Incremental Bidirectional Typing via Order Maintenance

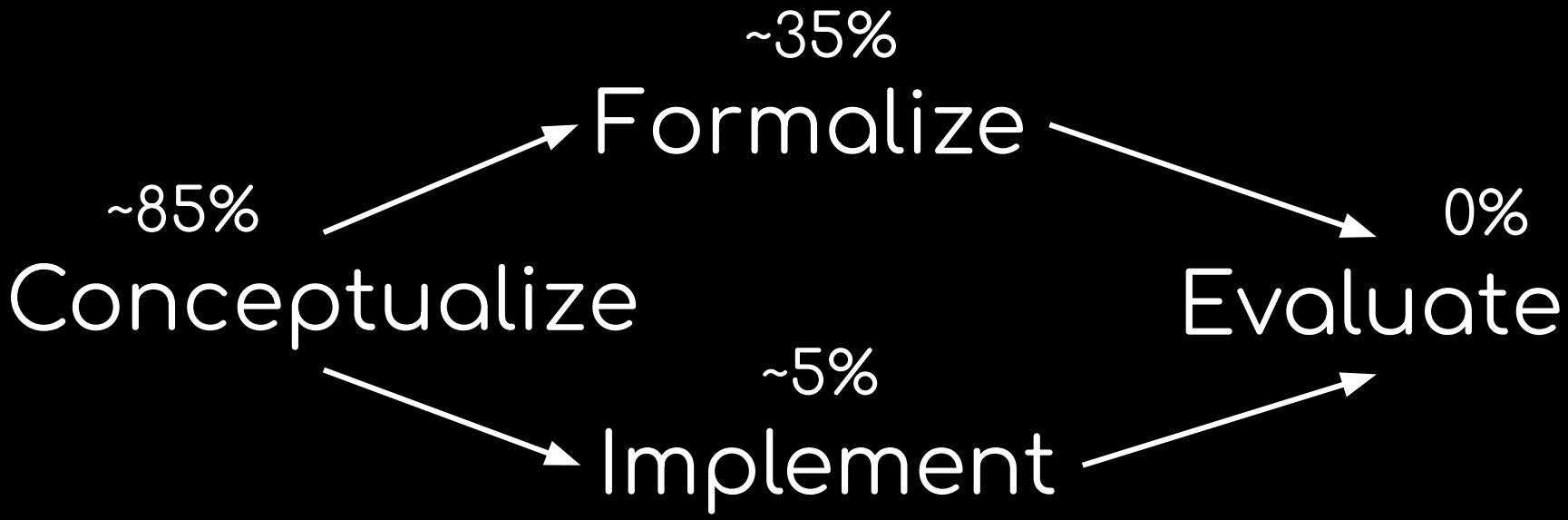
Thomas J. Porter[1] Marisa Kirisame[2] Liam Mulcahy[1]

Pavel Panchekha[2] Cyrus Omar[1]

[1] Future of Programming Lab, University of Michigan

[2] University of Utah

Work in progress:



Goal: *Live* programming

Subgoal: incremental type checking

Prior Work

Language implementations!

Adaptive FP: Acar et al, ‘02

Datalog: Pacak et al, ‘20 & Szabó et al, ‘16

Task engine: Wachsmuth et al, ‘13

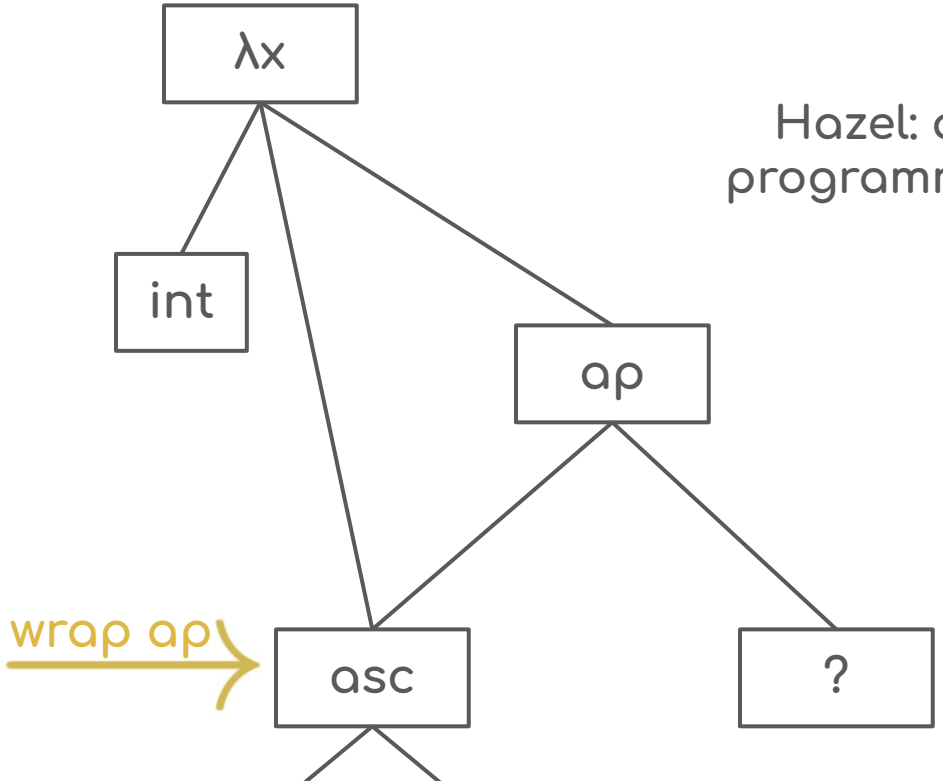
Memoization: Busi et al, ‘19

Forthcoming browser work

Structural Edits

Hazel: a live functional programming environment

λx : int. (... : ...)(?)



Non-assumptions

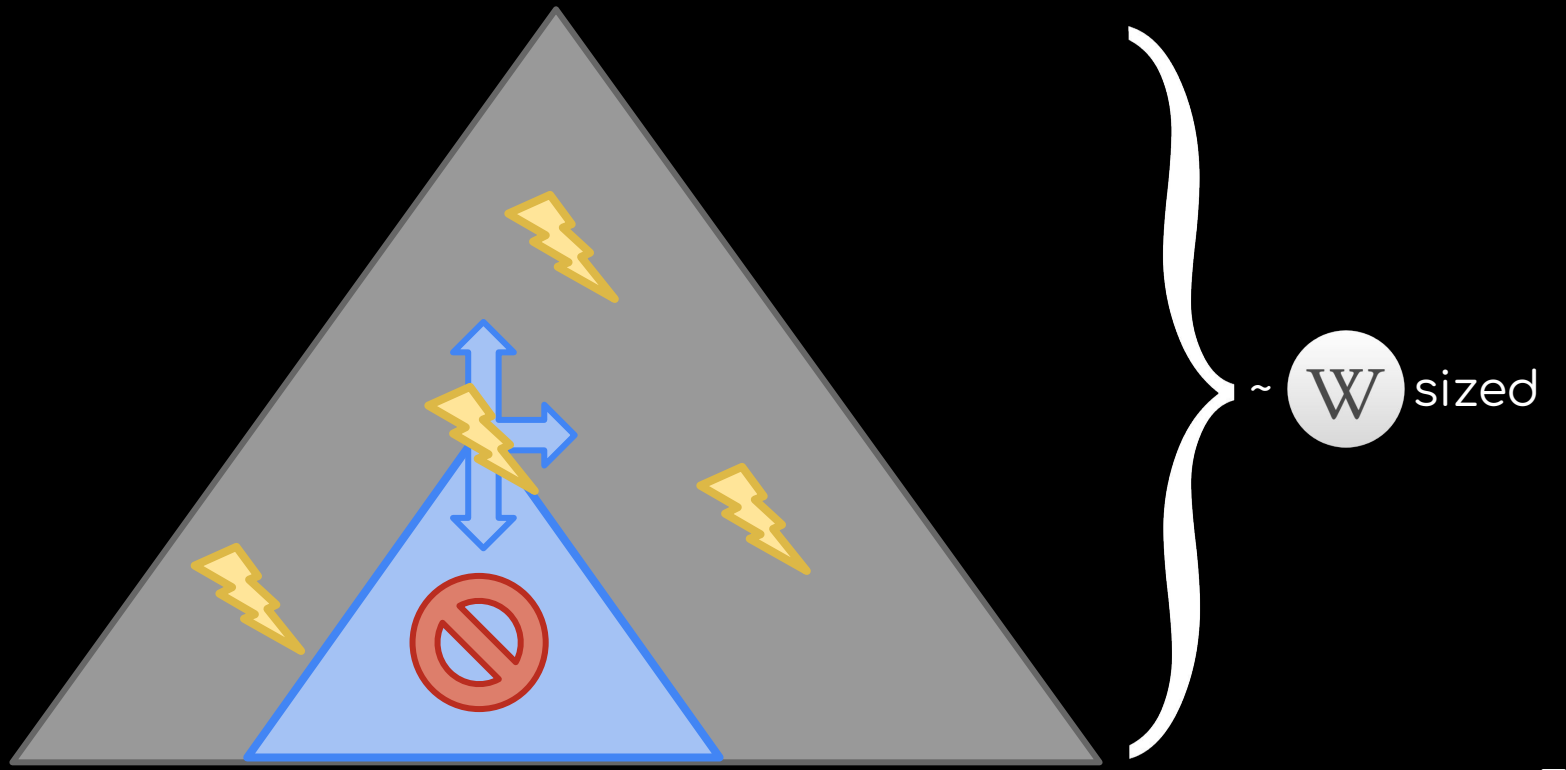
No cursor movement modeled (→ collaborativity)

No restrictions on program **structure** assumed

(modules, top level definitions)

No program **size** assumed

Vision: Live, Massively Collaborative Coding



One Year Ago...

Total Type Error Localization and Recovery with Holes

ERIC ZHAO, University of Michigan, USA
RAEF MAROOF, University of Michigan, USA
ANAND DUKKIPATI, University of Michigan, USA
ANDREW BLINN, University of Michigan, USA
ZHIYI PAN, University of Michigan, USA
CYRUS OMAR, University of Michigan, USA

THE MARKED LAMBDA CALCULUS

(in 60 seconds)

MKSAp2

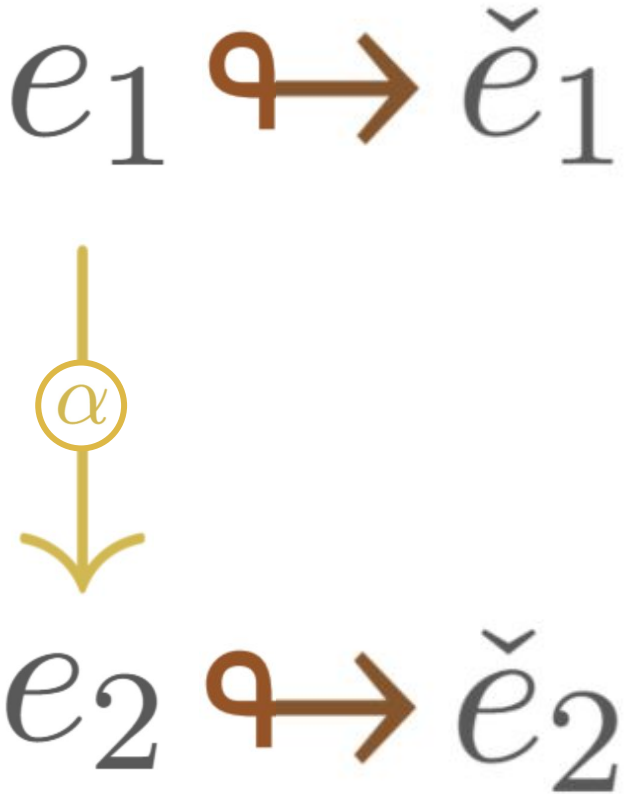
$$\frac{\Gamma \vdash e_1 \rightsquigarrow \check{e}_1 \Rightarrow \tau \qquad \tau \blacktriangleright_{\not\to} \qquad \Gamma \vdash e_2 \rightsquigarrow \check{e}_2 \Leftarrow ?}{\Gamma \vdash e_1\ e_2 \rightsquigarrow (\!|\check{e}_1|\!)^{\Rightarrow}_{\blacktriangleright_{\not\to}}\ \check{e}_2 \Rightarrow ?}$$

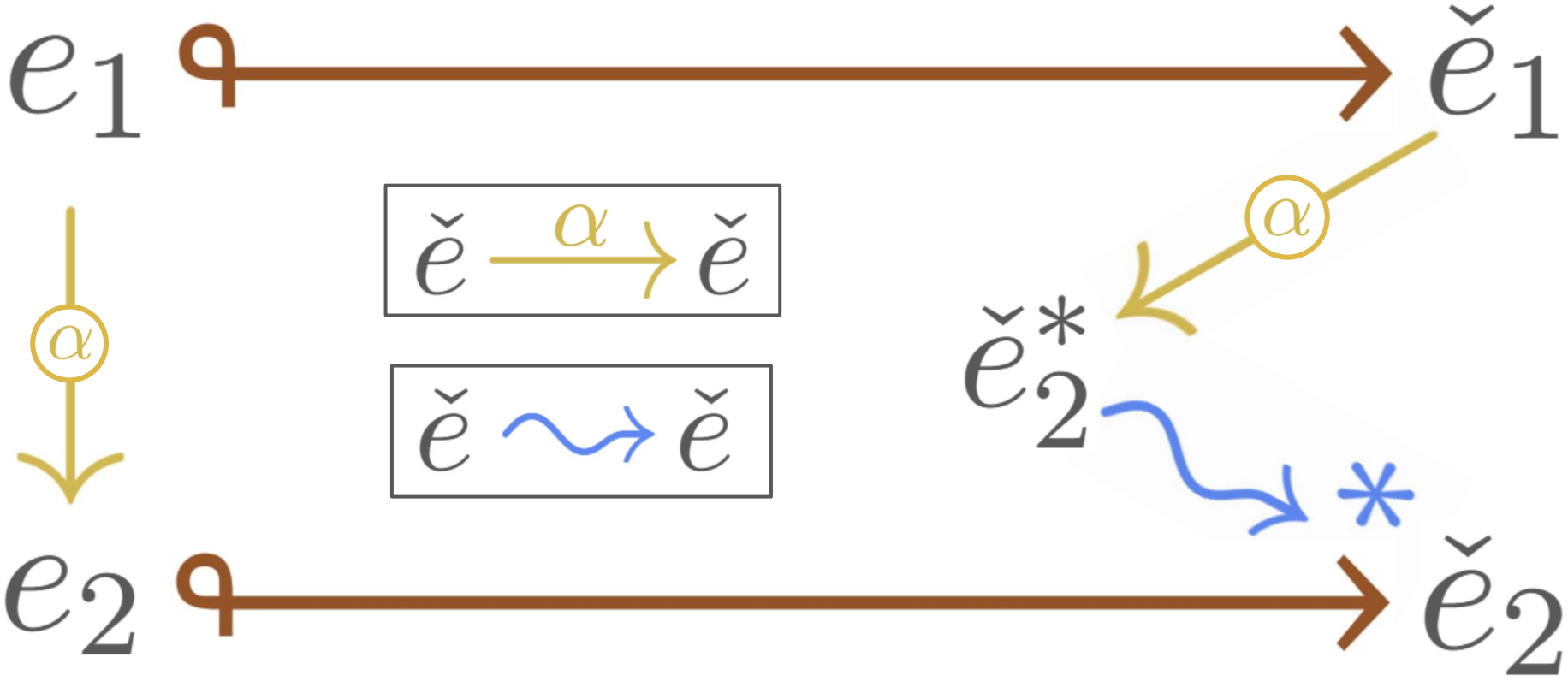
Theorem 2.1 (Marking Totality).

(1) For all Γ *and* e*, there exist* $\check{e}$ *and* τ *such that* $\Gamma \vdash e \rightsquigarrow \check{e} \Rightarrow \tau$ *and* $\Gamma \vdash_M \check{e} \Rightarrow \tau$.

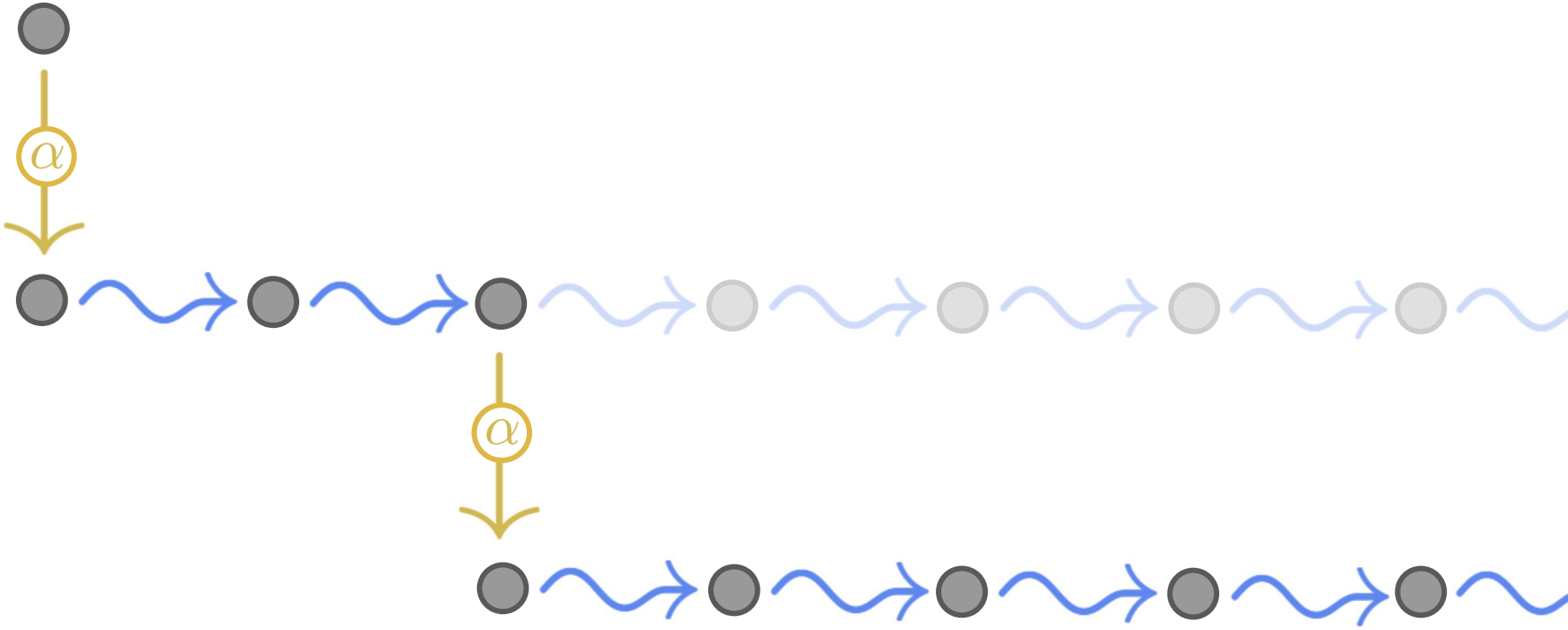
(2) For all Γ*,* e*, and* τ*, there exists* $\check{e}$ *such that* $\Gamma \vdash e \rightsquigarrow \check{e} \Leftarrow \tau$ *and* $\Gamma \vdash_M \check{e} \Leftarrow \tau$.

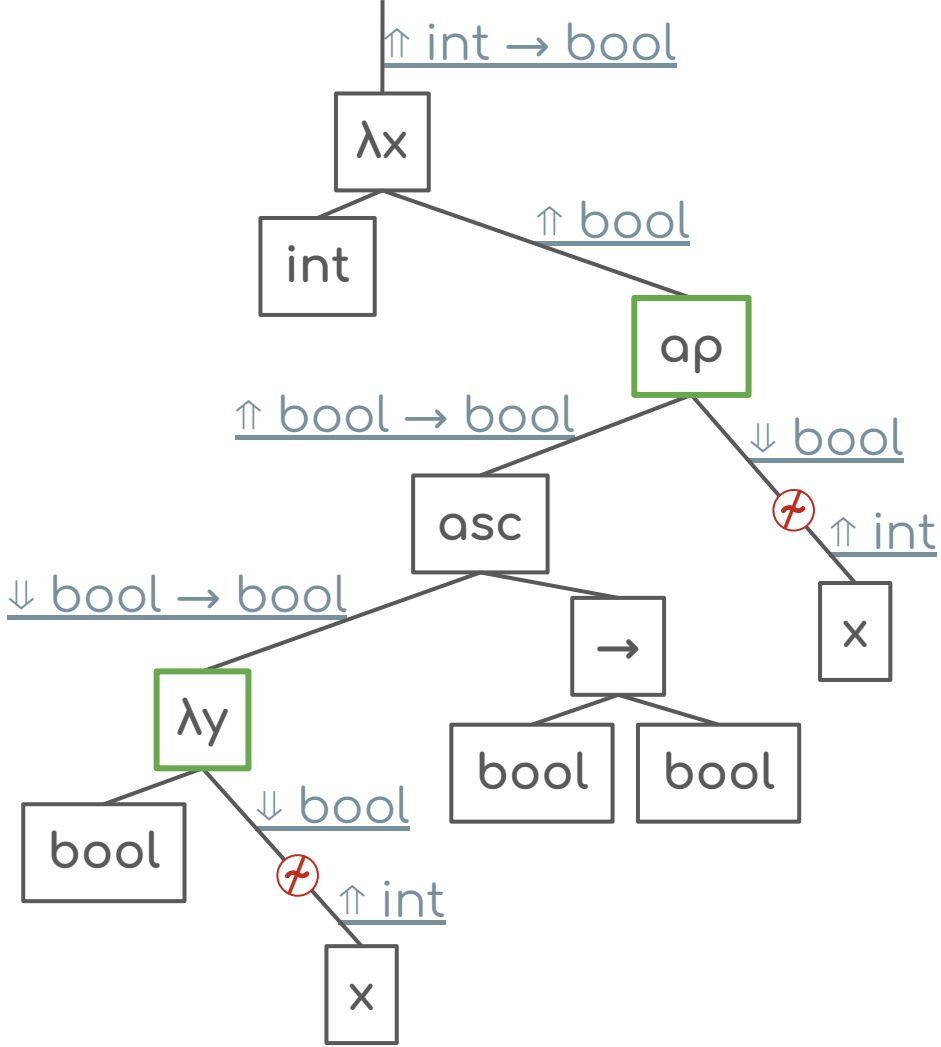
That is, we may mark *any* syntactically well-formed program in any context, resulting in a *well-typed* marked program.

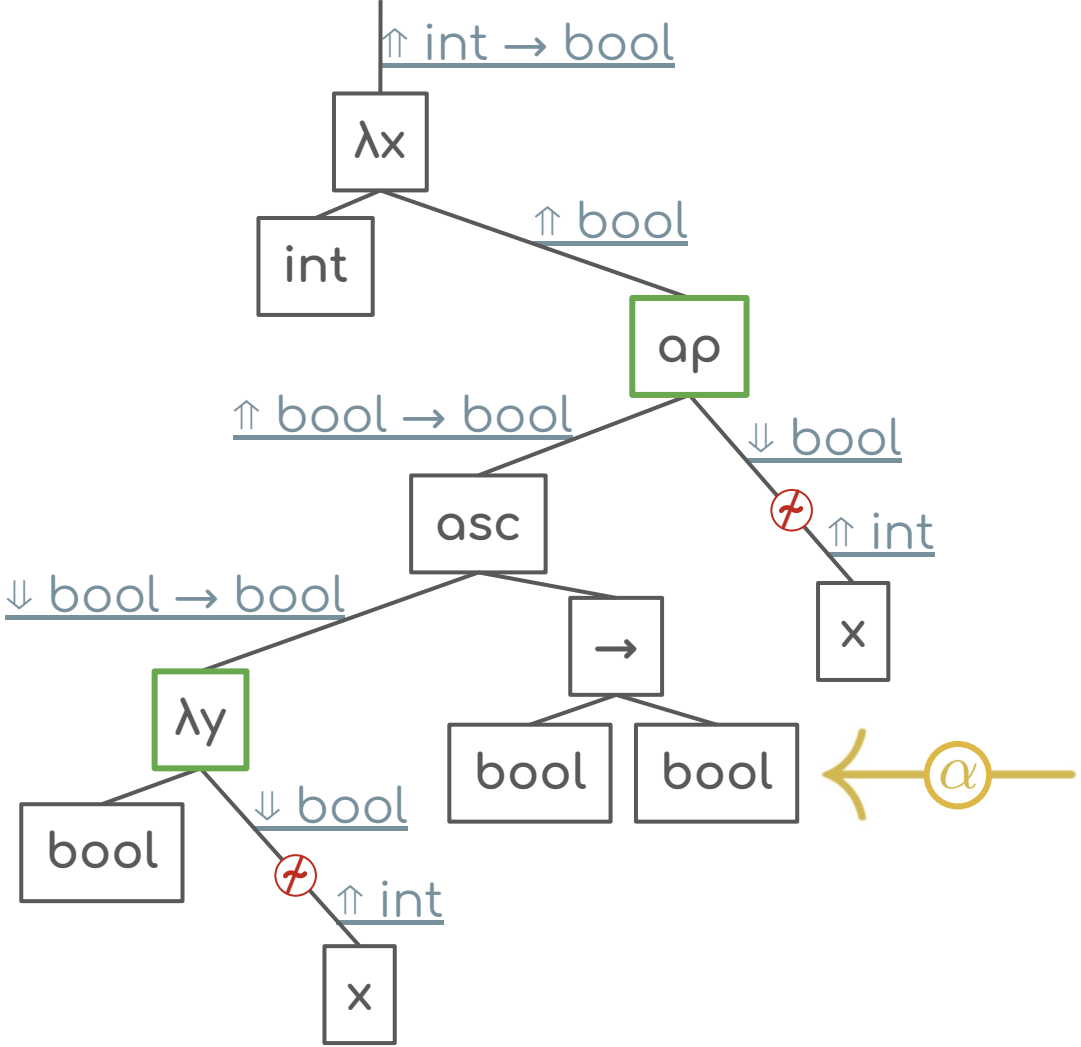

$e_1 \rightsquigarrow \check{e}_1$
α
$e_2 \rightsquigarrow \check{e}_2$

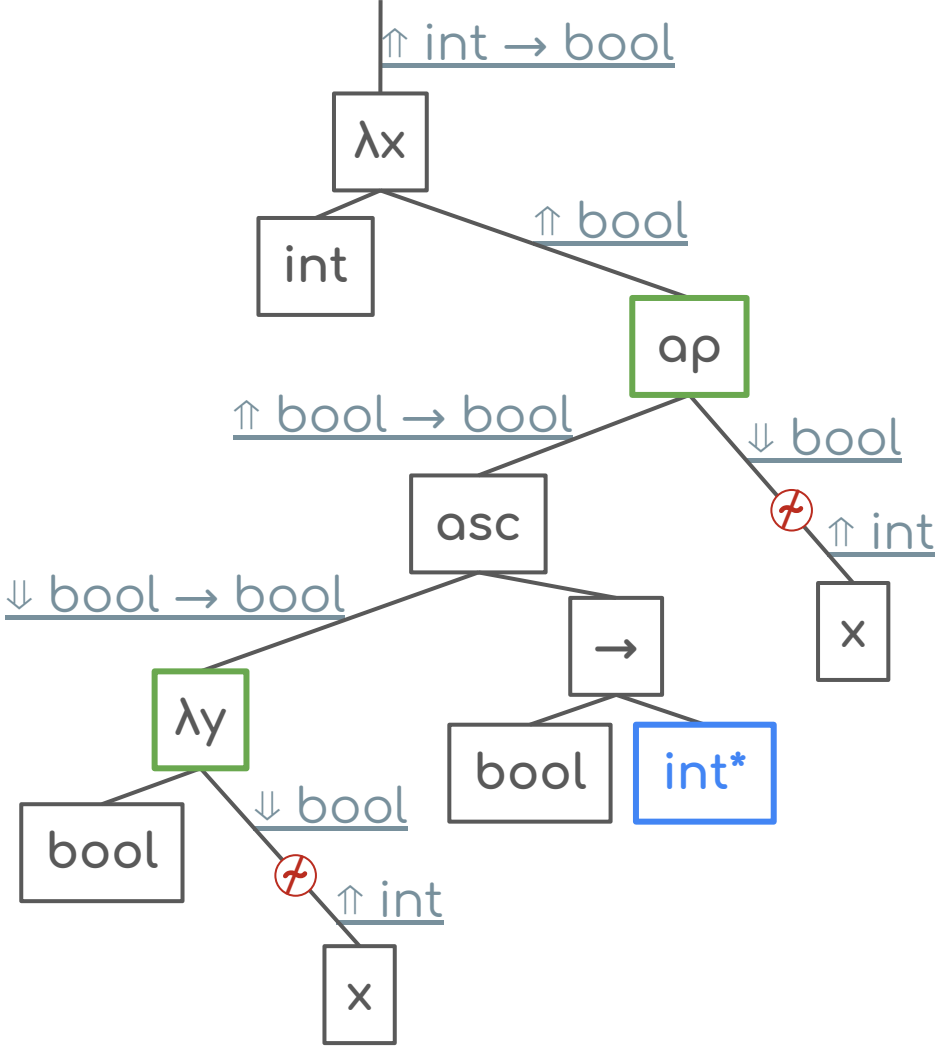

e_1
$\check{e}_1$
α
$\check{e} \xrightarrow{\alpha} \check{e}$
$\check{e} \rightsquigarrow \check{e}$
$\check{e}_2^*$
$*$
e_2
$\check{e}_2$

Interleaving Steps



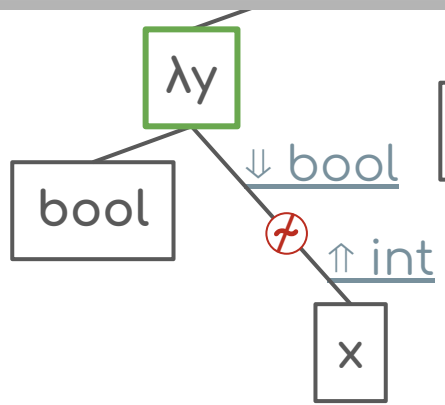

⇑ int → bool
λx
int
⇑ bool
ap
⇑ bool → bool
⇓ bool
asc
⇑ int
⇓ bool → bool
→
x
λy
bool
bool
bool
⇓ bool
⇑ int
x

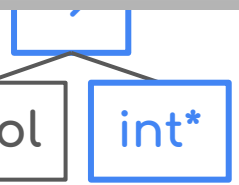

⇑ int → bool
λx
int
⇑ bool
ap
⇑ bool → bool
⇓ bool
asc
≠
⇑ int
⇓ bool → bool
→
x
λy
bool
bool
α
bool
⇓ bool
≠
⇑ int
x


⇑ int → bool
λx
int
⇑ bool
ap
⇑ bool → bool
⇓ bool
asc
≠
⇑ int
x
⇓ bool → bool
→
λy
bool
int*
bool
⇓ bool
≠
⇑ int
x

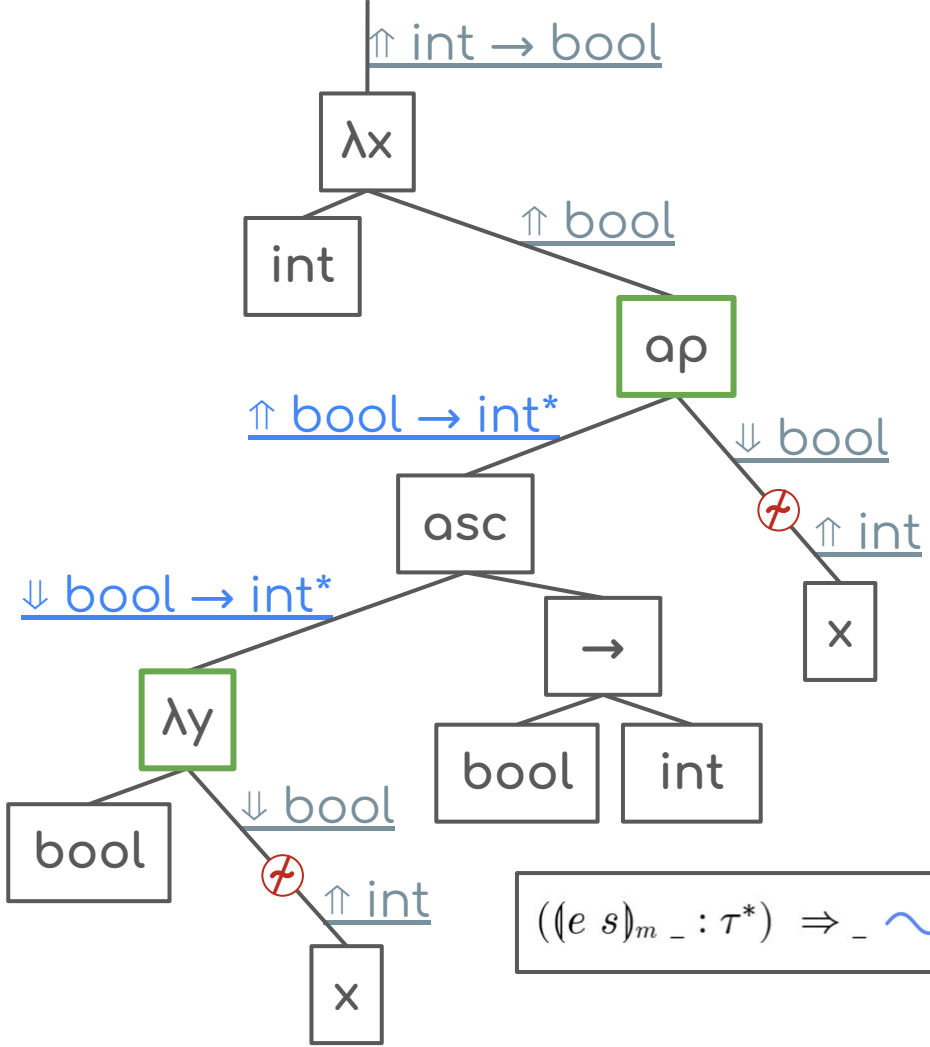
$\Uparrow \text{int} \to \text{bool}$

$$
\begin{array}{lll}
n & ::= & * \mid \quad \text{(newness)} \\
s & ::= & \nRightarrow \mid \Rightarrow \tau^n \text{ (synthesis data)} \\
a & ::= & \nLeftarrow \mid \Leftarrow \tau^n \text{ (analysis data)} \\
m & ::= & T \mid F \text{ (mark bit)} \\
e & ::= & ? \mid x \mid (\!| x : \tau \mapsto \breve{e} |\!)_m \mid (\!| \breve{e} |\!)_m \; \breve{e} \mid \breve{e} : \tau \text{ (expression)} \\
\breve{e} & ::= & (\!| e \;\; s |\!)_m \; a \text{ (expression with local info)}
\end{array}
$$

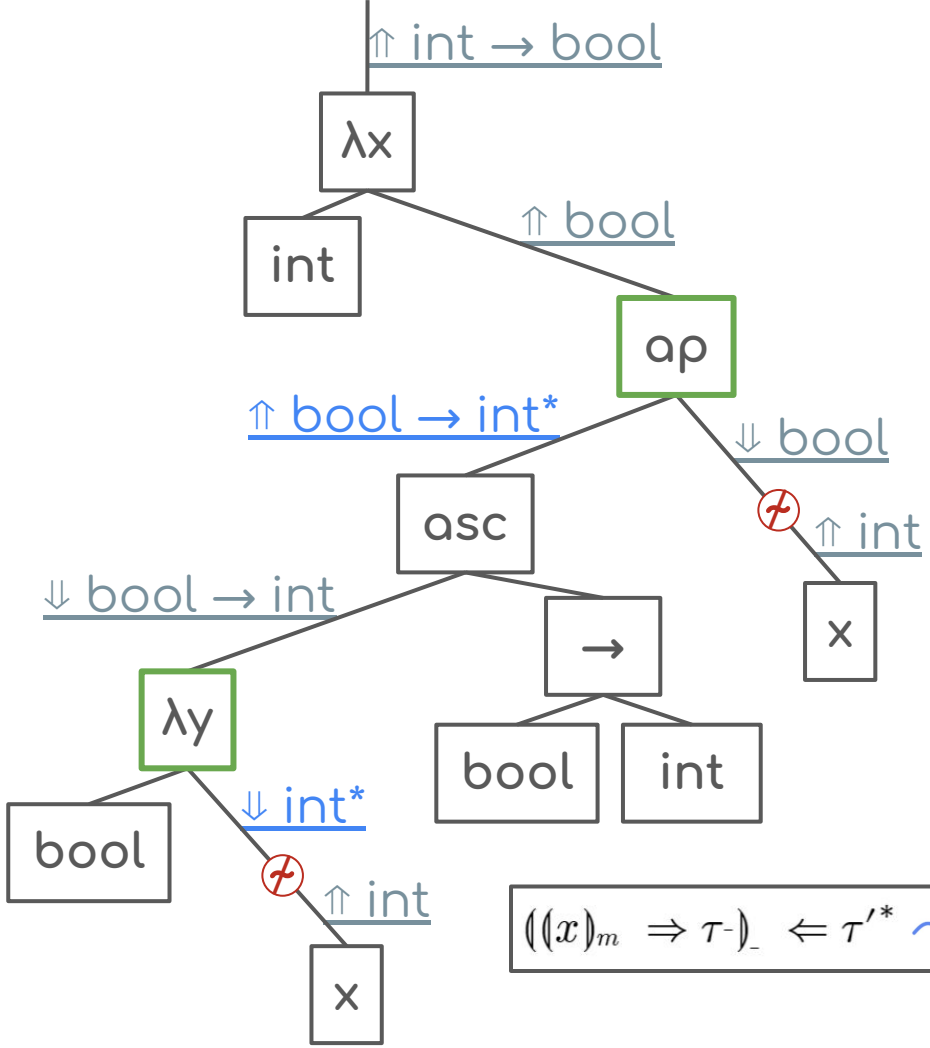




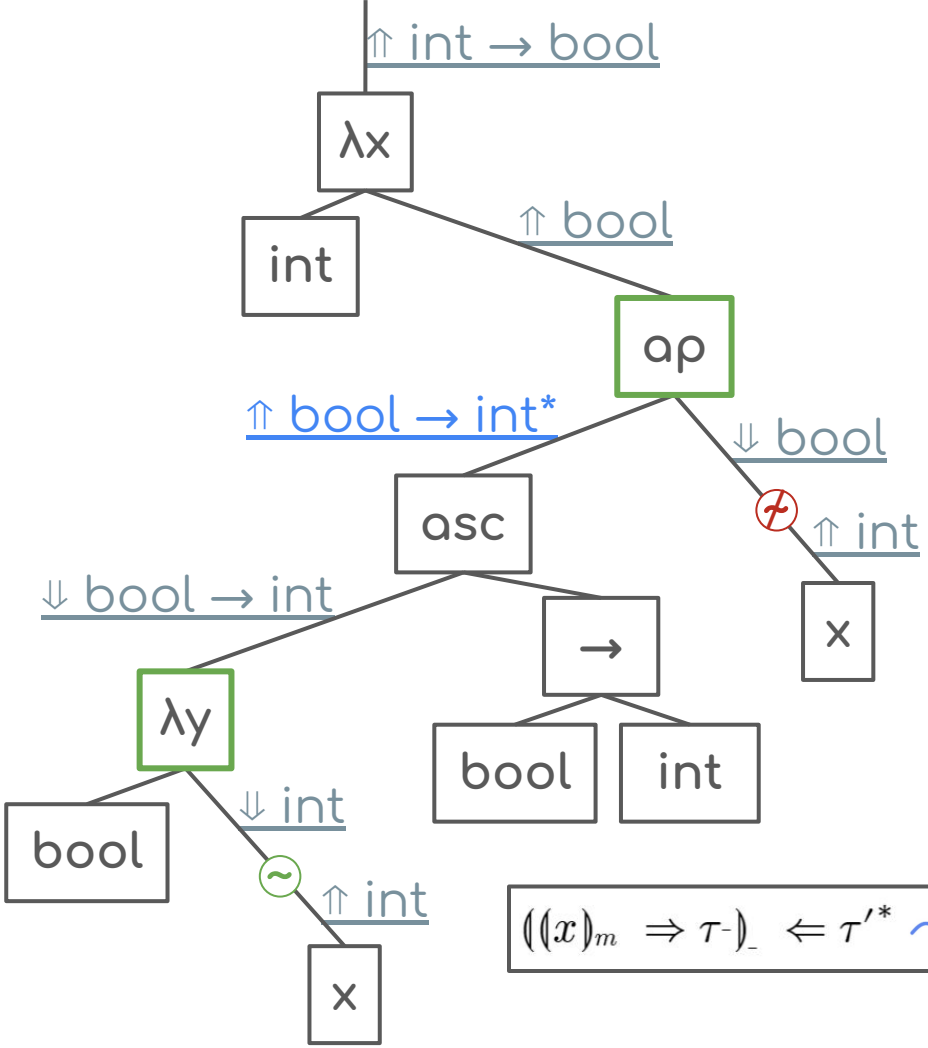
$$(\!| e \;\; s |\!)_m \; \_ : \tau^* \;) \Rightarrow \_ \rightsquigarrow (\!| e \;\; s |\!)_m \Leftarrow \tau^*) : \tau \Rightarrow \tau^*$$



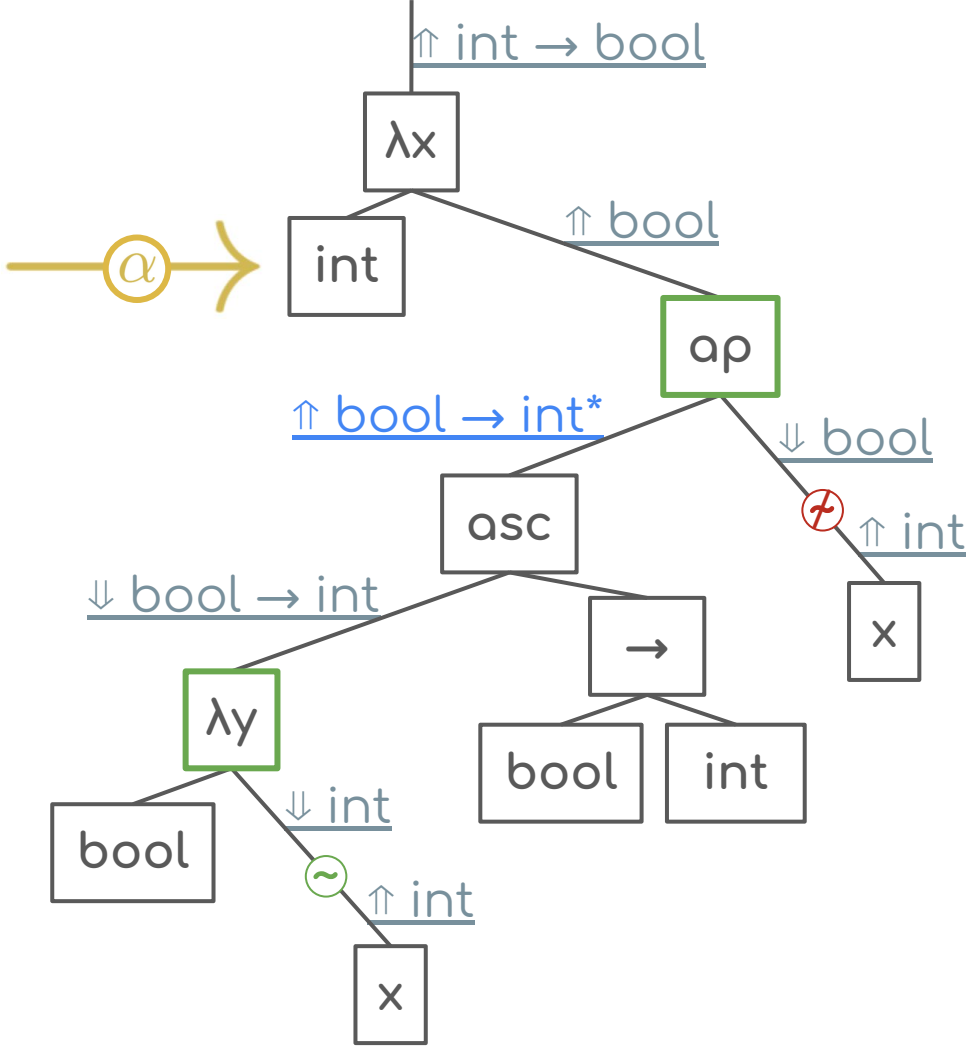
$$((e\ s)_m\ \_ : \tau^*)\ \Rightarrow\ \_ \rightsquigarrow ((e\ s)_m\ \Leftarrow \tau^*) : \tau\ \Rightarrow \tau^*$$

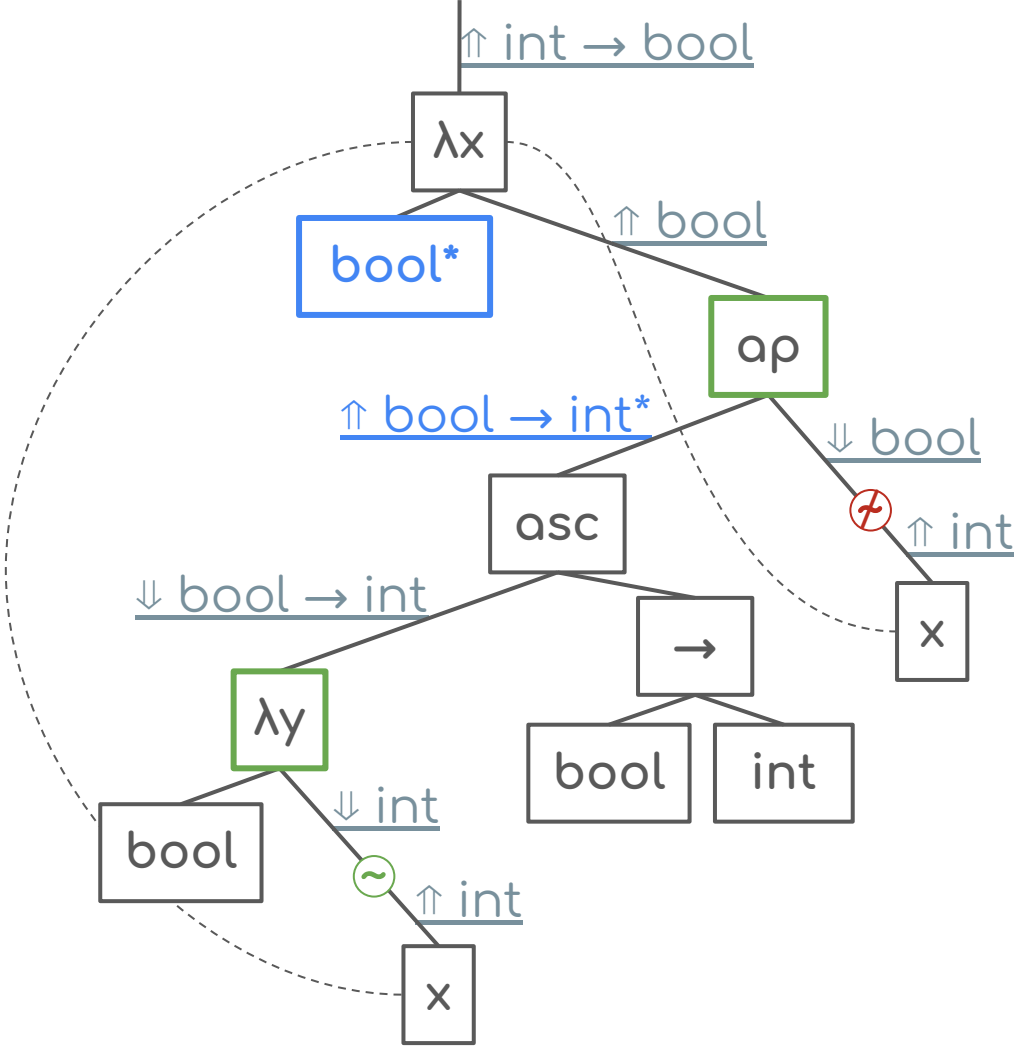


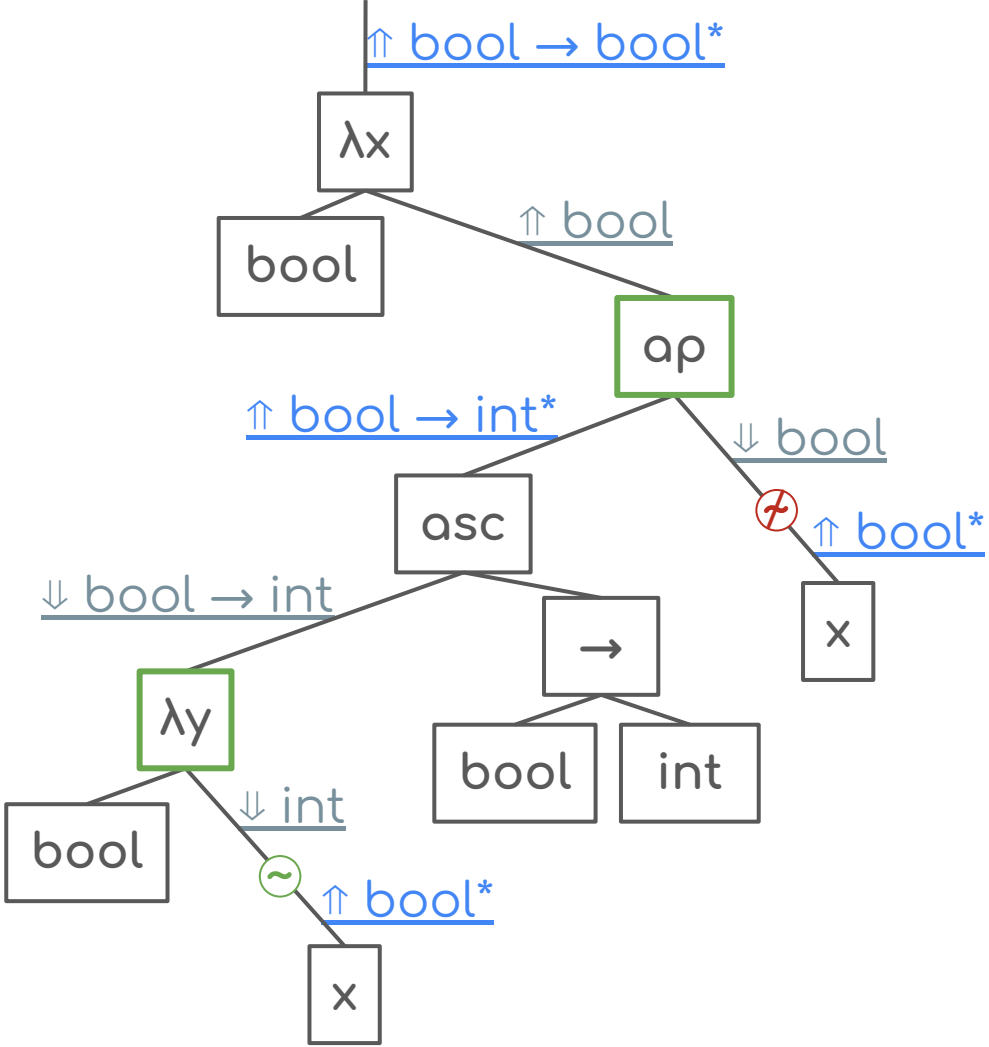
$$((x)_m \Rightarrow \tau\text{-})_{\_} \Leftarrow \tau'^{*} \rightsquigarrow ((x)_m \Rightarrow \tau)_{\tau \sim \tau'} \Leftarrow \tau'$$

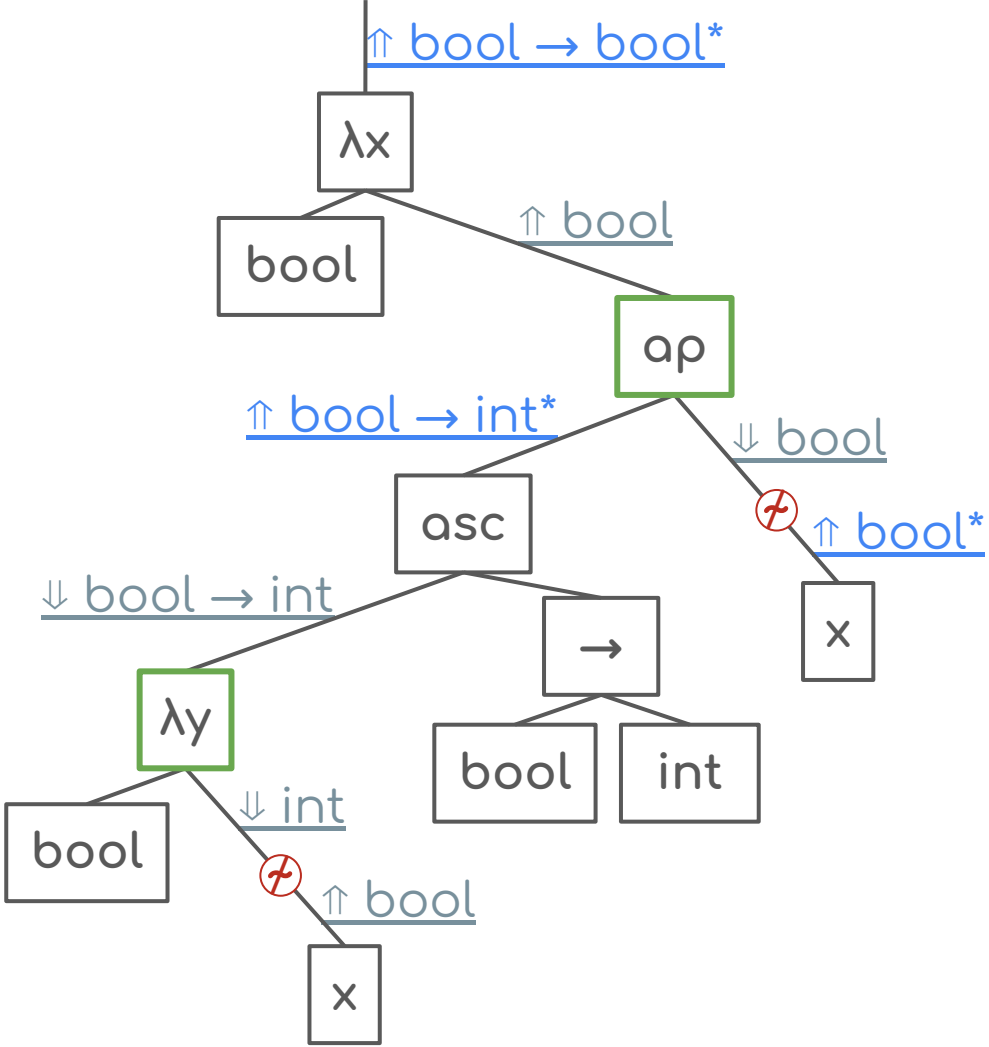


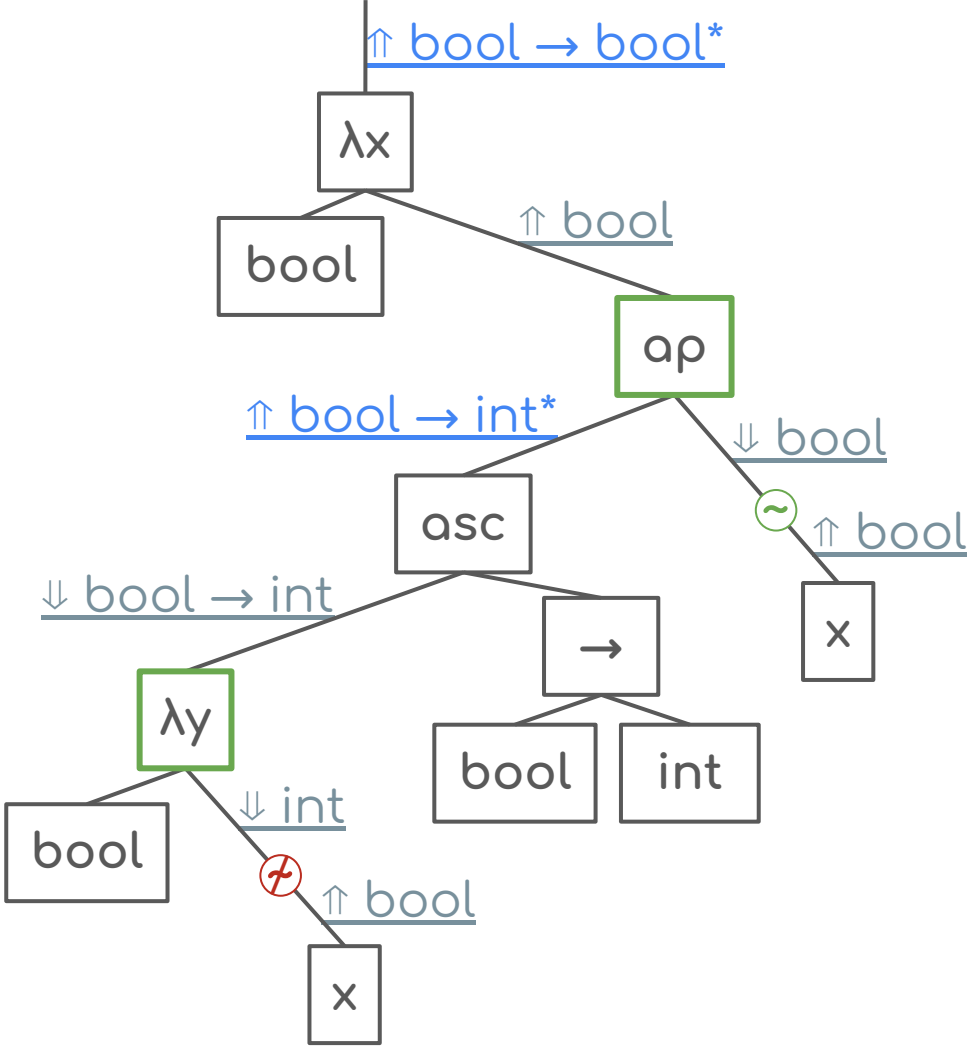
$$((x)_m \Rightarrow \tau_-)_- \Leftarrow \tau'^* \rightsquigarrow ((x)_m \Rightarrow \tau)_{\tau \sim \tau'} \Leftarrow \tau'$$

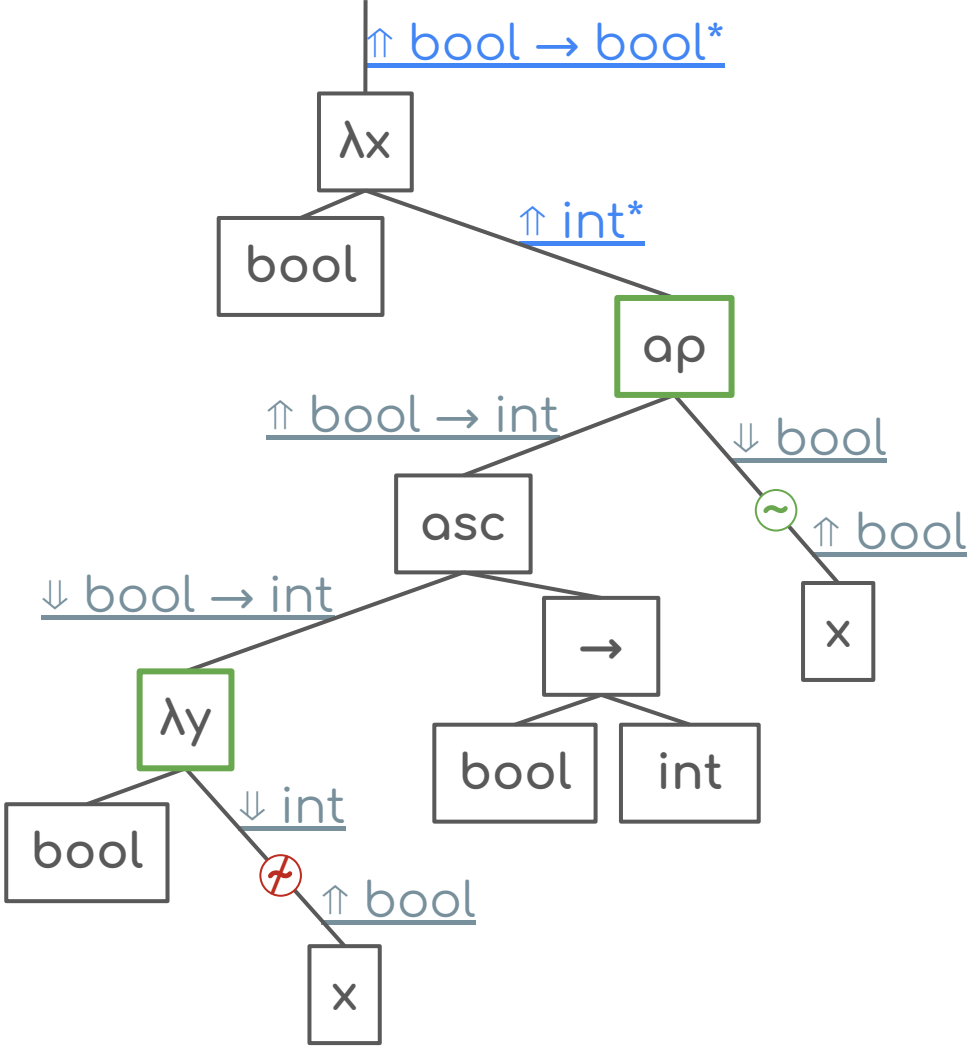

⇑ int → bool
λx
int
⇑ bool
ap
⇑ bool → int*
⇓ bool
asc
⇑ int
⇓ bool → int
→
x
λy
bool
int
bool
⇓ int
⇑ int
x

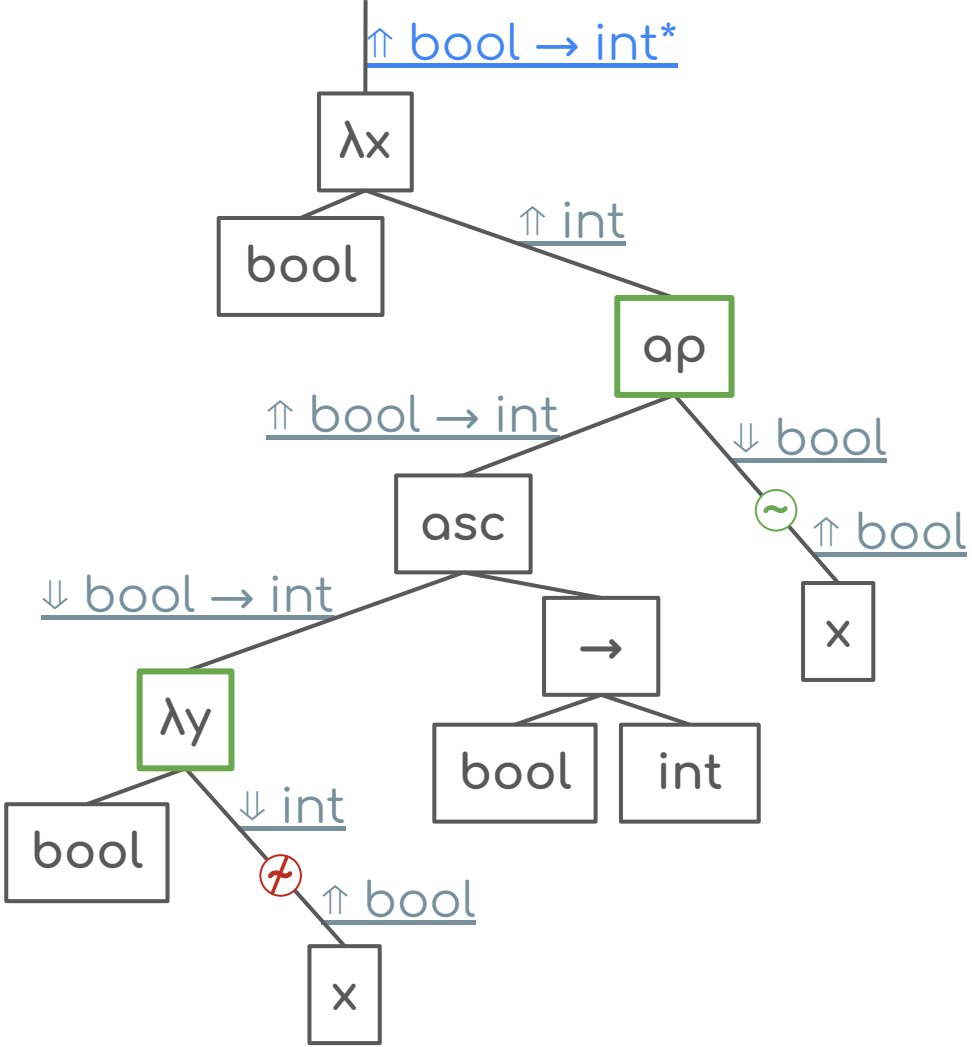

⇑ int → bool
λx
bool*
⇑ bool
ap
⇑ bool → int*
⇓ bool
asc
⇑ int
⇓ bool → int
x
→
λy
bool
int
⇓ int
bool
⇑ int
x


⇑ bool → bool*
λx
bool
⇑ bool
ap
⇑ bool → int*
⇓ bool
asc
⇑ bool*
x
⇓ bool → int
→
λy
bool
int
bool
⇓ int
⇑ bool*
x

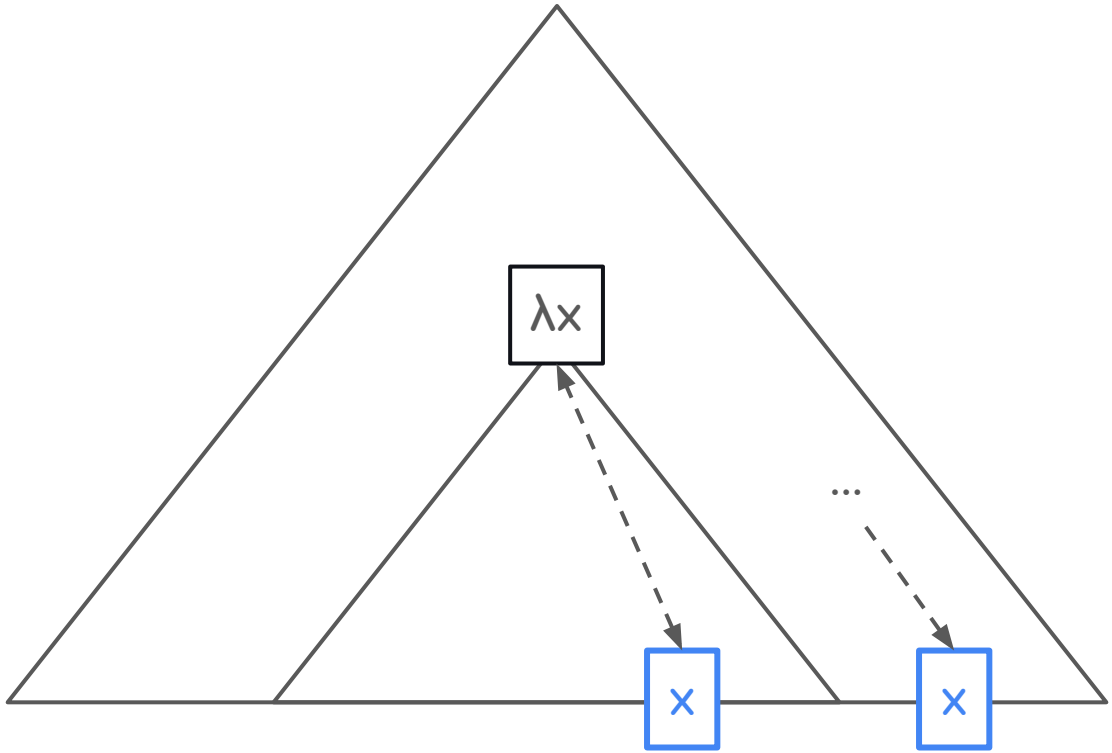

⇑ bool → bool*
λx
bool
⇑ bool
ap
⇑ bool → int*
⇓ bool
asc
≠
⇑ bool*
⇓ bool → int
→
x
λy
bool
int
bool
⇓ int
≠
⇑ bool
x


⇑ bool → bool*
λx
bool
⇑ bool
ap
⇑ bool → int*
⇓ bool
asc
~
⇑ bool
⇓ bool → int
→
x
λy
bool
int
bool
⇓ int
≠
⇑ bool
x

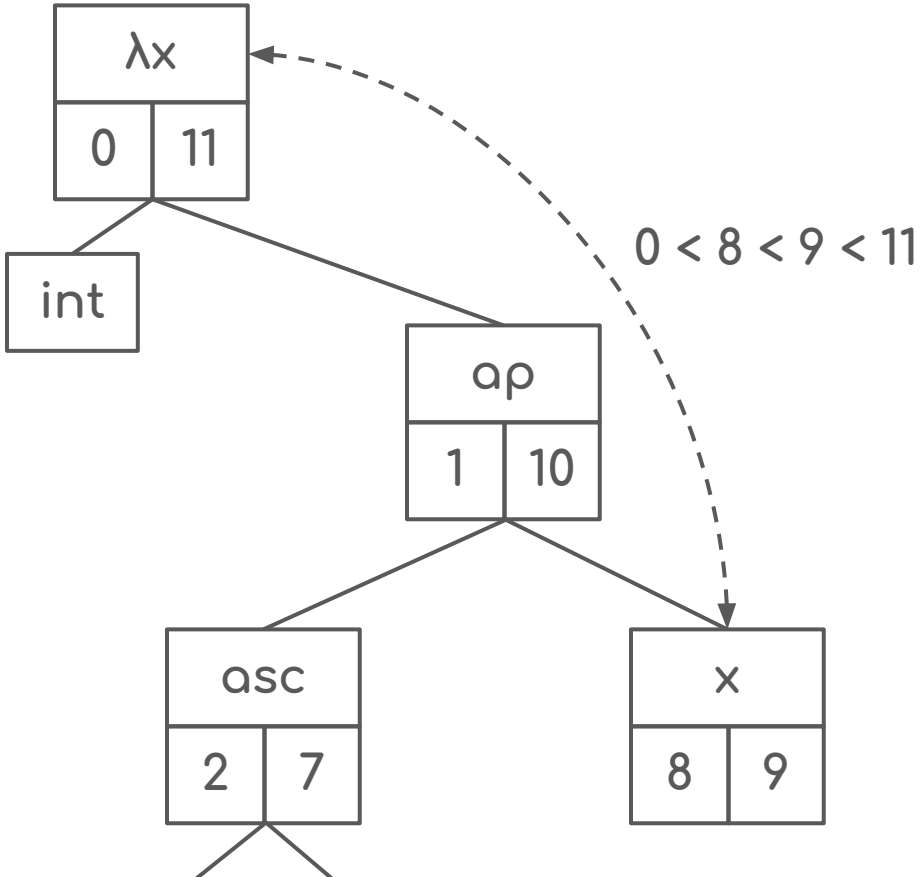

⇑ bool → bool*
λx
bool
⇑ int*
ap
⇑ bool → int
⇓ bool
asc
~
⇑ bool
x
⇓ bool → int
→
λy
bool
int
bool
⇓ int
≠
⇑ bool
x


⇑ bool → int*
λx
bool
⇑ int
ap
⇑ bool → int
⇓ bool
asc
~
⇑ bool
⇓ bool → int
→
x
λy
bool
int
bool
⇓ int
≠
⇑ bool
x

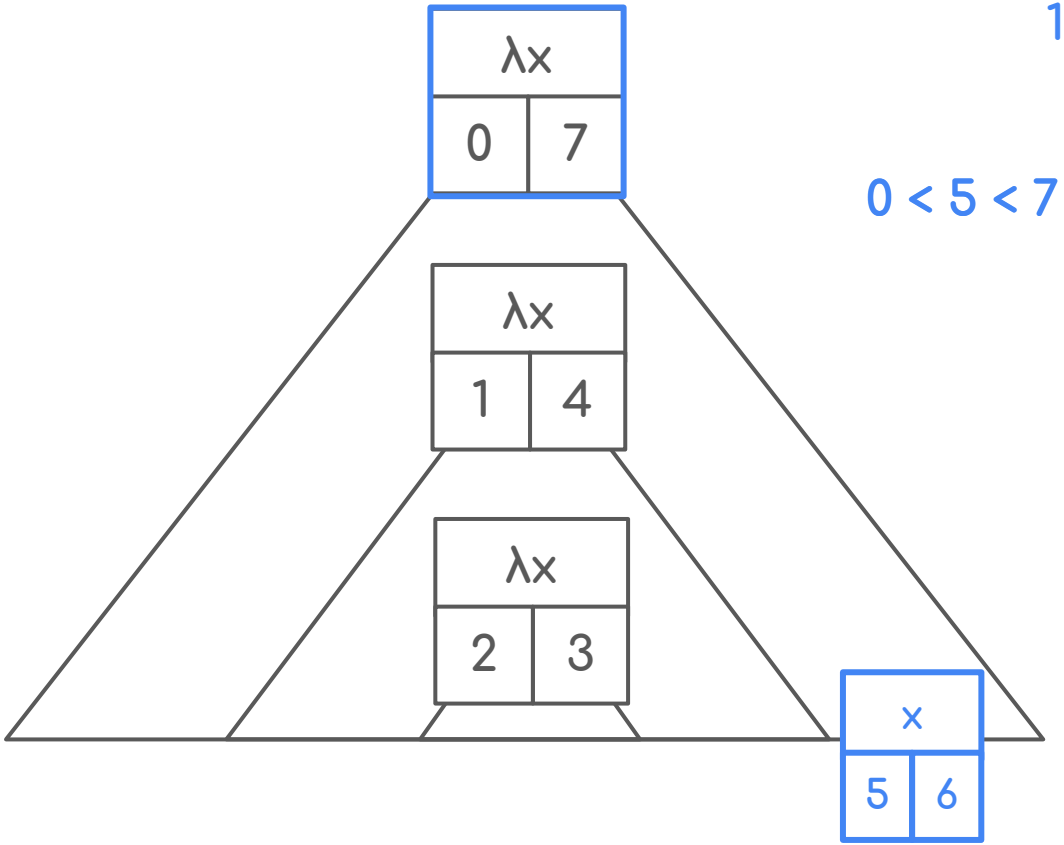
The Binding Problem

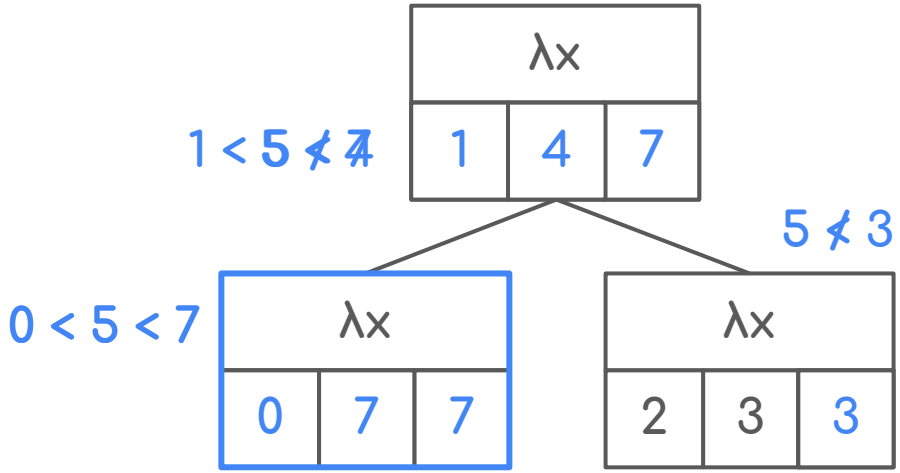


Ancestry Check

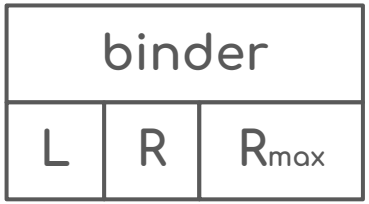


Lowest Ancestor





Balanced BST



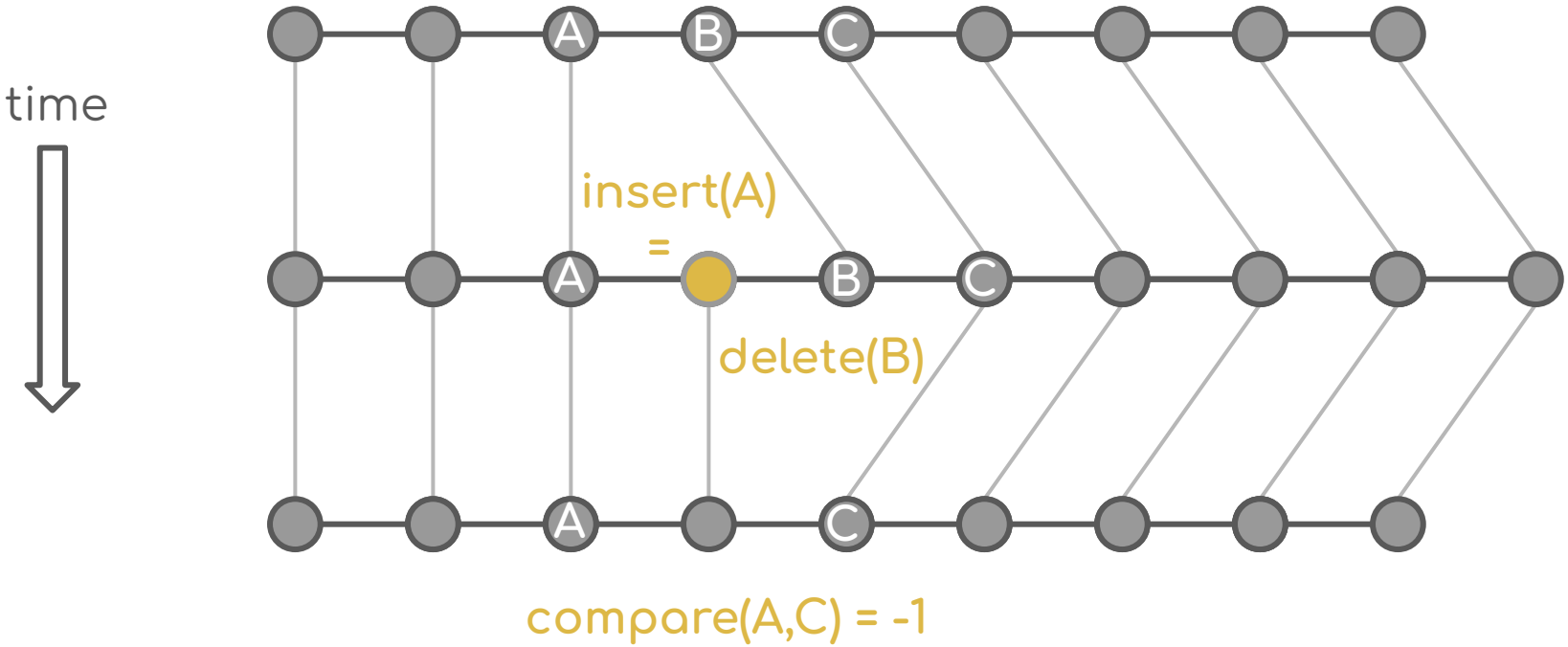
Many Years Ago...

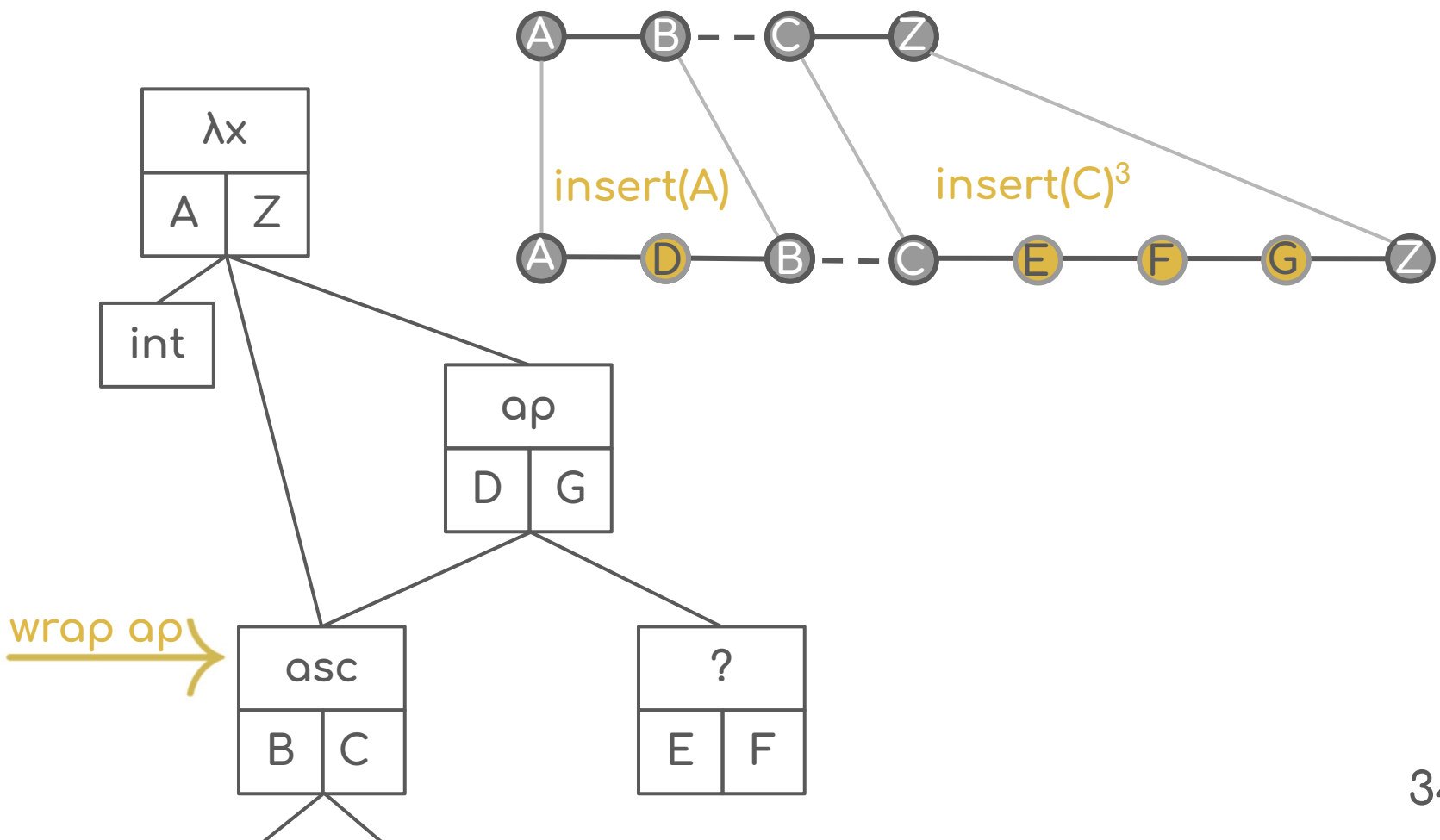
Two Algorithms for Maintaining Order in a List

Paul F. Dietz
Schlumberger-Doll Research
Ridgefield, CT 06877

Daniel D. Sleator
Carnegie-Mellon University
Pittsburgh, PA 15213

Maintaining Order




A
B
C
Z
insert(A)
insert(C)3
A
D
B
C
E
F
G
Z
λx
A
Z
int
ap
D
G
wrap ap
asc
B
C
?
E
F

Open Questions:

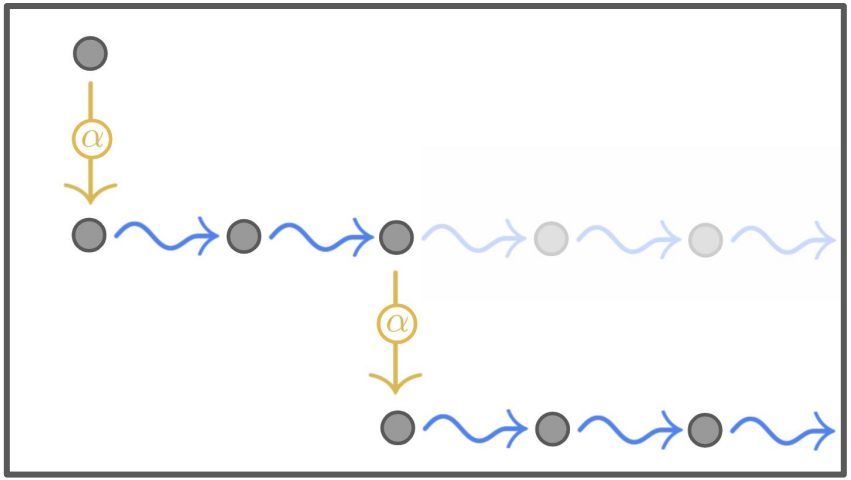
Existing implementation techniques?

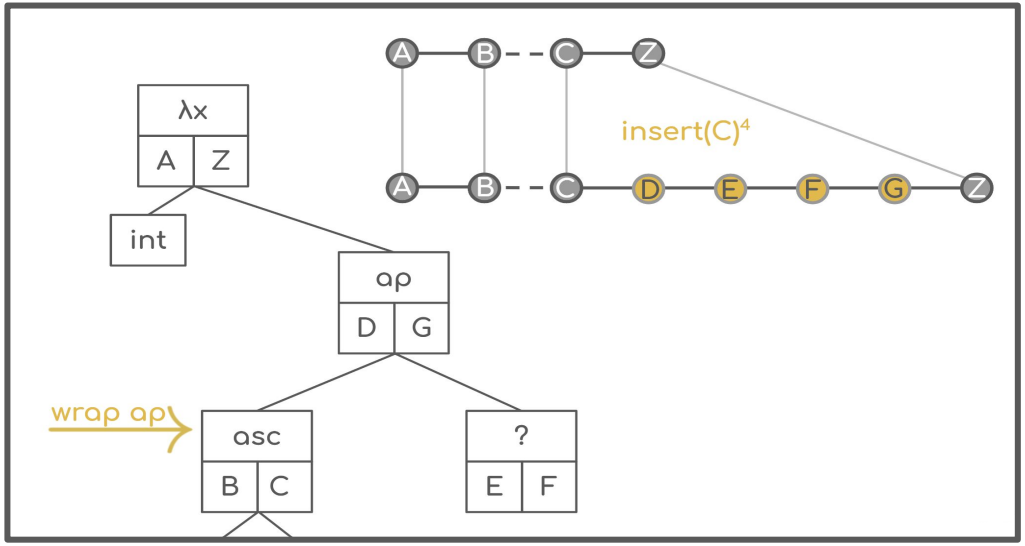
How to prioritize updates?

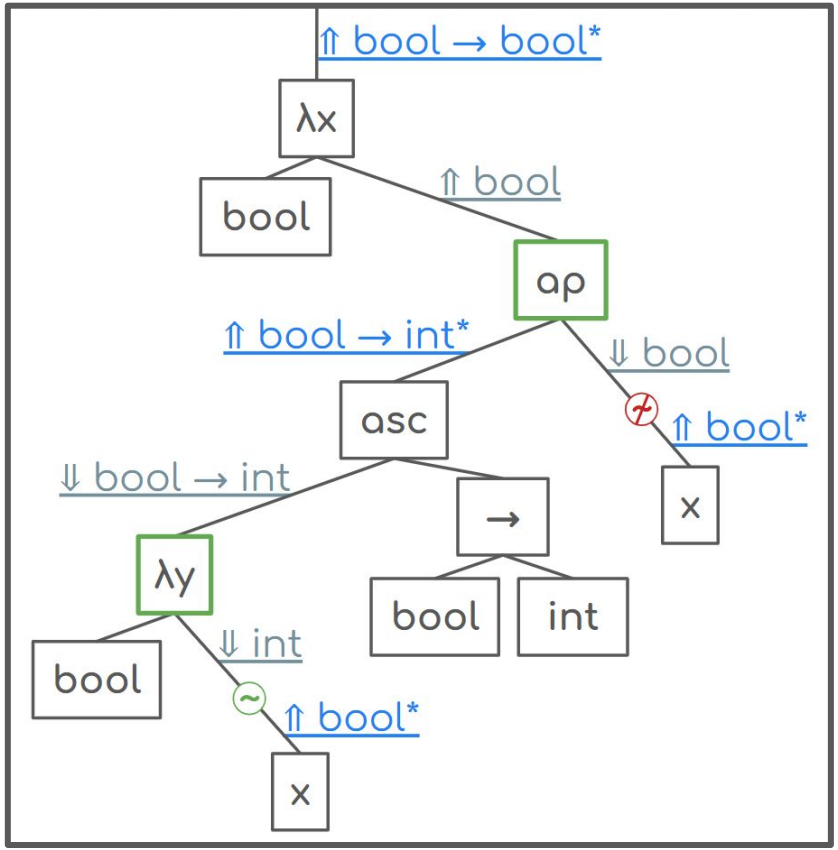
Can we achieve more incrementality?

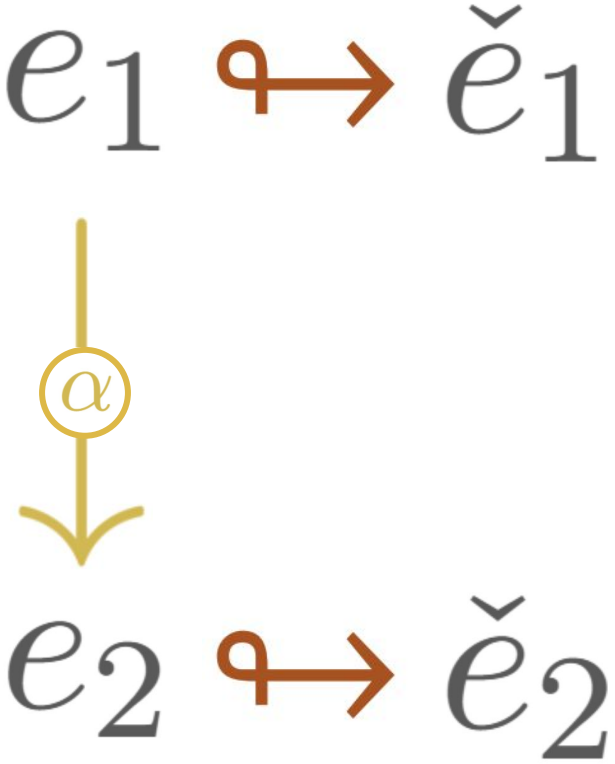
Can we use a generalized approach?

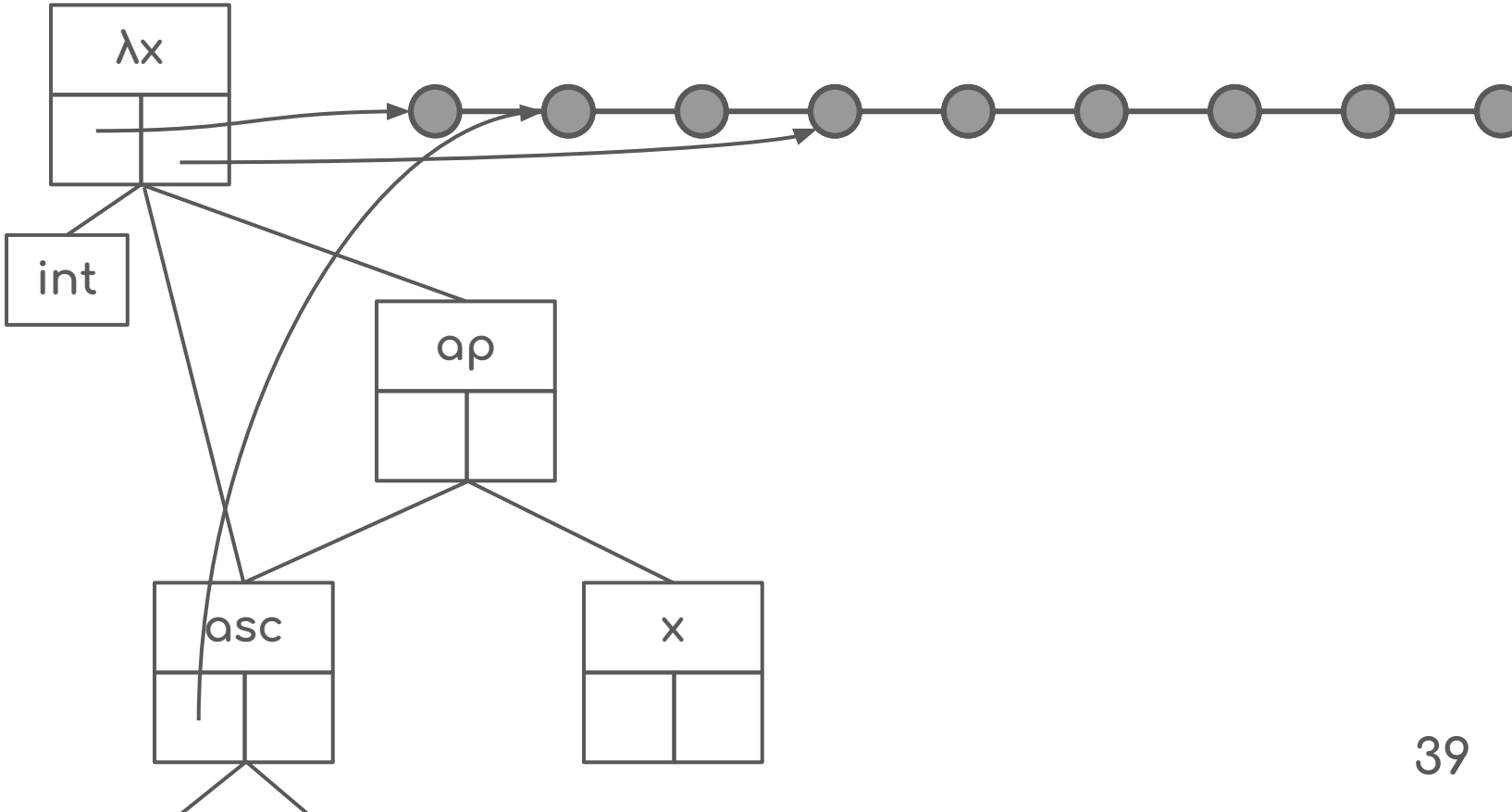
Thank you for listening!

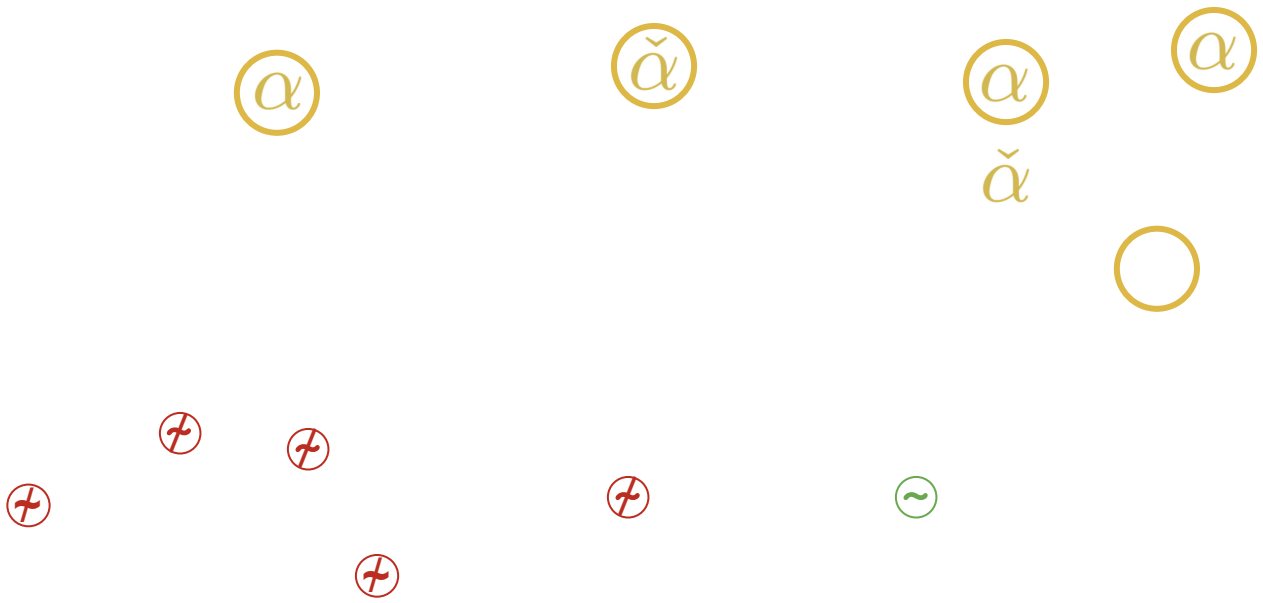

α
α

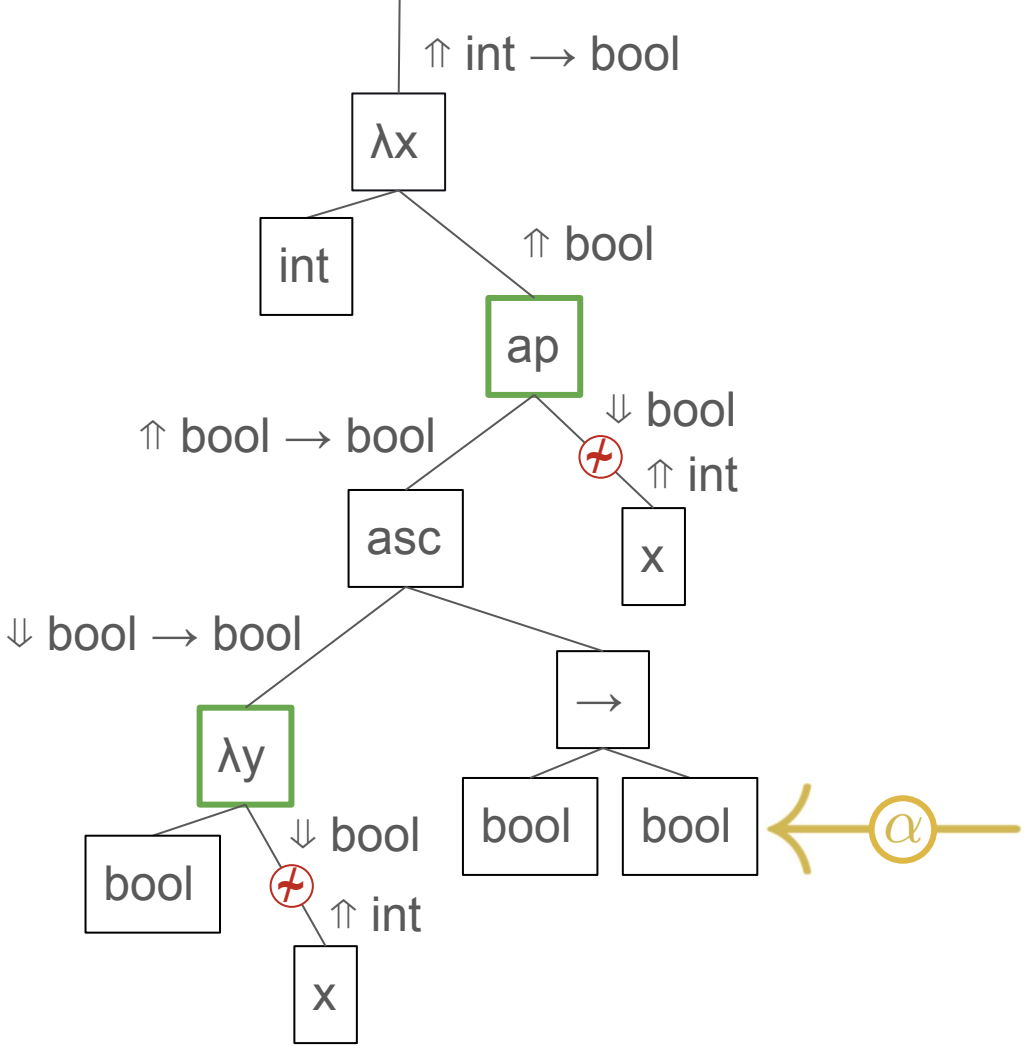

A
B
C
Z
insert(C)⁴
A
B
C
D
E
F
G
Z
λx
A
Z
int
ap
D
G
wrap ap
asc
B
C
?
E
F


⇑ bool → bool*
λx
bool
⇑ bool
ap
⇑ bool → int*
⇓ bool
asc
⇓ bool → int
⇑ bool*
x
λy
→
bool
int
bool
⇓ int
⇑ bool*
x


$e_1 \rightarrowtail \check{e}_1$
α
$e_2 \rightarrowtail \check{e}_2$


λx
int
ap
asc
x




⇑ int → bool
λx
int
⇑ bool
ap
⇓ bool
⇑ bool → bool
⇑ int
asc
x
⇓ bool → bool
→
λy
bool
bool
bool
⇓ bool
⇑ int
x
α