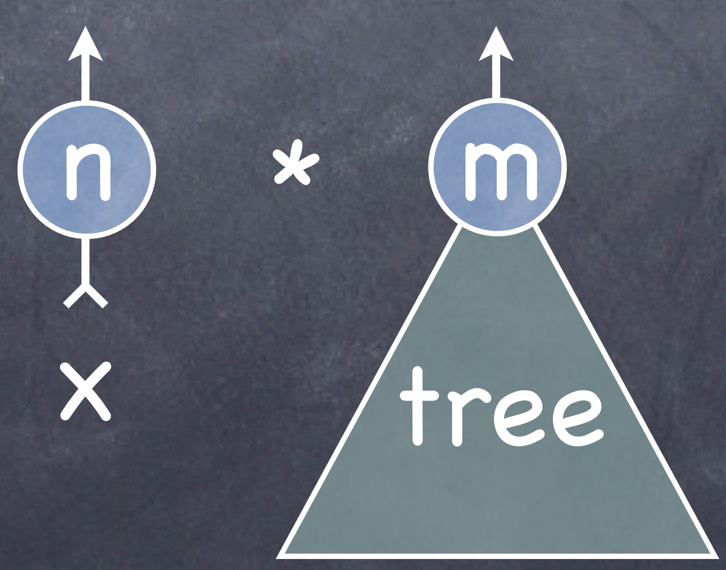
insertRight(n,T)

appendSub(n,m)

plus standard skip, variable assignment, sequencing, if-then-else and while commands

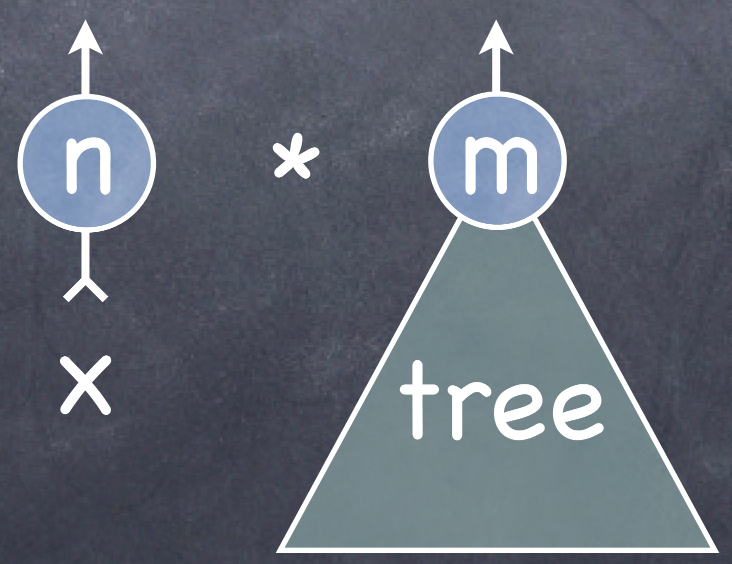
Append

$\text{append}(n,m) \triangleq$ insertRight(m,$\bullet[\emptyset_T]$);
t := getRight(m);
appendSub(t,m);
appendNode(n,m);
appendSub(m,t);
deleteNode(t);



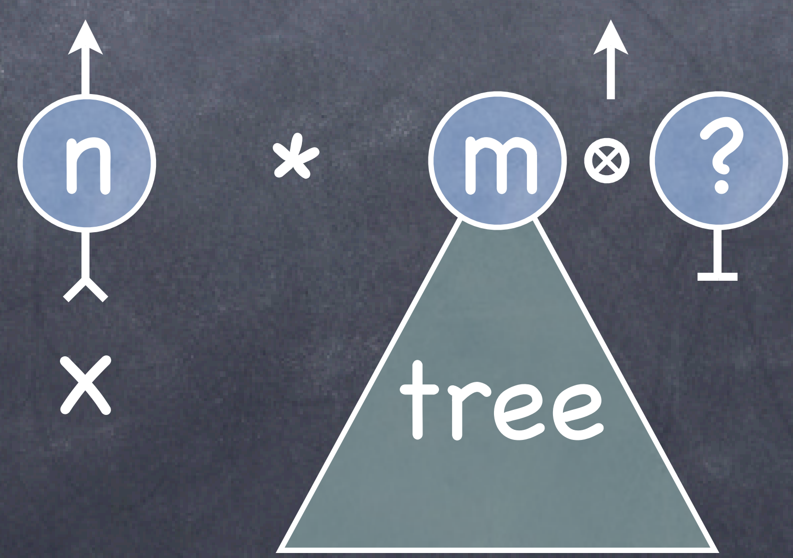
Append

$\text{append}(n,m) \triangleq$ insertRight(m,•[$\emptyset_T$]); ★
t := getRight(m);
appendSub(t,m);
appendNode(n,m);
appendSub(m,t);
deleteNode(t);



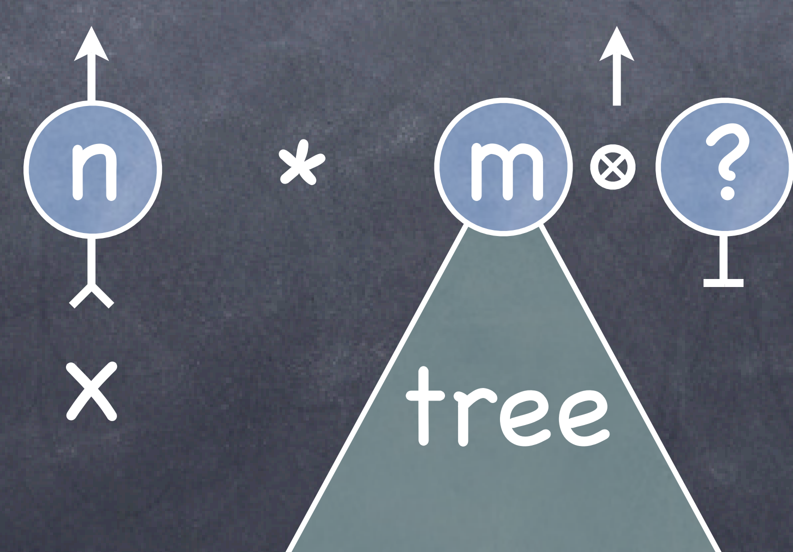
Append

append(n,m) $\triangleq$ insertRight(m,•[$\emptyset_T$]); ★
t := getRight(m);
appendSub(t,m);
appendNode(n,m);
appendSub(m,t);
deleteNode(t);



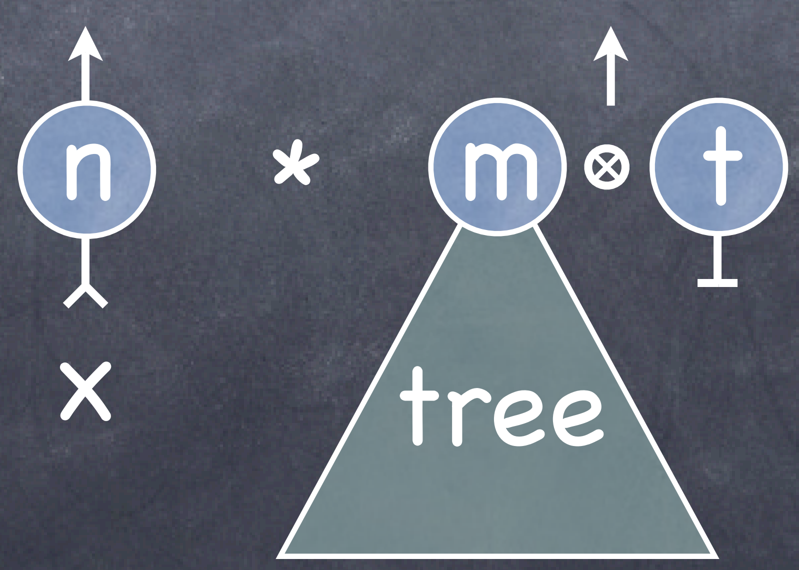
Append

append(n,m) $\triangleq$ insertRight(m,•[$\emptyset_T$]);
t := getRight(m);
appendSub(t,m);
appendNode(n,m);
appendSub(m,t);
deleteNode(t);



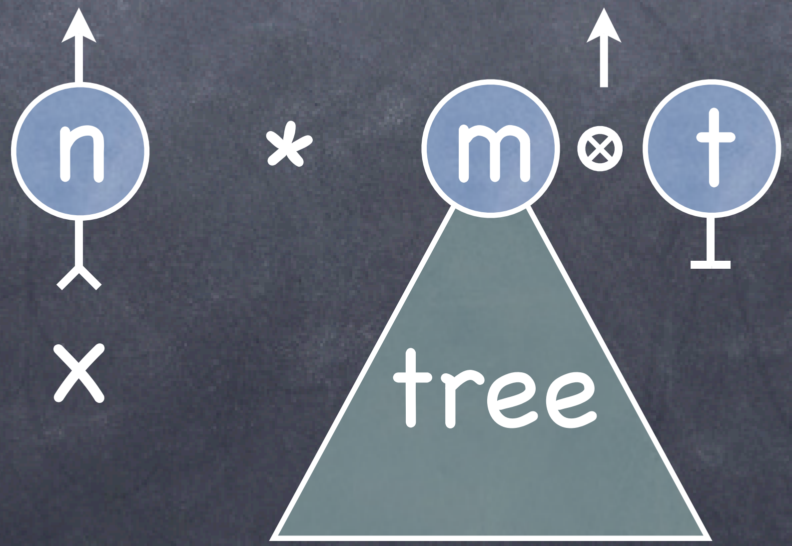
Append

$\text{append}(n,m) \triangleq$ insertRight(m,•[$\emptyset_T$]);
t := getRight(m); ★
appendSub(t,m);
appendNode(n,m);
appendSub(m,t);
deleteNode(t);



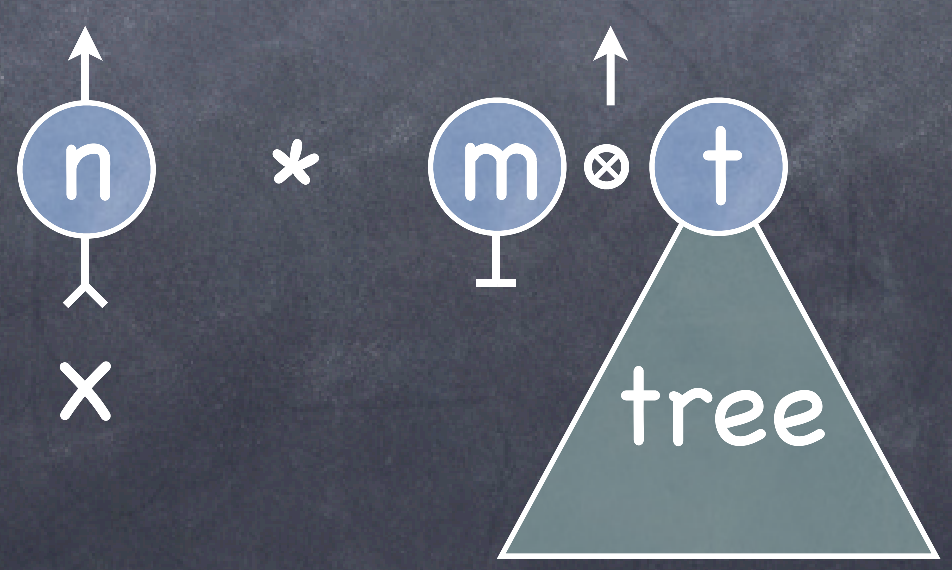
Append

append(n,m) $\triangleq$ insertRight(m,$\bullet[\emptyset_T]$);
t := getRight(m);
appendSub(t,m); ★
appendNode(n,m);
appendSub(m,t);
deleteNode(t);



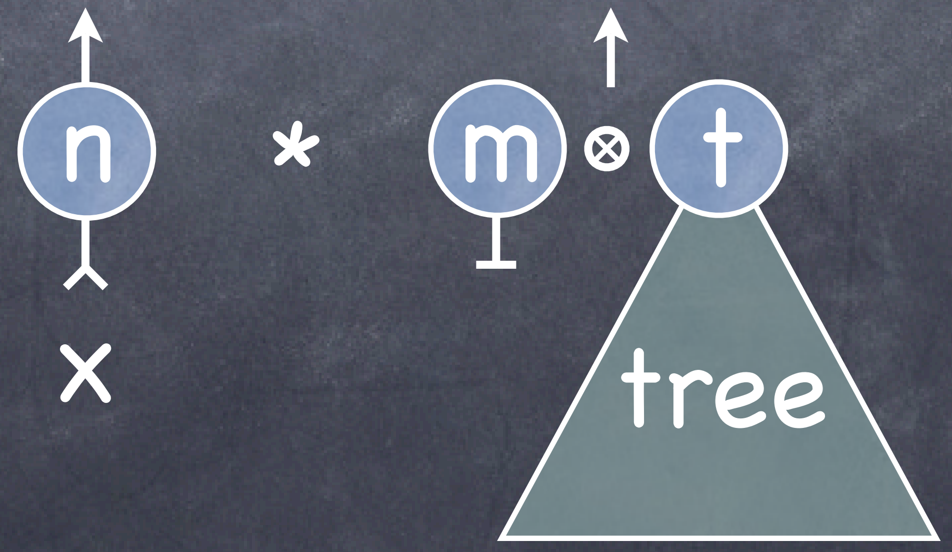
Append

append(n,m) ≜ insertRight(m,•[$\emptyset_T$]);
t := getRight(m);
appendSub(t,m); ★
appendNode(n,m);
appendSub(m,t);
deleteNode(t);



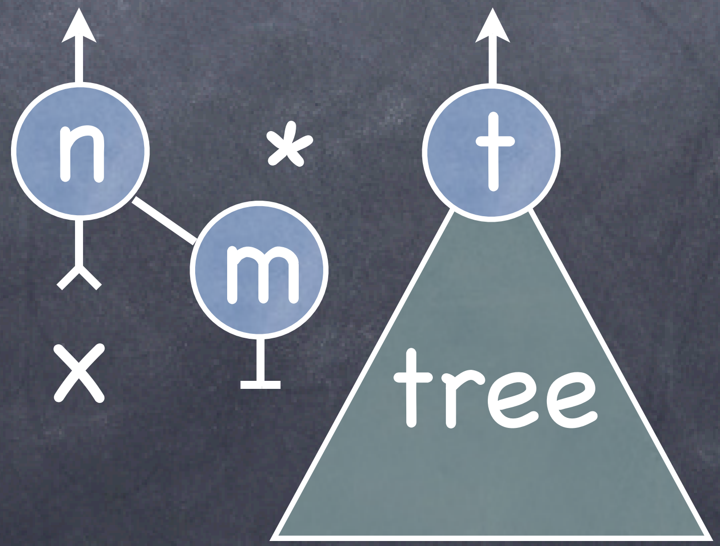
Append

$append(n,m) \triangleq$ insertRight(m,•[$\emptyset_T$]);
t := getRight(m);
appendSub(t,m);
appendNode(n,m); ★
appendSub(m,t);
deleteNode(t);



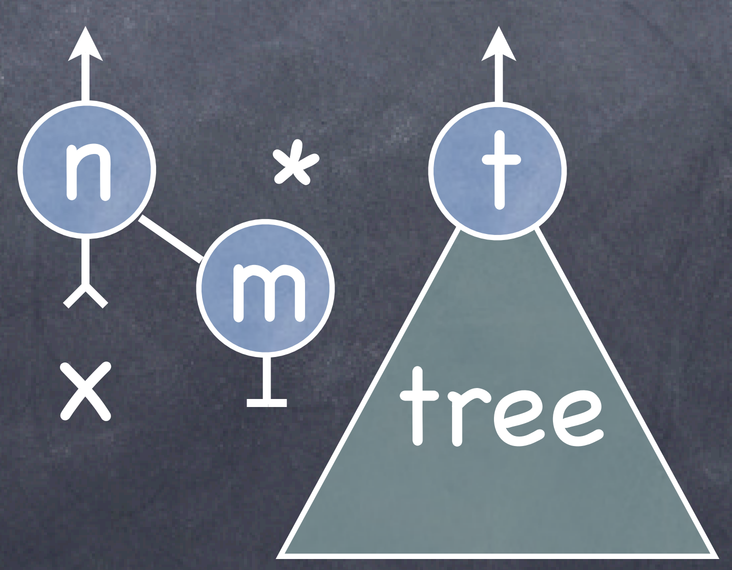
Append

append(n,m) $\triangleq$ insertRight(m,•[$\emptyset_T$]);
t := getRight(m);
appendSub(t,m);
appendNode(n,m);★
appendSub(m,t);
deleteNode(t);



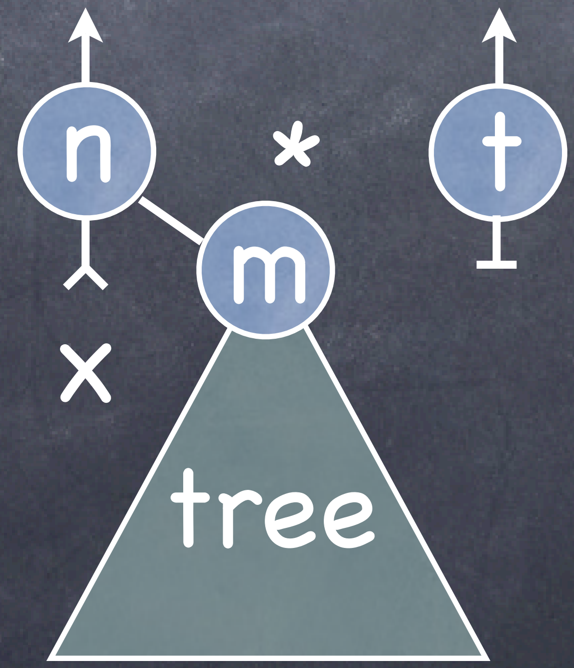
Append

append(n,m) $\triangleq$ insertRight(m,•[$\emptyset_T$]);
t := getRight(m);
appendSub(t,m);
appendNode(n,m);
appendSub(m,t); ★
deleteNode(t);



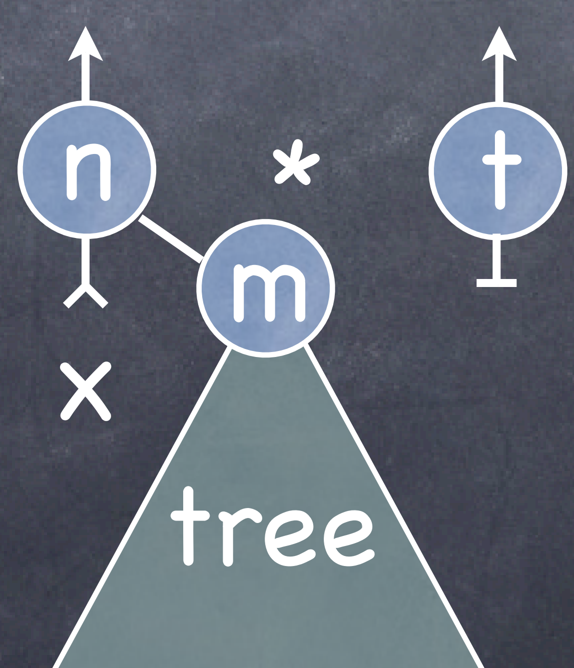
Append

append(n,m) $\triangleq$ insertRight(m,•[$\emptyset_T$]);
t := getRight(m);
appendSub(t,m);
appendNode(n,m);
appendSub(m,t); ★
deleteNode(t);



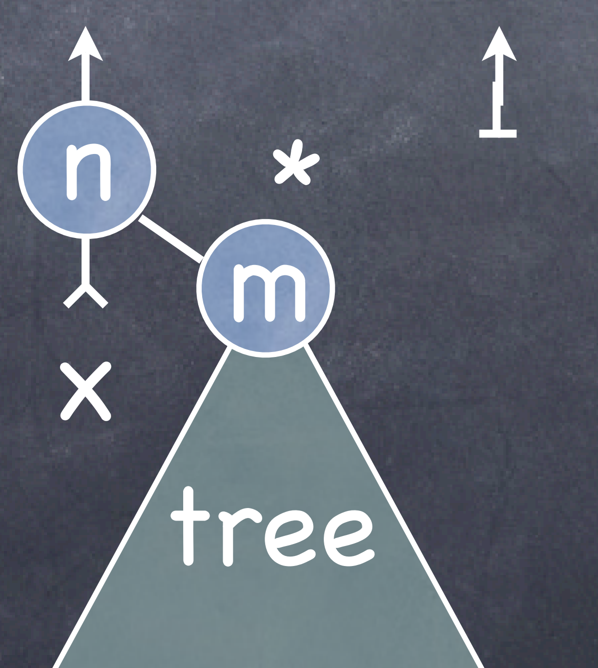
Append

append(n,m) $\triangleq$ insertRight(m,•[$\emptyset_T$]);
t := getRight(m);
appendSub(t,m);
appendNode(n,m);
appendSub(m,t);
deleteNode(t); ★



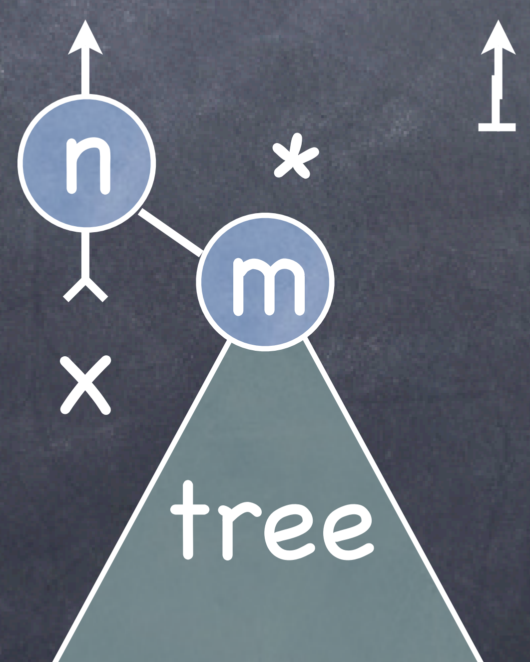
Append

append(n,m) $\triangleq$ insertRight(m,•[$\emptyset_T$]);
t := getRight(m);
appendSub(t,m);
appendNode(n,m);
appendSub(m,t);
deleteNode(t);



Append

append(n,m) ≜ insertRight(m,•[$\emptyset_T$]);
t := getRight(m);
appendSub(t,m);
appendNode(n,m);
appendSub(m,t);
deleteNode(t);



Operational Semantics

Defined over tree fragments

e.g.

Operational Semantics

Defined over tree fragments

e.g.

$$\frac{s(n)=\mathbf{n} \quad \mathbf{f} \equiv (\nu\ w,x,y,z)(\mathbf{f}'' + x \leftarrow \mathbf{n}[z] + y \leftarrow \mathbf{m}[w]) \qquad s(m)=\mathbf{m} \quad \mathbf{f}' \equiv (\nu\ w,x,y,z)(\mathbf{f}'' + x \leftarrow \mathbf{n}[z \otimes \mathbf{m}[\emptyset_C]] + y \leftarrow w)}{appendNode(n,m),s,\mathbf{f} \leadsto s,\mathbf{f}'}$$

Operational Semantics

Defined over tree fragments

e.g.

$$\frac{s(n)=\mathbf{n} \quad \mathbf{f} \equiv (\nu\ w,x,y,z)(\mathbf{f}'' + x \leftarrow \mathbf{n}[z] + y \leftarrow \mathbf{m}[w]) \qquad s(m)=\mathbf{m} \quad \mathbf{f}' \equiv (\nu\ w,x,y,z)(\mathbf{f}'' + x \leftarrow \mathbf{n}[z \otimes \mathbf{m}[\emptyset_C]] + y \leftarrow w)}{appendNode(n,m),s,\mathbf{f} \rightsquigarrow s,\mathbf{f}'}$$

$$\frac{s(n)=\mathbf{n} \quad \mathbf{f} \equiv (\nu\ x,y,z)(\mathbf{f}'' + x \leftarrow \mathbf{n}[z] + y \leftarrow \mathbf{m}[\mathbf{t}]) \qquad s(m)=\mathbf{m} \quad \mathbf{f}' \equiv (\nu\ x,y,z)(\mathbf{f}'' + x \leftarrow \mathbf{n}[z \otimes \mathbf{t}] + y \leftarrow m[\emptyset_C])}{appendSub(n,m),s,\mathbf{f} \rightsquigarrow s,\mathbf{f}'}$$

Overview

- Append - the command and its problems
- Our Model - a new data structure
- Tree Update Language - commands and OS
- The Logic - syntax and semantics
- Local Hoare Reasoning - small axioms
- Concluding Remarks

Logical Environment

maps logical variables to their values

label variables

tree fragment variables

e: $(LVar_X \rightarrow_{fin} X) \times (LVar_C \rightarrow_{fin} C) \times (LVar_F \rightarrow_{fin} F)$

tree context variables

The Logic - Syntax

P_C ::= $P_C \Rightarrow P_C$ | false_C

| $\emptyset_C$ | α | $n[P_C]$ | $P_C \otimes P_C$

| c | B | @α

| $\exists v.P_C$ | $\exists lv.P_C$

P_F ::= $P_F \Rightarrow P_F$ | false_F

| $P_F * P_F$ | $P_F -\!\!* P_F$ | α®P_F | α–®P_F

| $\emptyset_F$ | $\alpha \leftarrow P_C$ |

| f | B |

| $\exists v.P_F$ | $\exists lv.P_F$ | $\text{И}\alpha.P_F$

The Logic - Syntax

P_C ::= $P_C \Rightarrow P_C$ | false_C
| $\emptyset_C$ | α | $n[P_C]$ | $P_C \otimes P_C$
| c | B | @α
| $\exists v.P_C$ | $\exists lv.P_C$

P_F ::= $P_F \Rightarrow P_F$ | false_F
| $P_F * P_F$ | $P_F -\!\!* P_F$ | α®P_F | α–®P_F
| $\emptyset_F$ | $\alpha \leftarrow P_C$ |
| f | B |
| $\exists v.P_F$ | $\exists lv.P_F$ | Иα.P_F

The Logic - Semantics

The Logic - Semantics

$e,s,c \models_c @\alpha \quad \Leftrightarrow \quad e(\alpha) \in fn(c)$

The Logic - Semantics

$e,s,c \vDash_C @\alpha \quad \Leftrightarrow \quad e(\alpha) \in fn(c)$

$e,s,f \vDash_F P_F * P_F' \quad \Leftrightarrow \quad \exists f_1, f_2.\ f \equiv f_1 + f_2$

$\qquad \wedge\ e,s,f_1 \vDash_F P_F \ \wedge\ e,s,f_2 \vDash_F P_F'$

The Logic - Semantics

$e,s,c \vDash_C @\alpha \quad \Leftrightarrow \quad e(\alpha) \in fn(c)$

$e,s,f \vDash_F P_F * P_F' \quad \Leftrightarrow \quad \exists f_1,f_2.\ f \equiv f_1 + f_2$
$\quad \wedge\ e,s,f_1 \vDash_F P_F \ \wedge\ e,s,f_2 \vDash_F P_F'$

$e,s,f \vDash_F P_F \mathrel{-\!\!*} P_F' \Leftrightarrow \quad \forall f'.\ e,s,f \vDash_F P_F \wedge (f + f')\!\downarrow$
$\quad \Rightarrow e,s,(f + f') \vDash_F P_F'$

The Logic - Semantics

$e,s,c \vDash_C @\alpha \quad \Leftrightarrow \quad e(\alpha) \in fn(c)$

$e,s,f \vDash_F P_F * P_F' \quad \Leftrightarrow \quad \exists f_1,f_2.\ f \equiv f_1 + f_2$
$\quad \wedge\ e,s,f_1 \vDash_F P_F \ \wedge\ e,s,f_2 \vDash_F P_F'$

$e,s,f \vDash_F P_F -\!\!* P_F' \quad \Leftrightarrow \quad \forall f'.\ e,s,f \vDash_F P_F \ \wedge\ (f + f')\downarrow$
$\quad \Rightarrow\ e,s,(f + f') \vDash_F P_F'$

$e,s,f \vDash_F \alpha \circledR P_F \quad \Leftrightarrow \quad \exists x,f'.\ e(\alpha)=x \ \wedge\ f \equiv (\nu x)(f')$
$\quad \wedge\ e,s,f' \vDash_F P_F$

The Logic - Semantics

$e,s,c \models_C @\alpha \quad \Leftrightarrow \quad e(\alpha) \in fn(c)$

$e,s,f \models_F P_F * P_F' \quad \Leftrightarrow \quad \exists f_1,f_2.\ f \equiv f_1 + f_2$
$\quad \wedge\ e,s,f_1 \models_F P_F \ \wedge\ e,s,f_2 \models_F P_F'$

$e,s,f \models_F P_F -\!\!* P_F' \quad \Leftrightarrow \quad \forall f'.\ e,s,f \models_F P_F \ \wedge\ (f + f')\downarrow$
$\quad \Rightarrow\ e,s,(f + f') \models_F P_F'$

$e,s,f \models_F \alpha ® P_F \quad \Leftrightarrow \quad \exists x,f'.\ e(\alpha)=x \ \wedge\ f \equiv (\nu x)(f')$
$\quad \wedge\ e,s,f' \models_F P_F$

$e,s,f \models_F \alpha - ® P_F \quad \Leftrightarrow \quad \exists x,f'.\ e(\alpha)=x \ \wedge\ f' \equiv (\nu x)(f)$
$\quad \wedge\ e,s,f' \models_F P_F$

The Logic - Semantics

$e,s,c \vDash_C @\alpha \iff e(\alpha) \in fn(c)$

$e,s,f \vDash_F P_F * P_F' \iff \exists f_1,f_2.\ f \equiv f_1 + f_2$
$\quad \wedge\ e,s,f_1 \vDash_F P_F \ \wedge\ e,s,f_2 \vDash_F P_F'$

$e,s,f \vDash_F P_F -\!\!* P_F' \iff \forall f'.\ e,s,f \vDash_F P_F \wedge (f + f')\!\downarrow$
$\quad \Rightarrow e,s,(f + f') \vDash_F P_F'$

$e,s,f \vDash_F \alpha \circledR P_F \iff \exists x,f'.\ e(\alpha)=x \wedge f \equiv (\nu x)(f')$
$\quad \wedge\ e,s,f' \vDash_F P_F$

$e,s,f \vDash_F \alpha -\!\circledR P_F \iff \exists x,f'.\ e(\alpha)=x \wedge f' \equiv (\nu x)(f)$
$\quad \wedge\ e,s,f' \vDash_F P_F$

$e,s,f \vDash_F \alpha \leftarrow P_C \iff \exists c,x.\ e(\alpha)=x \wedge f \equiv x \leftarrow c$
$\quad \wedge\ e,s,c \vDash_C P_C$

The Logic - Semantics

$e,s,c \models_C @\alpha \quad \Leftrightarrow \quad e(\alpha) \in fn(c)$

$e,s,f \models_F P_F * P_F' \quad \Leftrightarrow \quad \exists f_1,f_2.\ f \equiv f_1 + f_2$
$\qquad \wedge\ e,s,f_1 \models_F P_F \ \wedge\ e,s,f_2 \models_F P_F'$

$e,s,f \models_F P_F -\!\!* P_F' \quad \Leftrightarrow \quad \forall f'.\ e,s,f \models_F P_F \wedge (f + f')\!\downarrow$
$\qquad \Rightarrow\ e,s,(f + f') \models_F P_F'$

$e,s,f \models_F \alpha \circledR P_F \quad \Leftrightarrow \quad \exists x,f'.\ e(\alpha)=x \ \wedge\ f \equiv (\nu x)(f')$
$\qquad \wedge\ e,s,f' \models_F P_F$

$e,s,f \models_F \alpha -\!\circledR P_F \quad \Leftrightarrow \quad \exists x,f'.\ e(\alpha)=x \ \wedge\ f' \equiv (\nu x)(f)$
$\qquad \wedge\ e,s,f' \models_F P_F$

$e,s,f \models_F \alpha \leftarrow P_C \quad \Leftrightarrow \quad \exists c,x.\ e(\alpha)=x \ \wedge\ f \equiv x \leftarrow c$
$\qquad \wedge\ e,s,c \models_C P_C$

$e,s,f \models_F Ͷ\alpha.P_F \quad \Leftrightarrow \quad \exists x.\ x\#e,f \ \wedge\ e[\alpha \mapsto x],s,f \models_F P_F$

Derived Formulae

$$\mathrm{tree}(P_C) \triangleq P_C \wedge \neg\exists\alpha.@\alpha$$

$$n \triangleq n[\emptyset_C]$$

$$\bullet[P_C] \triangleq \exists m.m[P_C]$$

$$\Diamond P_F \triangleq \mathrm{true}_F * P_F$$

$$\mathsf{H}\alpha.P_B \triangleq \text{И}\alpha.\alpha \circledR P_B$$

Overview

- Append - the command and its problems
- Our Model - a new data structure
- Tree Update Language - commands and OS
- The Logic - syntax and semantics
- Local Hoare Reasoning - small axioms
- Concluding Remarks

Local Hoare Reasoning

fault avoiding partial correctness:

$$\{P_F\}\ C\ \{Q_F\} \Leftrightarrow \forall e,s,f.\ \text{free}(C) \cup \text{free}(P_F) \cup \text{free}(Q_F) \subseteq \text{dom}(s) \wedge e,s,f \vDash_F P_F \Rightarrow C,s,f \not\rightsquigarrow \text{fault} \wedge \forall s',f'.\ C,s,f \rightsquigarrow s'f' \Rightarrow e,s',f' \vDash_F Q_F$$

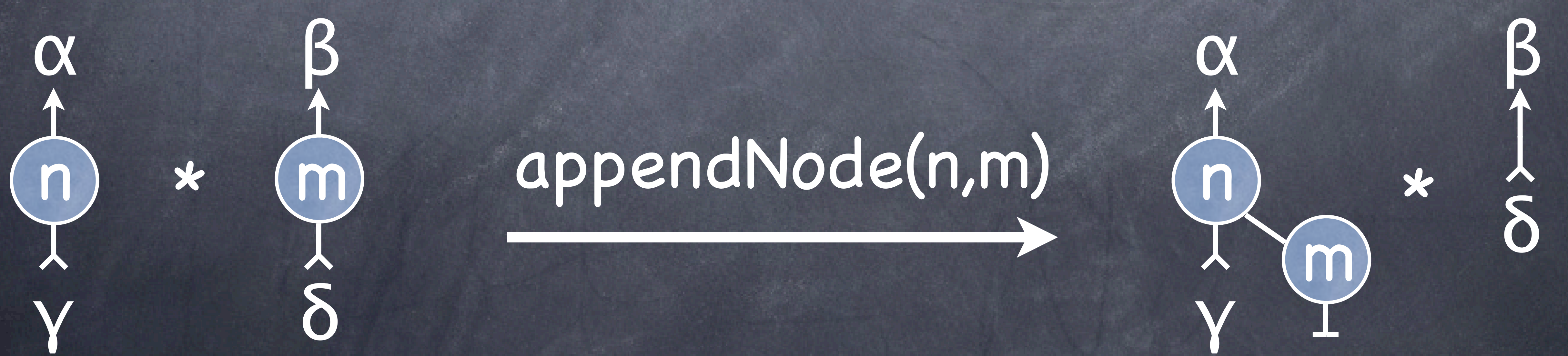
free(A) is the set of free program varibales in A

Small Axioms

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

appendNode(n,m)

$$\{\ \alpha \leftarrow n[\gamma \otimes m[\emptyset_C]] * \beta \leftarrow \delta\ \}$$

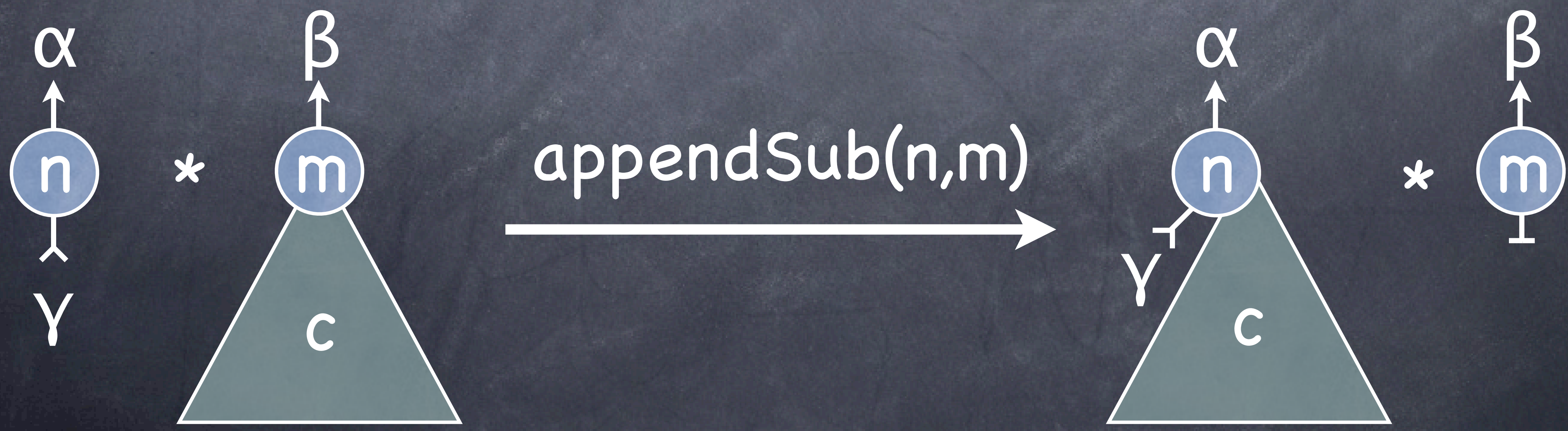


Small Axioms

$$\{ \alpha \leftarrow n[\gamma] * \beta \leftarrow m[tree(c)] \}$$

appendSub(n,m)

$$\{ \alpha \leftarrow n[\gamma \otimes tree(c)] * \beta \leftarrow m[\emptyset_C] \}$$



Frame Rules

Revelation Frame:

$$\frac{\{\ P_F\ \}\ \ C\ \ \{\ Q_F\ \}}{\{\ \alpha \circledR P_F\ \}\ \ C\ \ \{\ \alpha \circledR Q_F\ \}}$$

Separation Frame:

$$\frac{\{\ P_F\ \}\ \ C\ \ \{\ Q_F\ \}}{\{\ R_F * P_F\ \}\ \ C\ \ \{\ R_F * Q_F\ \}}\ \ \mathrm{mod}(C) \cup \mathrm{free}(R_F) = \{\ \}$$

Variable Quantification

Fresh Label Elimination:

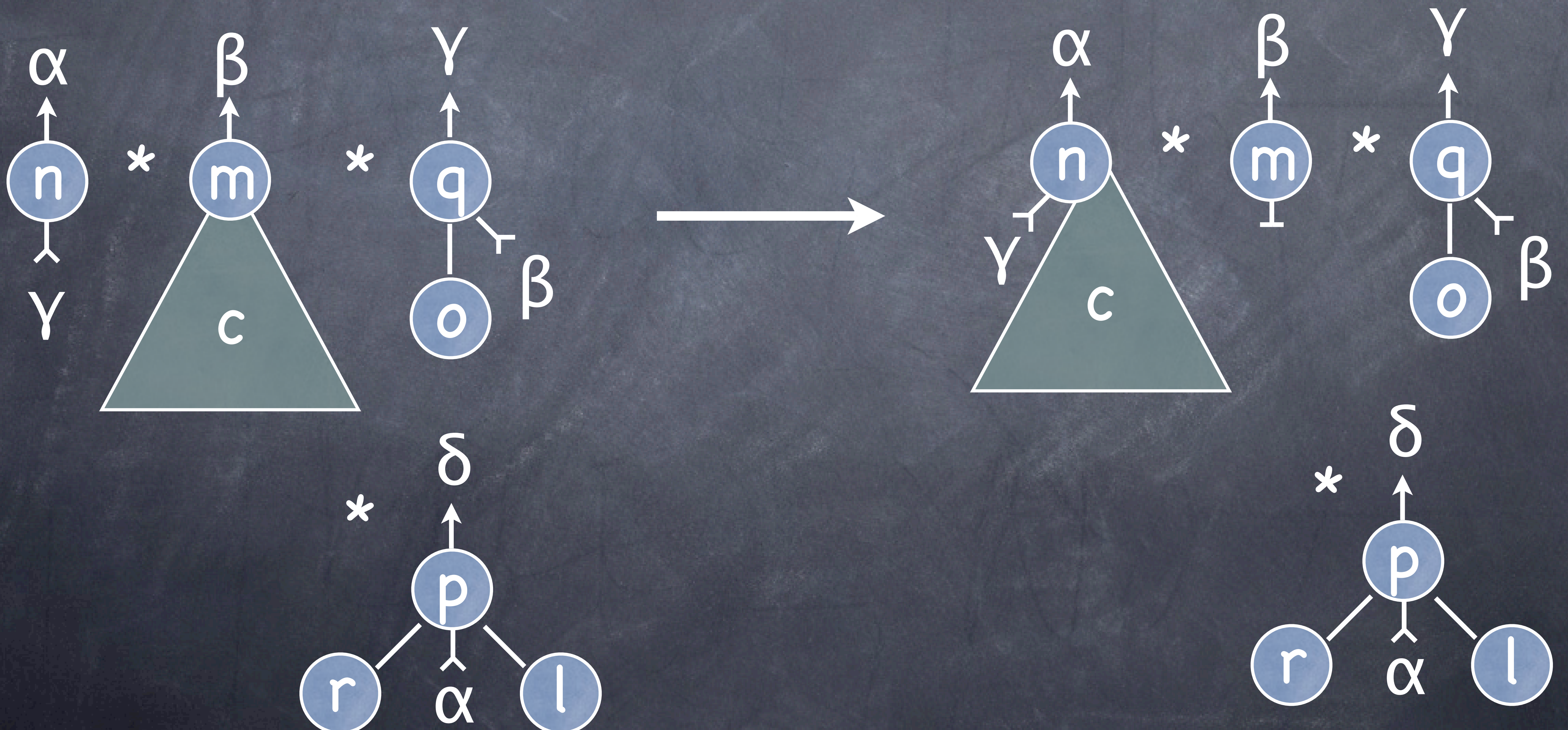
$$\frac{\{\ P_F\ \}\ \ C\ \ \{\ Q_F\ \}}{\{\ \text{И}\alpha.P_F\ \}\ \ C\ \ \{\ \text{И}\alpha.Q_F\ \}}$$

Auxiliary Variable Elimination:

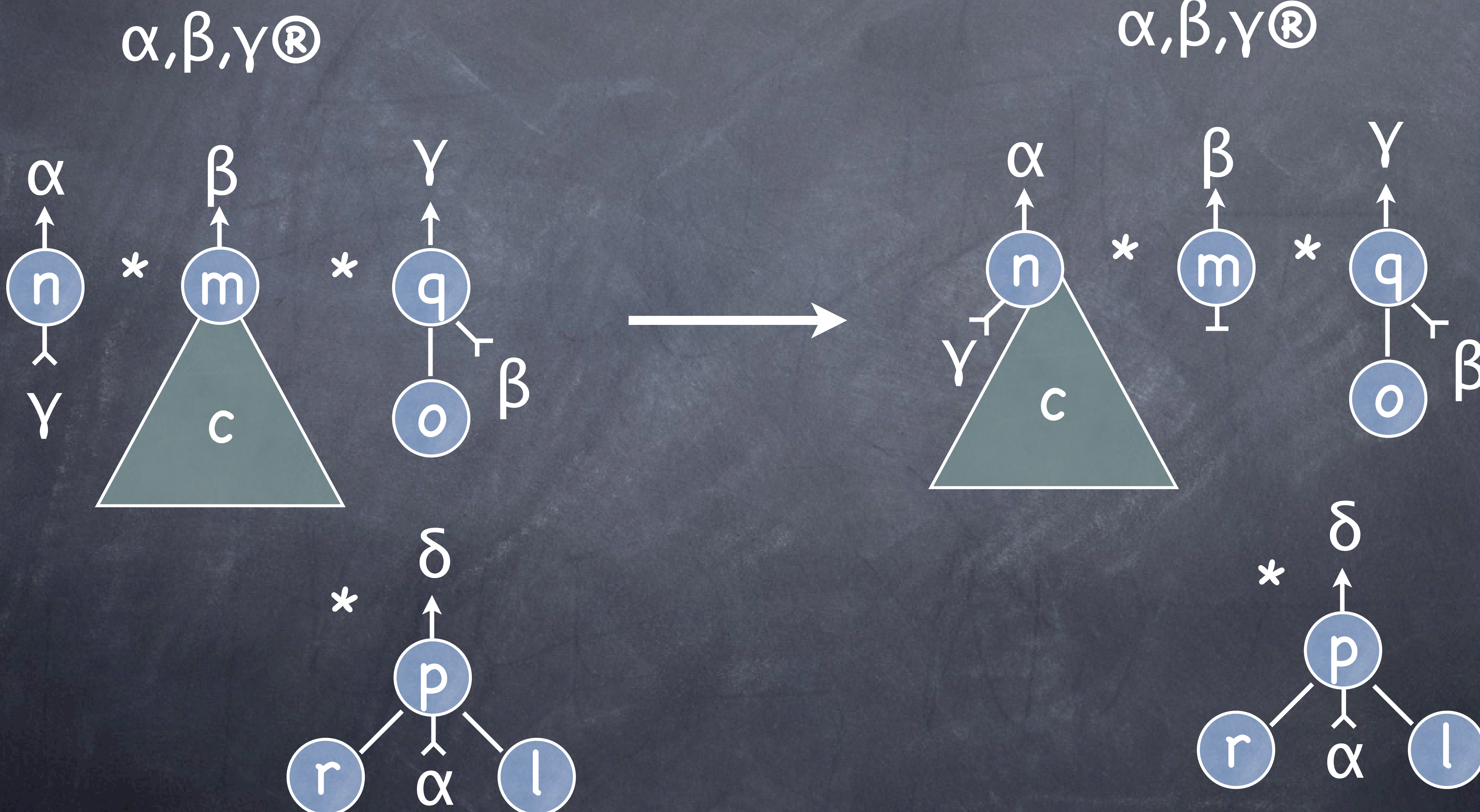
$$\frac{\{\ P_F\ \}\ \ C\ \ \{\ Q_F\ \}}{\{\ \exists n.P_F\ \}\ \ C\ \ \{\ \exists n.Q_F\ \}}\quad n \notin \mathrm{free}(C)$$

Example - Frame

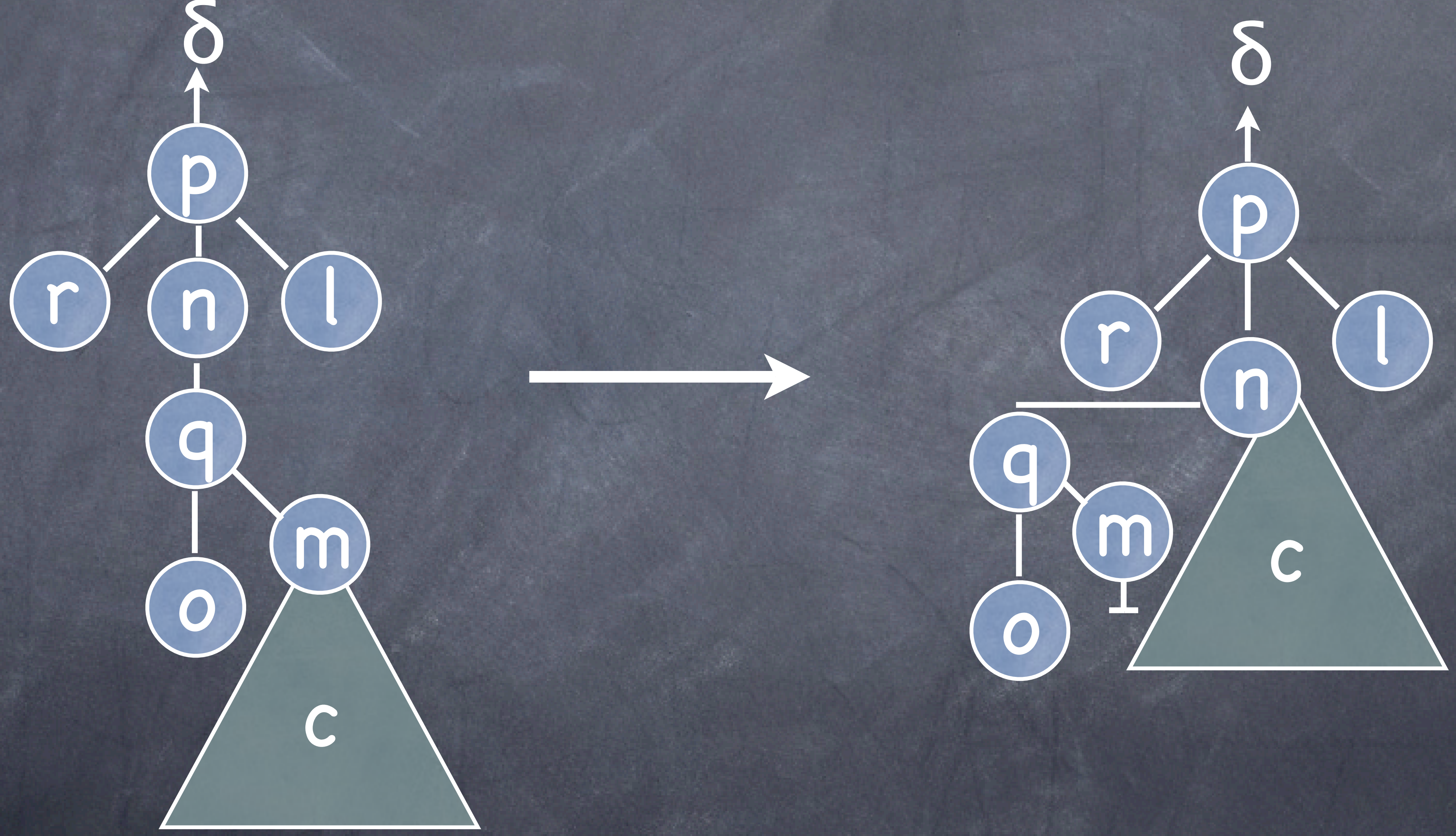
Example - Frame



Example - Frame



Example - Frame



Deriving Append's Axiom

```
append(n,m) ≜ insertRight(m,•[∅T]);
              t := getRight(m);
              appendSub(t,m);
              appendNode(n,m);
              appendSub(m,t);
              deleteNode(t);
```

Deriving Append's Axiom

insertRight(m,•[$\emptyset_T$]);

t := getRight(m);

appendSub(t,m);

appendNode(n,m);

appendSub(m,t);

deleteNode(t);

Deriving Append's Axiom

```
insertRight(m,•[ØT]);

t := getRight(m);

appendSub(t,m);

appendNode(n,m);

appendSub(m,t);

deleteNode(t);
```

Deriving Append's Axiom

$\{ \alpha \leftarrow n[\gamma] * \beta \leftarrow m[tree(c)] \}$

insertRight(m,•[$\emptyset_T$]);

t := getRight(m);

appendSub(t,m);

appendNode(n,m);

appendSub(m,t);

deleteNode(t);

Deriving Append's Axiom

{ δ®(α←n[γ] * β←m[δ] * δ←tree(c)) }

insertRight(m,•[$\emptyset_T$]);

t := getRight(m);

appendSub(t,m);

appendNode(n,m);

appendSub(m,t);

deleteNode(t);

Deriving Append's Axiom

$\{ \delta®(\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] * \delta \leftarrow tree(c)) \}$

insertRight(m,•[$\emptyset_T$]);

t := getRight(m);

appendSub(t,m);

appendNode(n,m);

appendSub(m,t);

deleteNode(t);

Deriving Append's Axiom

$\{ \delta®(\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] * \delta \leftarrow tree(c)) \}$

insertRight(m,•[$\emptyset_T$]);

$\{ \delta®(\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] \otimes \bullet[\emptyset_T] * \delta \leftarrow tree(c)) \}$

t := getRight(m);

appendSub(t,m);

appendNode(n,m);

appendSub(m,t);

deleteNode(t);

Deriving Append's Axiom

$\{ \delta®(\alpha{\leftarrow}n[\gamma] * \beta{\leftarrow}m[\delta] * \delta{\leftarrow}tree(c)) \}$

insertRight(m,•[$\varnothing_T$]);

$\{ \delta®(\alpha{\leftarrow}n[\gamma] * \beta{\leftarrow}m[\delta] \otimes \bullet[\varnothing_C] * \delta{\leftarrow}tree(c)) \}$

t := getRight(m);

appendSub(t,m);

appendNode(n,m);

appendSub(m,t);

deleteNode(t);

Deriving Append's Axiom

$\{ \delta ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] * \delta \leftarrow tree(c)) \}$

insertRight(m,•[$\emptyset_T$]);

$\{ \delta ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] \otimes \bullet[\emptyset_C] * \delta \leftarrow tree(c)) \}$

t := getRight(m);

$\{ \delta ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] \otimes t[\emptyset_C] * \delta \leftarrow tree(c)) \}$

appendSub(t,m);

appendNode(n,m);

appendSub(m,t);

deleteNode(t);

Deriving Append's Axiom

$\{\ \delta{®}(\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] * \delta \leftarrow tree(c))\ \}$

insertRight(m,•[$\emptyset_T$]);

$\{\ \delta{®}(\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] \otimes \bullet[\emptyset_C] * \delta \leftarrow tree(c))\ \}$

t := getRight(m);

$\{\ \delta{®}(\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] \otimes t[\emptyset_C] * \delta \leftarrow tree(c))\ \}$

appendSub(t,m);

appendNode(n,m);

appendSub(m,t);

deleteNode(t);

Deriving Append's Axiom

$\{ \delta ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] * \delta \leftarrow tree(c)) \}$

insertRight(m,•[$\emptyset_T$]);

$\{ \delta ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] \otimes \bullet[\emptyset_C] * \delta \leftarrow tree(c)) \}$

t := getRight(m);

$\{ \alpha \leftarrow n[\gamma] * \beta \leftarrow m[tree(c)] \otimes t[\emptyset_C] \}$

appendSub(t,m);

appendNode(n,m);

appendSub(m,t);

deleteNode(t);

Deriving Append's Axiom

$\{ \delta ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] * \delta \leftarrow tree(c)) \}$

insertRight(m,•[$\emptyset_T$]);

$\{ \delta ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] \otimes \bullet[\emptyset_C] * \delta \leftarrow tree(c)) \}$

t := getRight(m);

$\{ \omega,\lambda ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[tree(c)] * \lambda \leftarrow t[\emptyset_C]) \}$

appendSub(t,m);

appendNode(n,m);

appendSub(m,t);

deleteNode(t);

Deriving Append's Axiom

{ δ®($\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] * \delta \leftarrow tree(c)$) }

insertRight(m,•[$\emptyset_T$]);

{ δ®($\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] \otimes \bullet[\emptyset_C] * \delta \leftarrow tree(c)$) }

t := getRight(m);

{ ω,λ®($\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[tree(c)] * \lambda \leftarrow t[\emptyset_C]$) }

appendSub(t,m);

appendNode(n,m);

appendSub(m,t);

deleteNode(t);

Deriving Append's Axiom

{ δ®(α←n[γ] * β←m[δ] * δ←tree(c)) }

insertRight(m,•[$\emptyset_T$]);

{ δ®(α←n[γ] * β←m[δ] ⊗ •[$\emptyset_C$] * δ←tree(c)) }

t := getRight(m);

{ ω,λ®(α←n[γ] * β←ω⊗λ * ω←m[tree(c)] * λ←t[$\emptyset_C$]) }

appendSub(t,m);

{ ω,λ®(α←n[γ] * β←ω⊗λ * ω←m[$\emptyset_C$] * λ←t[tree(c)]) }

appendNode(n,m);

appendSub(m,t);

deleteNode(t);

Deriving Append's Axiom

$\{\ \delta\circledR(\alpha\leftarrow n[\gamma] * \beta\leftarrow m[\delta] * \delta\leftarrow tree(c))\ \}$

insertRight(m,•[$\emptyset_T$]);

$\{\ \delta\circledR(\alpha\leftarrow n[\gamma] * \beta\leftarrow m[\delta] \otimes \bullet[\emptyset_C] * \delta\leftarrow tree(c))\ \}$

t := getRight(m);

$\{\ \omega,\lambda\circledR(\alpha\leftarrow n[\gamma] * \beta\leftarrow \omega\otimes\lambda * \omega\leftarrow m[tree(c)] * \lambda\leftarrow t[\emptyset_C])\ \}$

appendSub(t,m);

$\{\ \omega,\lambda\circledR(\alpha\leftarrow n[\gamma] * \beta\leftarrow \omega\otimes\lambda * \omega\leftarrow m[\emptyset_C] * \lambda\leftarrow t[tree(c)])\ \}$

appendNode(n,m);

appendSub(m,t);

deleteNode(t);

Deriving Append's Axiom

$\{\ \delta ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] * \delta \leftarrow tree(c))\ \}$

insertRight(m,$\bullet[\emptyset_T]$);

$\{\ \delta ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] \otimes \bullet[\emptyset_C] * \delta \leftarrow tree(c))\ \}$

t := getRight(m);

$\{\ \omega,\lambda ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[tree(c)] * \lambda \leftarrow t[\emptyset_C])\ \}$

appendSub(t,m);

$\{\ \omega,\lambda ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[\emptyset_C] * \lambda \leftarrow t[tree(c)])\ \}$

appendNode(n,m);

$\{\ \omega,\lambda ® (\alpha \leftarrow n[\gamma \otimes m[\emptyset_C]] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow \emptyset_C * \lambda \leftarrow t[tree(c)])\ \}$

appendSub(m,t);

deleteNode(t);

Deriving Append's Axiom

{ δ®(α←n[γ] * β←m[δ] * δ←tree(c)) }

insertRight(m,•[$\emptyset_T$]);

{ δ®(α←n[γ] * β←m[δ] ⊗ •[$\emptyset_C$] * δ←tree(c)) }

t := getRight(m);

{ ω,λ®(α←n[γ] * β←ω⊗λ * ω←m[tree(c)] * λ←t[$\emptyset_C$]) }

appendSub(t,m);

{ ω,λ®(α←n[γ] * β←ω⊗λ * ω←m[$\emptyset_C$] * λ←t[tree(c)]) }

appendNode(n,m);

{ ω,λ®(α←n[γ⊗m[$\emptyset_C$]] * β←ω⊗λ * ω←$\emptyset_C$ * λ←t[tree(c)]) }

appendSub(m,t);

deleteNode(t);

Deriving Append's Axiom

$\{ \delta®(\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] * \delta \leftarrow tree(c)) \}$

insertRight(m,•[$\emptyset_T$]);

$\{ \delta®(\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] \otimes \bullet[\emptyset_C] * \delta \leftarrow tree(c)) \}$

t := getRight(m);

$\{ \omega,\lambda®(\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[tree(c)] * \lambda \leftarrow t[\emptyset_C]) \}$

appendSub(t,m);

$\{ \omega,\lambda®(\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[\emptyset_C] * \lambda \leftarrow t[tree(c)]) \}$

appendNode(n,m);

$\{ (\alpha \leftarrow n[\gamma \otimes m[\emptyset_C]] * \beta \leftarrow t[tree(c)]) \}$

appendSub(m,t);

deleteNode(t);

Deriving Append's Axiom

$\{ \delta ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] * \delta \leftarrow tree(c)) \}$

insertRight(m,•[$\emptyset_T$]);

$\{ \delta ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] \otimes \bullet[\emptyset_C] * \delta \leftarrow tree(c)) \}$

t := getRight(m);

$\{ \omega,\lambda ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[tree(c)] * \lambda \leftarrow t[\emptyset_C]) \}$

appendSub(t,m);

$\{ \omega,\lambda ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[\emptyset_C] * \lambda \leftarrow t[tree(c)]) \}$

appendNode(n,m);

$\{ \omega ® (\alpha \leftarrow n[\gamma \otimes \omega] * \omega \leftarrow m[\emptyset_C] * \beta \leftarrow t[tree(c)]) \}$

appendSub(m,t);

deleteNode(t);

Deriving Append's Axiom

{ $\delta ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] * \delta \leftarrow tree(c))$ }

insertRight(m,•[$\varnothing_T$]);

{ $\delta ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] \otimes \bullet[\varnothing_C] * \delta \leftarrow tree(c))$ }

t := getRight(m);

{ $\omega,\lambda ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[tree(c)] * \lambda \leftarrow t[\varnothing_C])$ }

appendSub(t,m);

{ $\omega,\lambda ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[\varnothing_C] * \lambda \leftarrow t[tree(c)])$ }

appendNode(n,m);

{ $\omega ® (\alpha \leftarrow n[\gamma \otimes \omega] * \omega \leftarrow m[\varnothing_C] * \beta \leftarrow t[tree(c)])$ }

appendSub(m,t);

deleteNode(t);

Deriving Append's Axiom

$\{ \delta ®(\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] * \delta \leftarrow tree(c)) \}$

insertRight(m,•[$\emptyset_T$]);

$\{ \delta ®(\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] \otimes \bullet[\emptyset_C] * \delta \leftarrow tree(c)) \}$

t := getRight(m);

$\{ \omega,\lambda ®(\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[tree(c)] * \lambda \leftarrow t[\emptyset_C]) \}$

appendSub(t,m);

$\{ \omega,\lambda ®(\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[\emptyset_C] * \lambda \leftarrow t[tree(c)]) \}$

appendNode(n,m);

$\{ \omega ®(\alpha \leftarrow n[\gamma \otimes \omega] * \omega \leftarrow m[\emptyset_C] * \beta \leftarrow t[tree(c)]) \}$

appendSub(m,t);

$\{ \omega ®(\alpha \leftarrow n[\gamma \otimes \omega] * \omega \leftarrow m[tree(c)] * \beta \leftarrow t[\emptyset_C]) \}$

deleteNode(t);

Deriving Append's Axiom

$\{ \delta®(\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] * \delta \leftarrow tree(c)) \}$

insertRight(m,•[$\emptyset_T$]);

$\{ \delta®(\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] \otimes \bullet[\emptyset_C] * \delta \leftarrow tree(c)) \}$

t := getRight(m);

$\{ \omega,\lambda®(\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[tree(c)] * \lambda \leftarrow t[\emptyset_C]) \}$

appendSub(t,m);

$\{ \omega,\lambda®(\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[\emptyset_C] * \lambda \leftarrow t[tree(c)]) \}$

appendNode(n,m);

$\{ \omega®(\alpha \leftarrow n[\gamma \otimes \omega] * \omega \leftarrow m[\emptyset_C] * \beta \leftarrow t[tree(c)]) \}$

appendSub(m,t);

$\{ \omega®(\alpha \leftarrow n[\gamma \otimes \omega] * \omega \leftarrow m[tree(c)] * \beta \leftarrow t[\emptyset_C]) \}$

deleteNode(t);

Deriving Append's Axiom

$\{ \delta ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] * \delta \leftarrow tree(c)) \}$

insertRight(m,•[$\emptyset_T$]);

$\{ \delta ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] \otimes \bullet[\emptyset_C] * \delta \leftarrow tree(c)) \}$

t := getRight(m);

$\{ \omega,\lambda ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[tree(c)] * \lambda \leftarrow t[\emptyset_C]) \}$

appendSub(t,m);

$\{ \omega,\lambda ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[\emptyset_C] * \lambda \leftarrow t[tree(c)]) \}$

appendNode(n,m);

$\{ \omega ® (\alpha \leftarrow n[\gamma \otimes \omega] * \omega \leftarrow m[\emptyset_C] * \beta \leftarrow t[tree(c)]) \}$

appendSub(m,t);

$\{ \alpha \leftarrow n[\gamma \otimes m[tree(c)]] * \beta \leftarrow t[\emptyset_C] \}$

deleteNode(t);

Deriving Append's Axiom

$\{\ \delta ®(\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] * \delta \leftarrow tree(c))\ \}$

insertRight(m,•[$\emptyset_T$]);

$\{\ \delta ®(\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] \otimes \bullet[\emptyset_C] * \delta \leftarrow tree(c))\ \}$

t := getRight(m);

$\{\ \omega,\lambda ®(\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[tree(c)] * \lambda \leftarrow t[\emptyset_C])\ \}$

appendSub(t,m);

$\{\ \omega,\lambda ®(\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[\emptyset_C] * \lambda \leftarrow t[tree(c)])\ \}$

appendNode(n,m);

$\{\ \omega ®(\alpha \leftarrow n[\gamma \otimes \omega] * \omega \leftarrow m[\emptyset_C] * \beta \leftarrow t[tree(c)])\ \}$

appendSub(m,t);

$\{\ \alpha \leftarrow n[\gamma \otimes m[tree(c)]] * \beta \leftarrow t[\emptyset_C]\ \}$

deleteNode(t);

Deriving Append's Axiom

$\{ \delta ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] * \delta \leftarrow tree(c)) \}$

insertRight(m,•[$\emptyset_T$]);

$\{ \delta ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] \otimes \bullet[\emptyset_C] * \delta \leftarrow tree(c)) \}$

t := getRight(m);

$\{ \omega,\lambda ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[tree(c)] * \lambda \leftarrow t[\emptyset_C]) \}$

appendSub(t,m);

$\{ \omega,\lambda ® (\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[\emptyset_C] * \lambda \leftarrow t[tree(c)]) \}$

appendNode(n,m);

$\{ \omega ® (\alpha \leftarrow n[\gamma \otimes \omega] * \omega \leftarrow m[\emptyset_C] * \beta \leftarrow t[tree(c)]) \}$

appendSub(m,t);

$\{ \alpha \leftarrow n[\gamma \otimes m[tree(c)]] * \beta \leftarrow t[\emptyset_C] \}$

deleteNode(t);

$\{ \alpha \leftarrow n[\gamma \otimes m[tree(c)]] * \beta \leftarrow \emptyset_C \}$

Deriving Append's Axiom

$\{\ \delta \circledR (\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] * \delta \leftarrow tree(c))\ \}$

insertRight(m,•[$\emptyset_T$]);

$\{\ \delta \circledR (\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] \otimes \bullet[\emptyset_C] * \delta \leftarrow tree(c))\ \}$

t := getRight(m);

$\{\ \omega,\lambda \circledR (\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[tree(c)] * \lambda \leftarrow t[\emptyset_C])\ \}$

appendSub(t,m);

$\{\ \omega,\lambda \circledR (\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[\emptyset_C] * \lambda \leftarrow t[tree(c)])\ \}$

appendNode(n,m);

$\{\ \omega \circledR (\alpha \leftarrow n[\gamma \otimes \omega] * \omega \leftarrow m[\emptyset_C] * \beta \leftarrow t[tree(c)])\ \}$

appendSub(m,t);

$\{\ \alpha \leftarrow n[\gamma \otimes m[tree(c)]] * \beta \leftarrow t[\emptyset_C]\ \}$

deleteNode(t);

$\{\ \alpha \leftarrow n[\gamma \otimes m[tree(c)]] * \beta \leftarrow \emptyset_C\ \}$

Deriving Append's Axiom

$\{\ \alpha \leftarrow n[\gamma] * \beta \leftarrow m[tree(c)]\ \}$

insertRight(m,•[$\emptyset_T$]);

$\{\ \delta ®(\alpha \leftarrow n[\gamma] * \beta \leftarrow m[\delta] \otimes \bullet[\emptyset_C] * \delta \leftarrow tree(c))\ \}$

t := getRight(m);

$\{\ \omega,\lambda ®(\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[tree(c)] * \lambda \leftarrow t[\emptyset_C])\ \}$

appendSub(t,m);

$\{\ \omega,\lambda ®(\alpha \leftarrow n[\gamma] * \beta \leftarrow \omega \otimes \lambda * \omega \leftarrow m[\emptyset_C] * \lambda \leftarrow t[tree(c)])\ \}$

appendNode(n,m);

$\{\ \omega ®(\alpha \leftarrow n[\gamma \otimes \omega] * \omega \leftarrow m[\emptyset_C] * \beta \leftarrow t[tree(c)])\ \}$

appendSub(m,t);

$\{\ \alpha \leftarrow n[\gamma \otimes m[tree(c)]] * \beta \leftarrow t[\emptyset_C]\ \}$

deleteNode(t);

$\{\ \alpha \leftarrow n[\gamma \otimes m[tree(c)]] * \beta \leftarrow \emptyset_C\ \}$

Concluding Remarks

- We can now provide Small Axioms for complex tree update commands such as append.
- Our reasoning system now works on arbitrary pieces of tree, specifically including disjoint tree fragments.
- The fragment level of reasoning is model independent, so we can 'plug-and play' with other data structures (e.g. lists)
- Our finer grain of model allows us to think about concurrent tree update. We can make use of rules from Separation Logic such as the Parallel Rule and Resource rules of disjoint concurrency
- With this new way of breaking up data structures we can simplify the axioms and notation of our formal DOM specification.
- We also want to carry out some kind of analysis into how footprint and specification sizes match up.
- But first...this stuff needs a name!