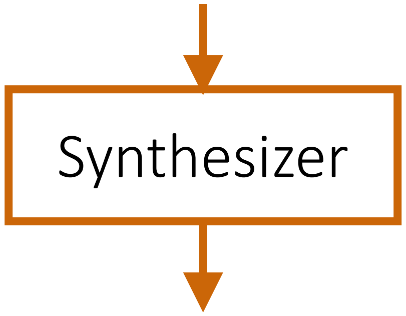
Programs Synthesis from Polymorphic Refinement Types

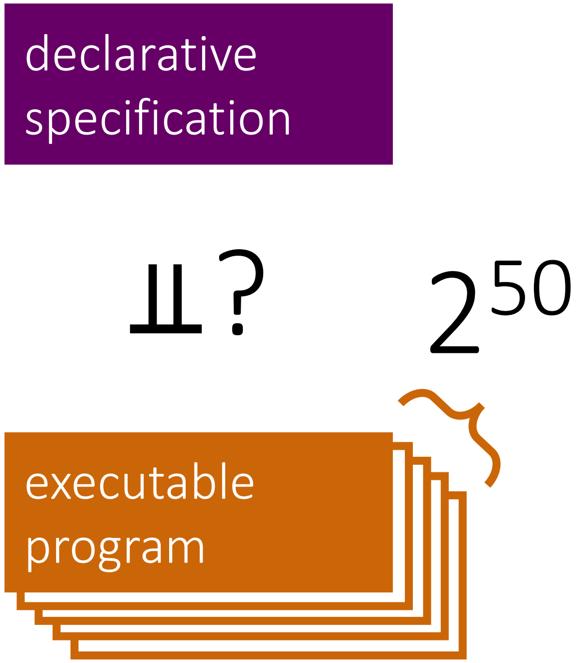
Nadia Polikarpova
Ivan Kuraj
Armando Solar-Lezama

Program synthesis

“Make a list with *n* copies of *x*”

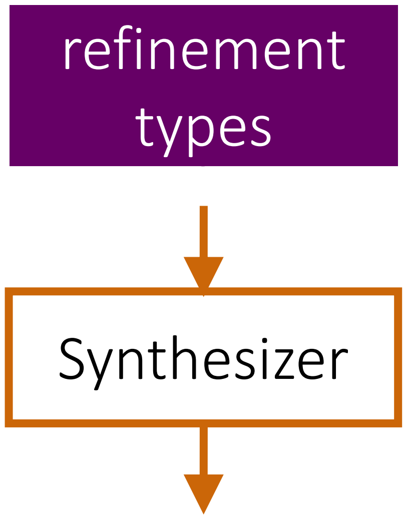


```
replicate n x =
  if n ≤ 0
    then Nil
    else Cons x
      (replicate (dec n) x)
```



Modular verification for synthesis

Specifications for synthesis



1. supports automatic, modular verification
2. abstract and concise
3. sufficiently expressive

```
replicate n x =
  if n ≤ 0
    then Nil
    else Cons x (replicate (dec n) x)
```

Demo: replicate

```
-- Specification:
replicate :: n: Nat → x: α → {v: List α | len v = n}
replicate = ??

-- Components:
zero :: {v: Int | v = 0}
inc :: x: Int → {v: Int | v = x + 1}
dec :: x: Int → {v: Int | v = x - 1}
leq :: x: Int → y: Int → {Bool | v = (x ≤ y) }
neq :: x: Int → y: Int → {Bool | v = (x ≠ y) }
```

Synthesis from refinement types

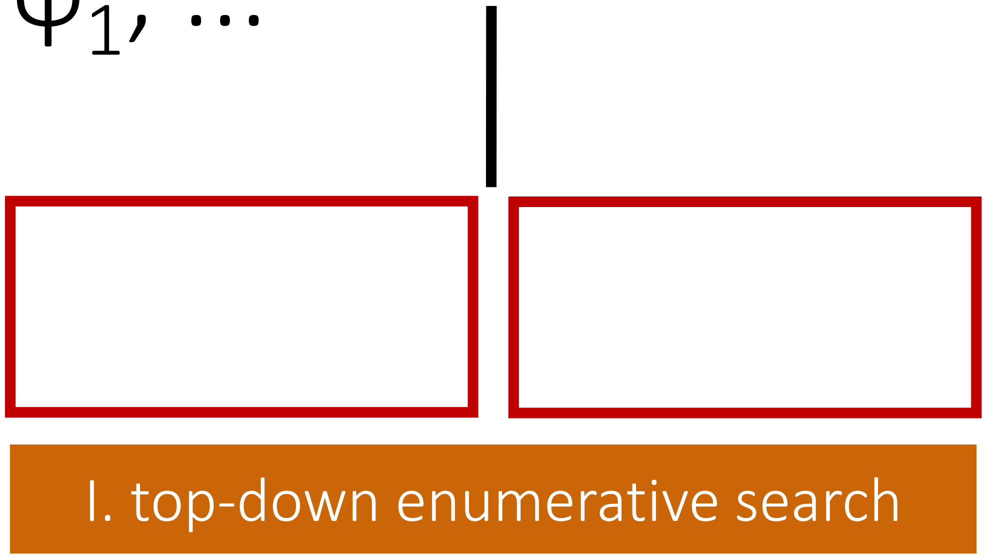
$$\Gamma \vdash ?? :: T$$

Synthesis from refinement types

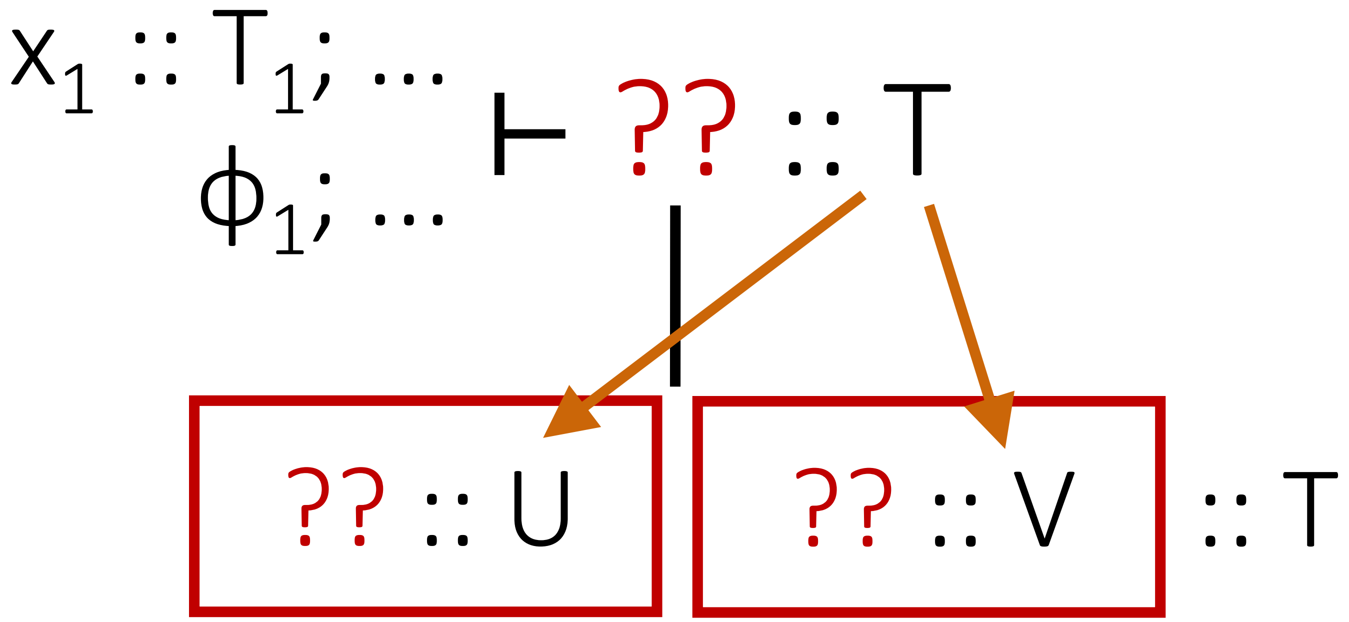
$$\begin{array}{r} x_1 :: T_1; \ldots \\ \phi_1; \ldots \end{array} \vdash \;??\; :: T$$

Synthesis from refinement types

$$\begin{array}{l} x_1 :: T_1; \ldots \\ \quad \phi_1; \ldots \end{array} \vdash \textcolor{red}{??} :: T$$

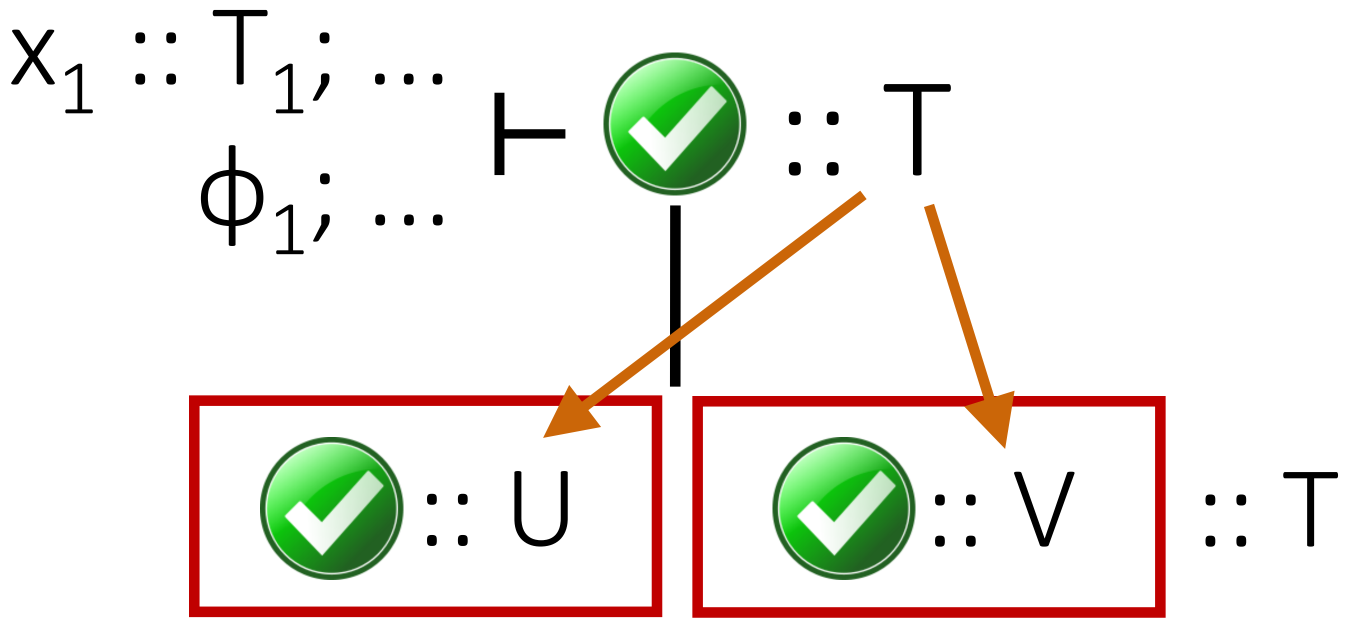


Synthesis from refinement types



I. top-down enumerative search

Synthesis from refinement types



I. top-down enumerative search

Synthesis from refinement types

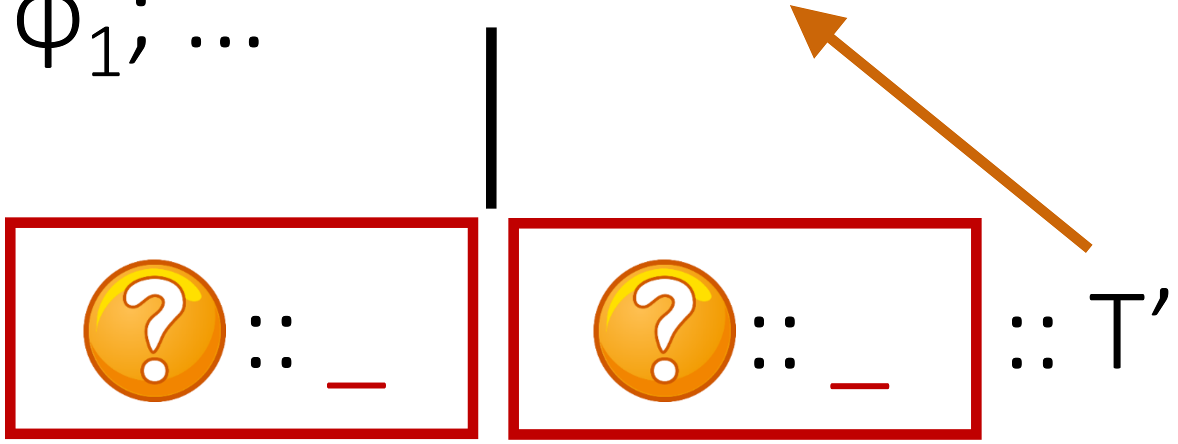
$$\begin{array}{l} x_1 :: T_1; \ldots \\ \quad \phi_1; \ldots \end{array} \vdash ?? :: T$$

$$\begin{tikzcd} ?? :: T \arrow[d, dash] \\ \boxed{?? :: \_}\ \boxed{?? :: \_} :: T' \end{tikzcd}$$

I. top-down enumerative search

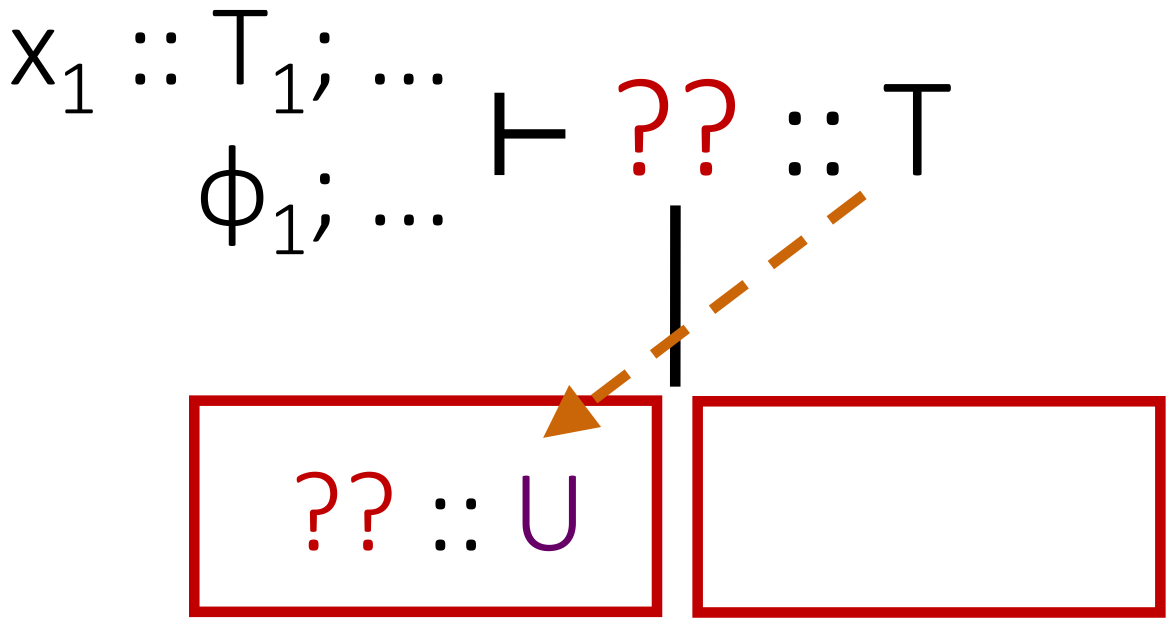
Synthesis from refinement types

$$x_1 :: T_1; \ldots \quad \phi_1; \ldots \vdash {\color{red}??} :: T$$



I. top-down enumerative search

Synthesis from refinement types

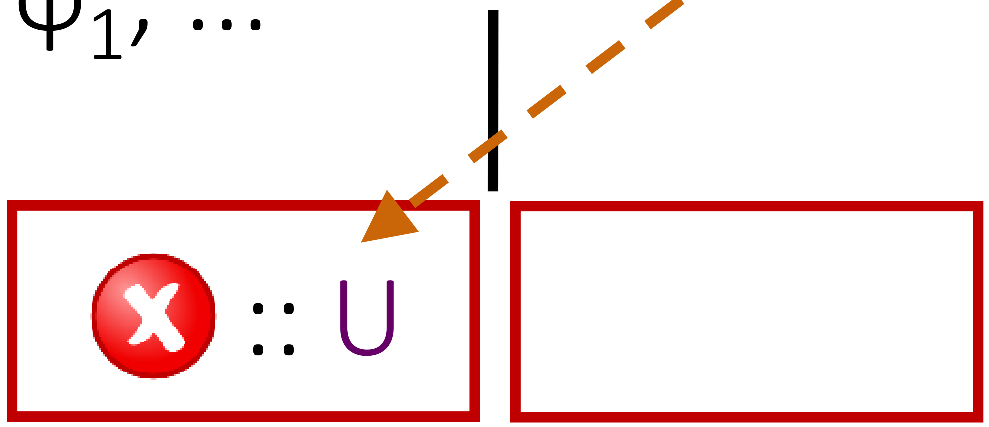


I. top-down enumerative search

II. round-trip type checking

Synthesis from refinement types

$$\begin{array}{l} x_1 :: T_1; \ldots \\ \quad \phi_1; \ldots \end{array} \vdash \;??\; :: T$$

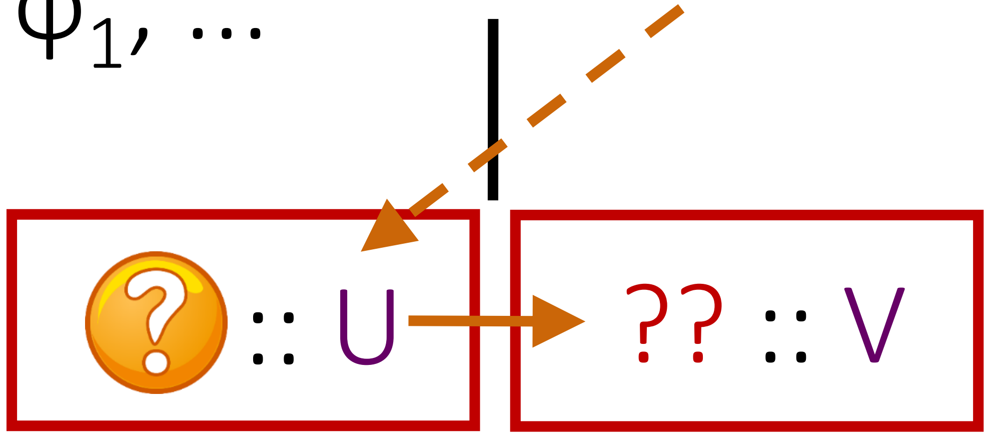


I. top-down enumerative search

II. round-trip type checking

Synthesis from refinement types

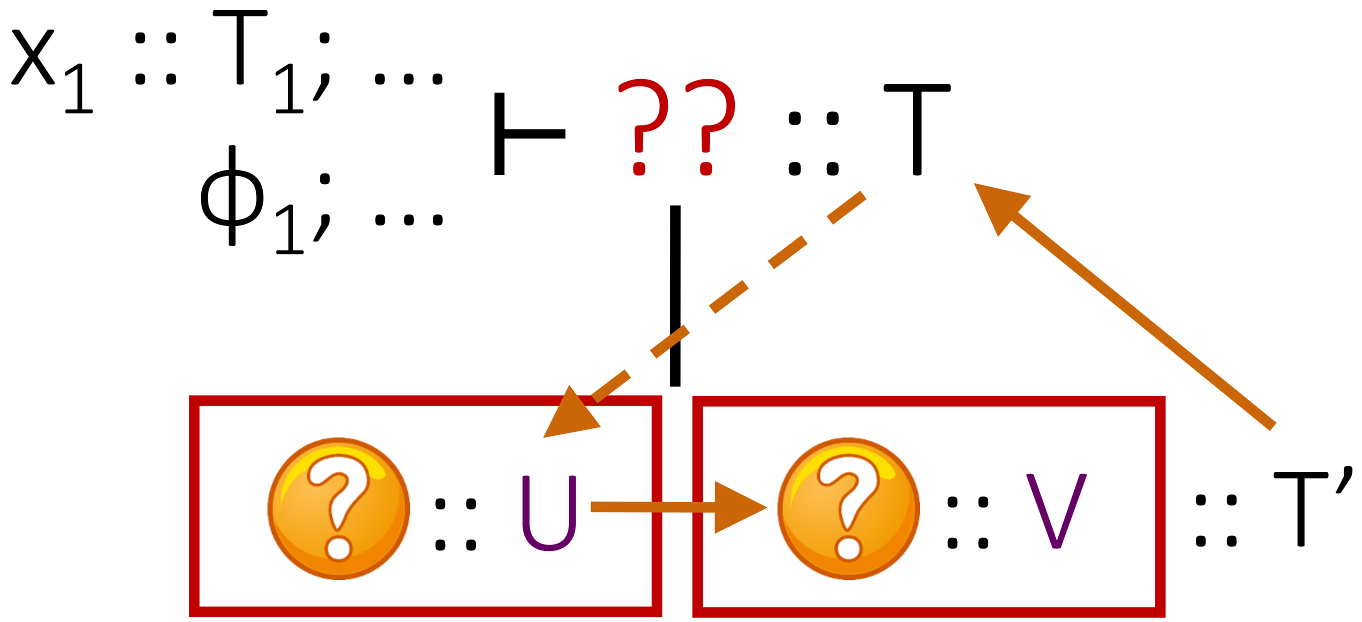
$$x_1 :: T_1; \dots \quad \phi_1; \dots \vdash {\color{red}??} :: T$$



I. top-down enumerative search

II. round-trip type checking

Synthesis from refinement types

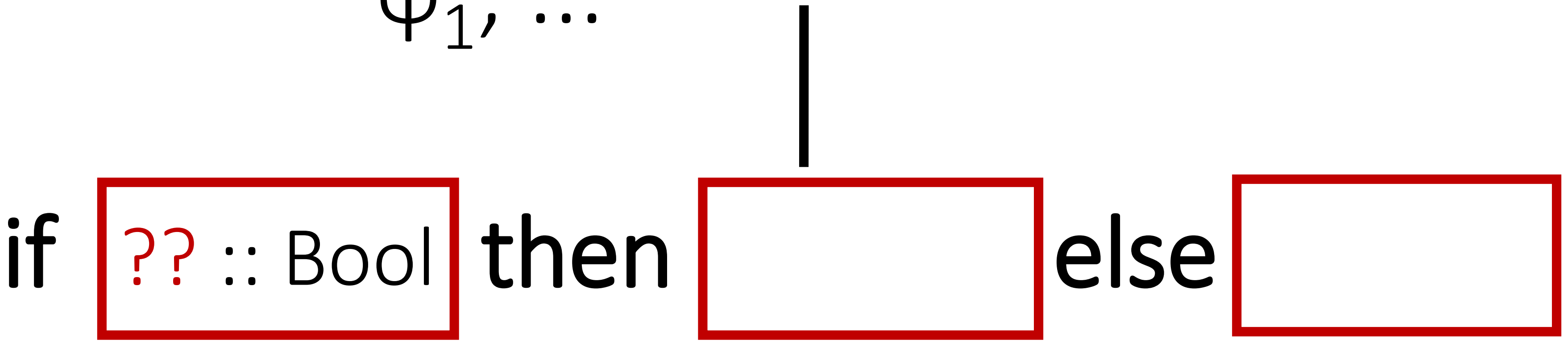


I. top-down enumerative search

II. round-trip type checking

Synthesis from refinement types

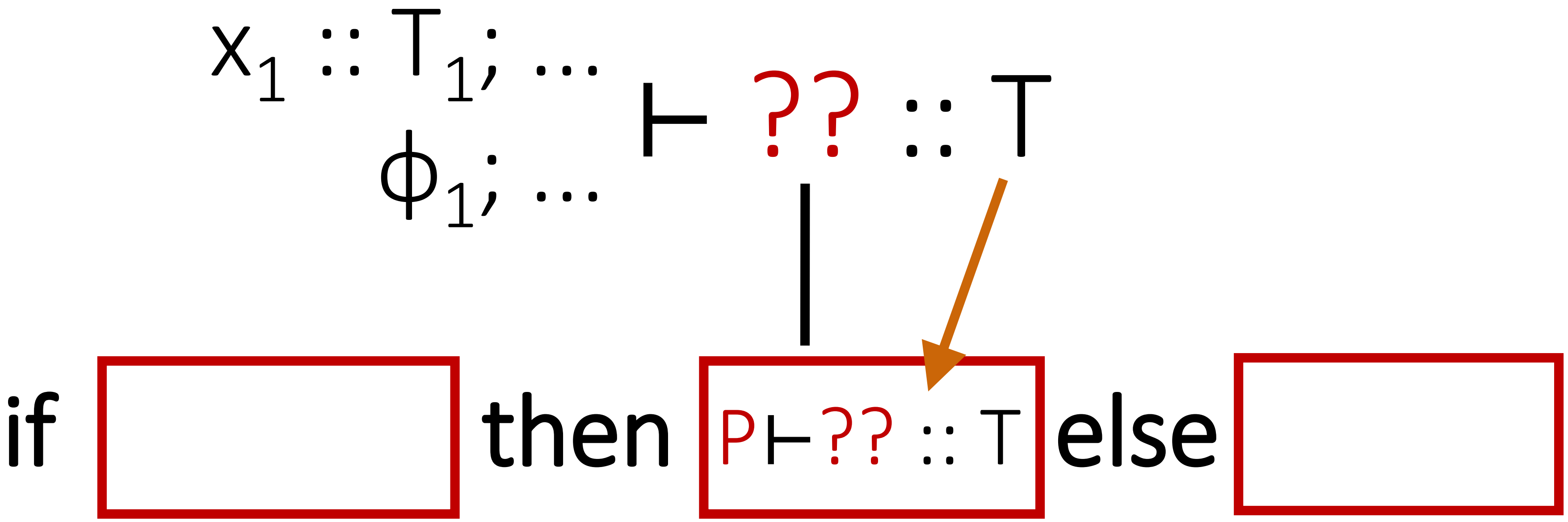
$$\begin{array}{r} x_1 :: T_1; \ldots \\ \phi_1; \ldots \end{array} \vdash \;??\; :: T$$



I. top-down enumerative search

II. round-trip type checking

Synthesis from refinement types

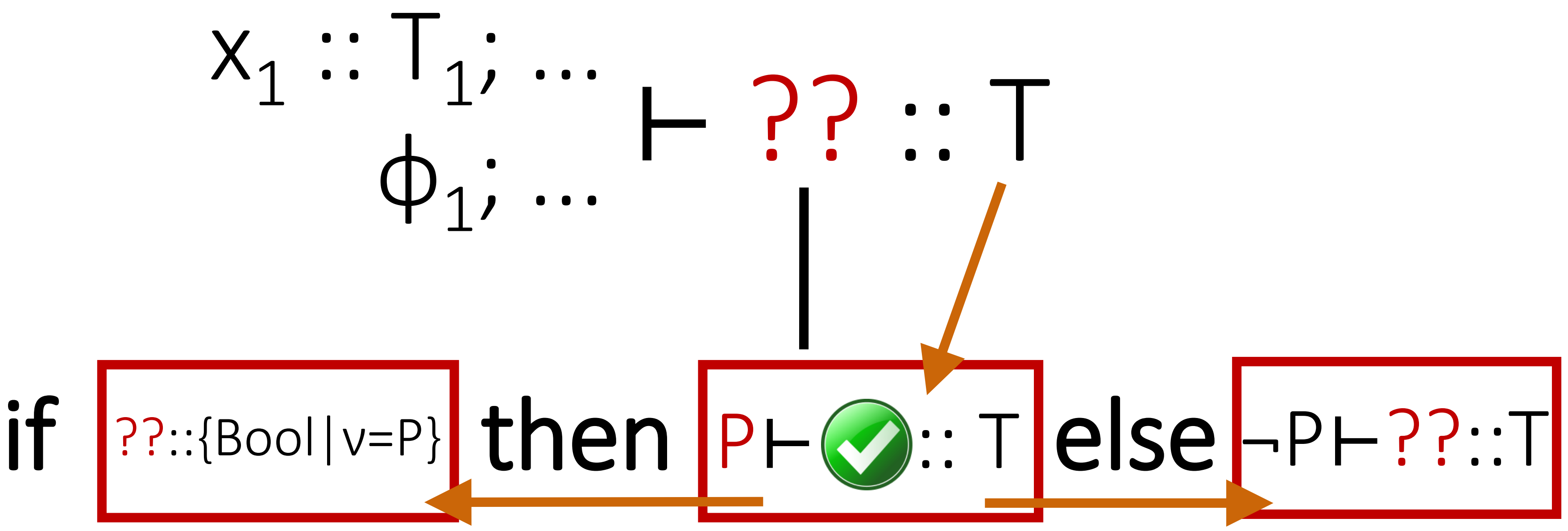


I. top-down enumerative search

II. round-trip type checking

III. condition abduction

Synthesis from refinement types



I. top-down enumerative search

II. round-trip type checking

III. condition abduction

Round-trip type checking

$$\Gamma \vdash \; ?? \; :: \; \{\text{List Neg} \mid \text{len } \nu \geq 5\}$$

Round-trip type checking

$$\begin{array}{r} \text{Nil ; 0 ; 5 ; -5} \\ \text{zeros} \\ \text{replicate ; Cons} \end{array} \vdash \;\; ?? \;\; :: \;\; \{\text{List Neg} \mid \text{len } \nu \geq 5\}$$

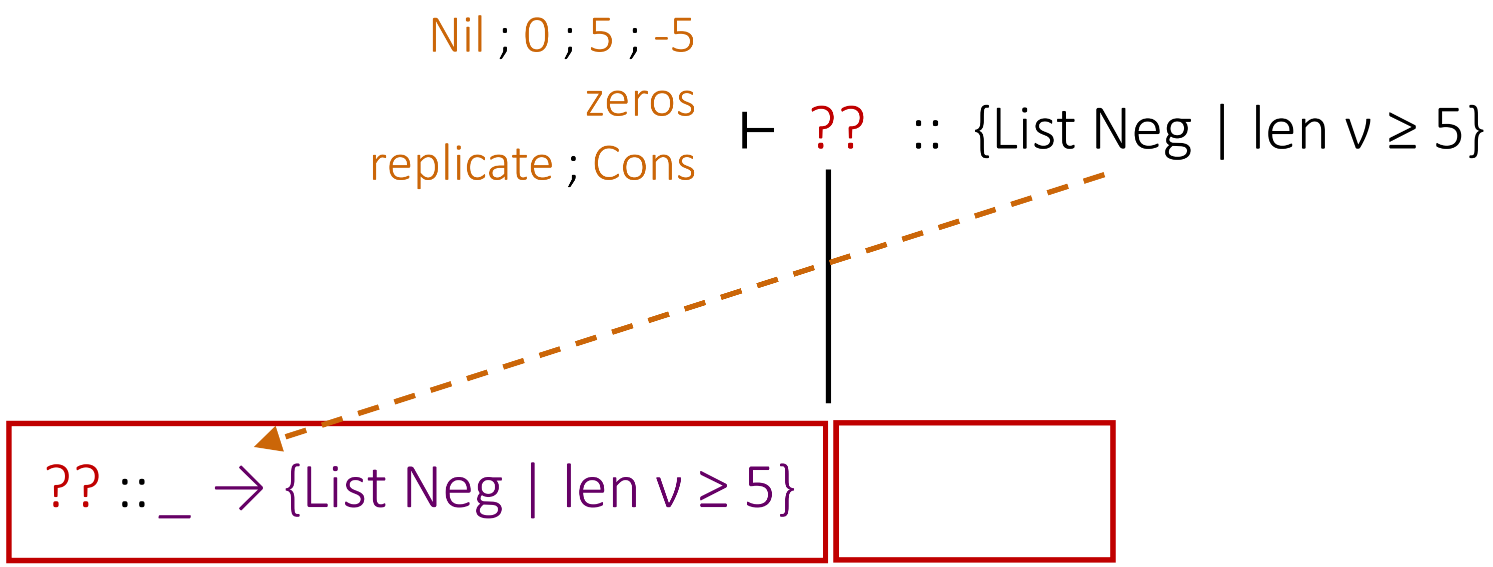
Round-trip type checking

Nil :: {List a | len ν = 0} ; 0 ; 5 ; -5
zeros
replicate ; Cons

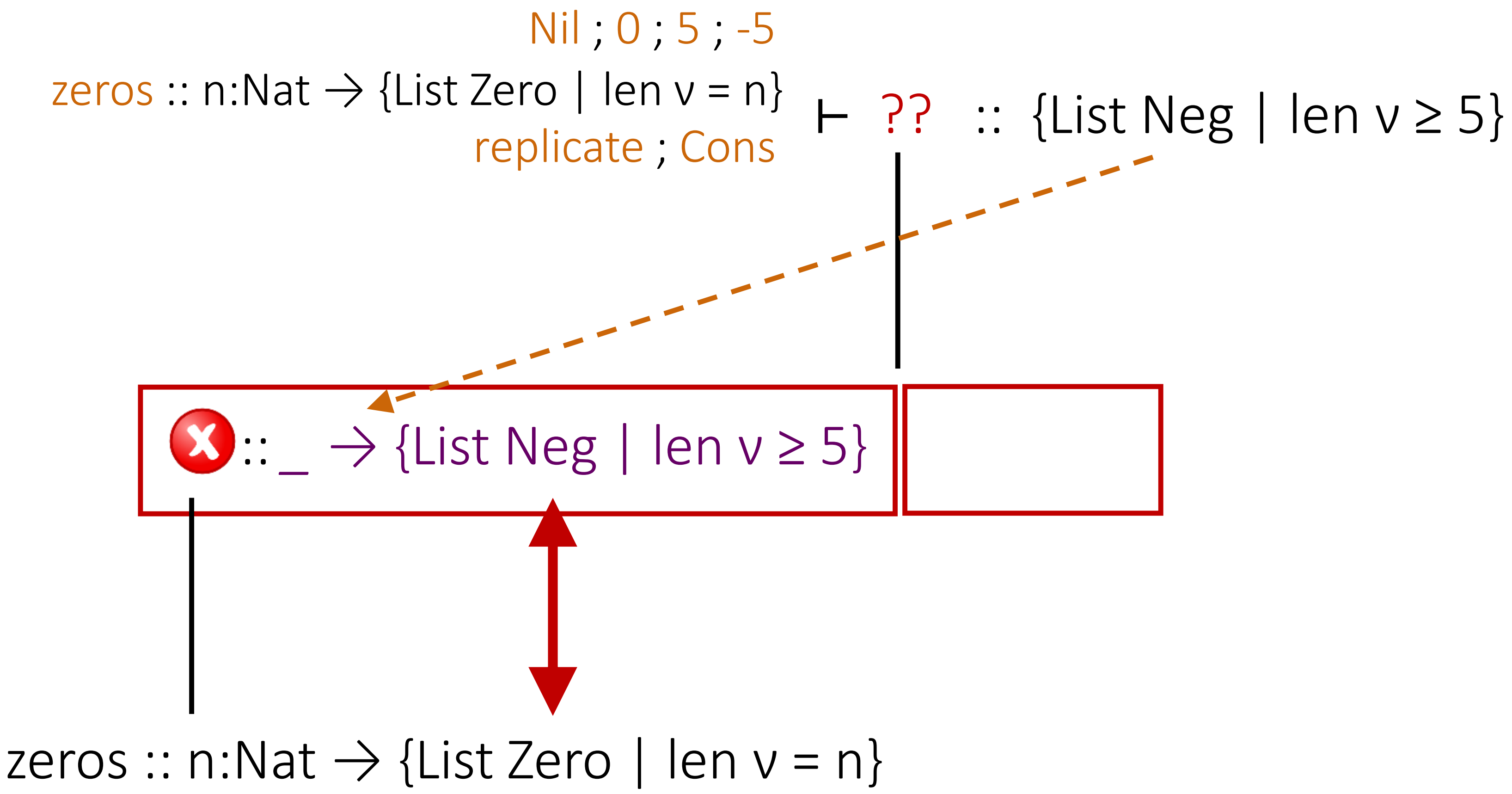
⊢ ✗ :: {List Neg | len ν ≥ 5}

Nil :: {List Neg|len ν = 0}

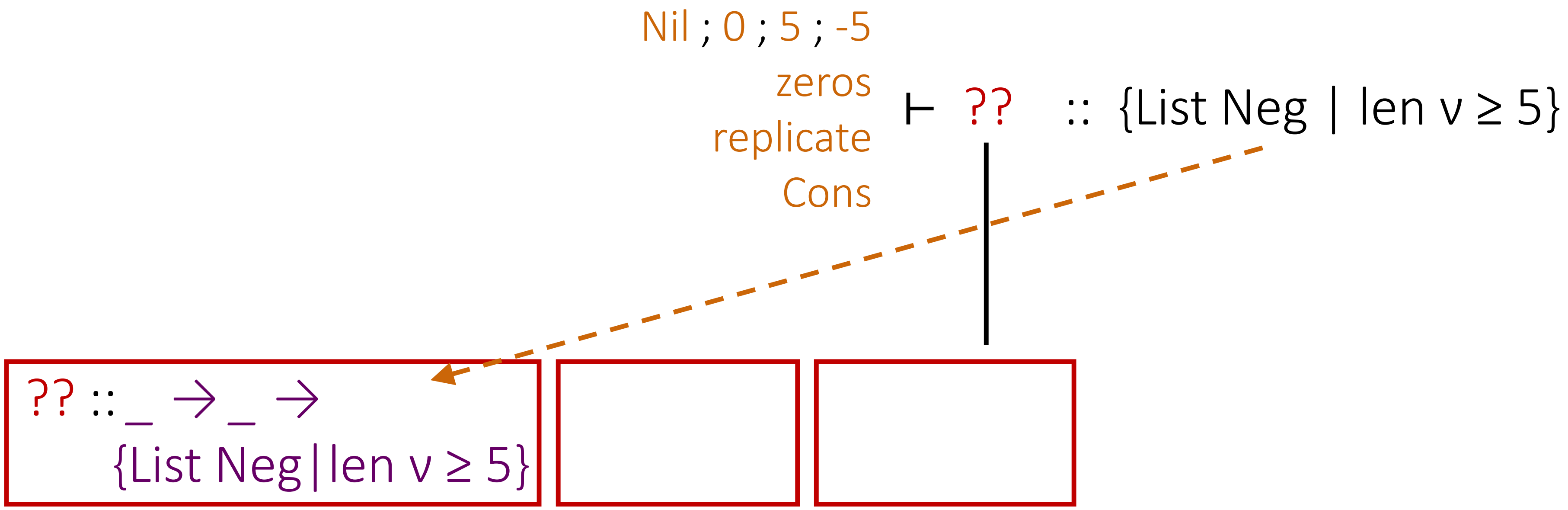
Round-trip type checking


Nil ; 0 ; 5 ; -5
zeros
replicate ; Cons
⊢ ?? :: {List Neg | len ν ≥ 5}
?? :: _ → {List Neg | len ν ≥ 5}

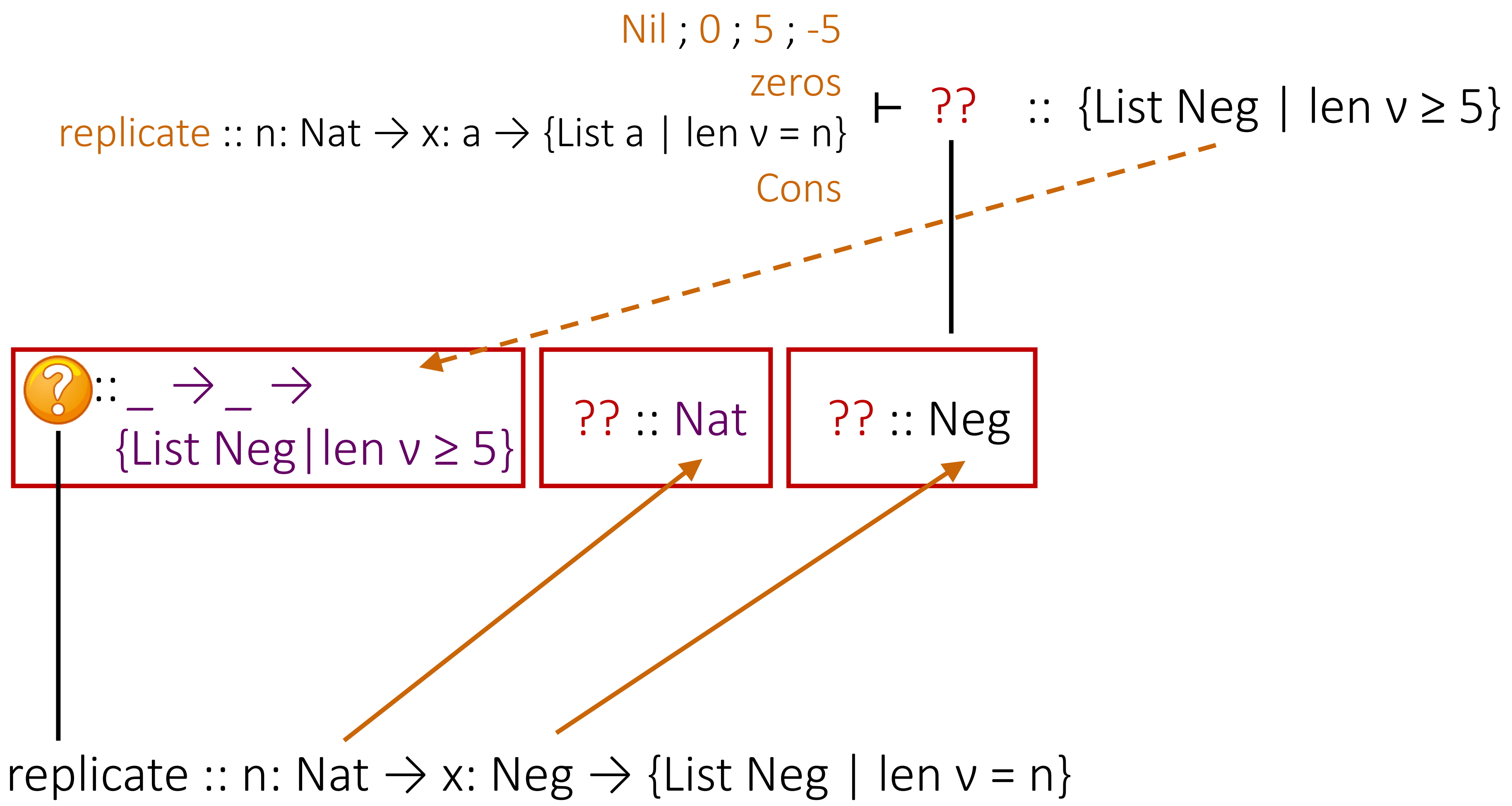
Round-trip type checking


Nil ; 0 ; 5 ; -5
zeros :: n:Nat → {List Zero | len ν = n}
replicate ; Cons
⊢ ?? :: {List Neg | len ν ≥ 5}
:: _ → {List Neg | len ν ≥ 5}
zeros :: n:Nat → {List Zero | len ν = n}

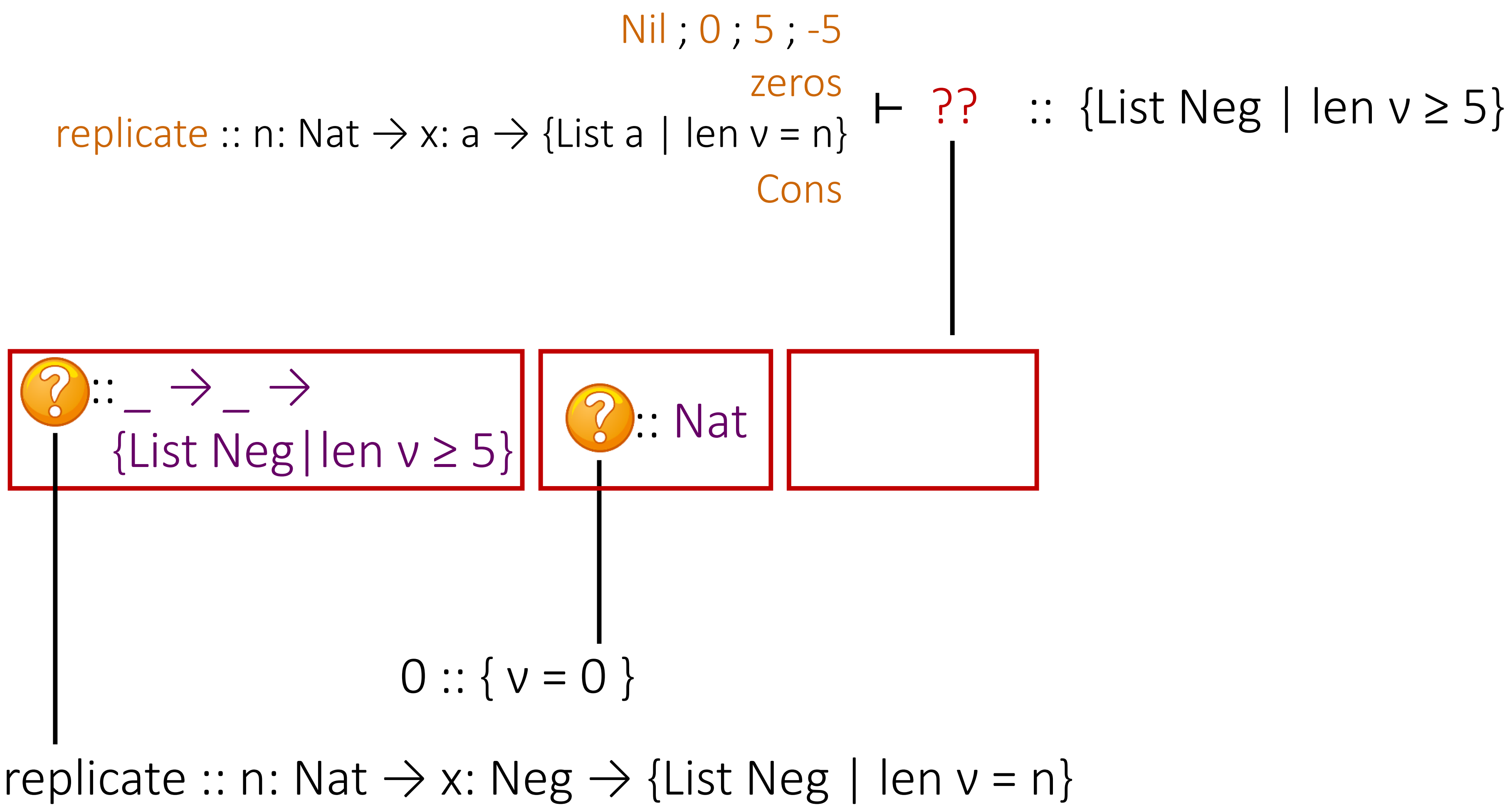
Round-trip type checking



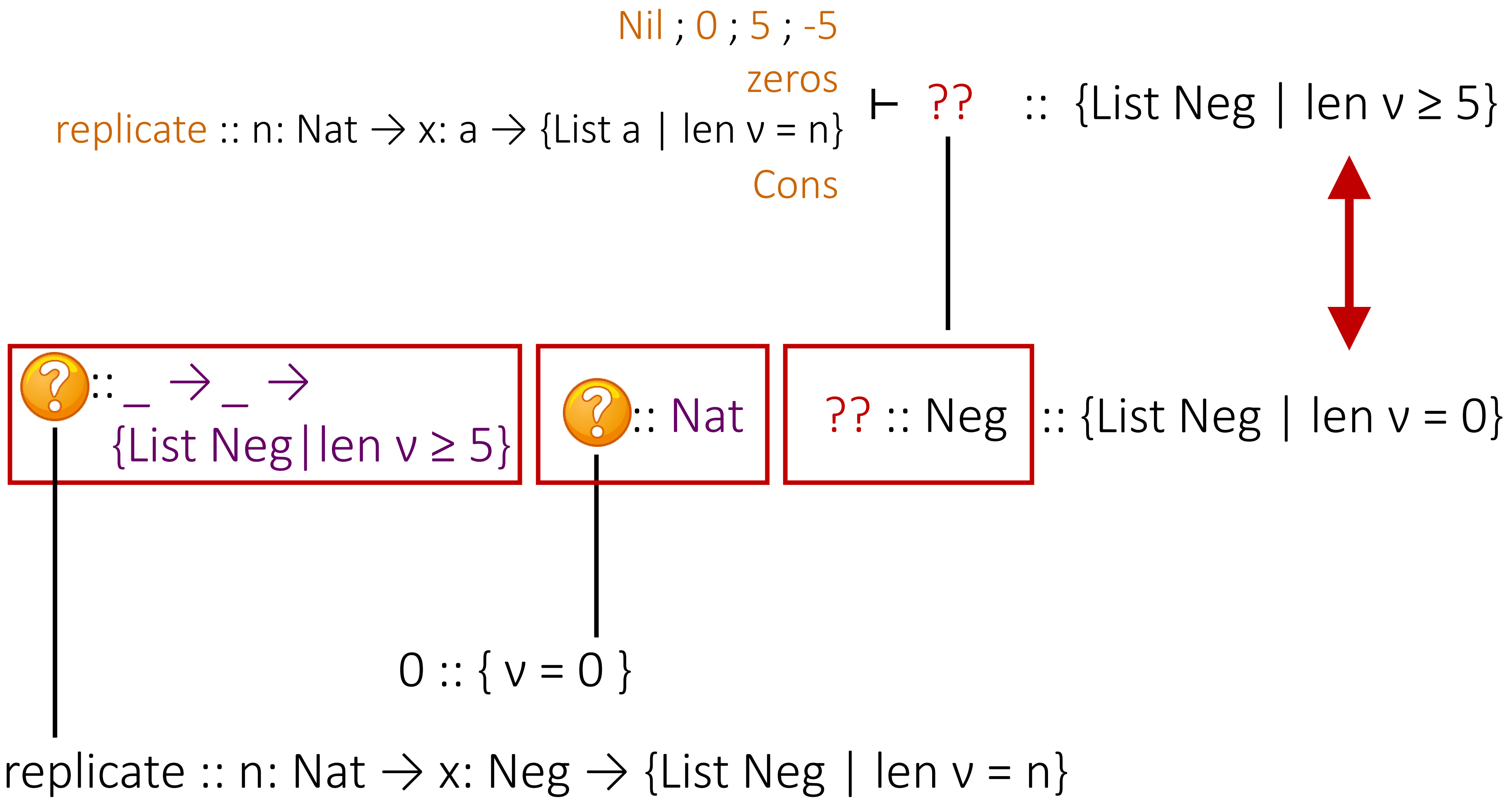
Round-trip type checking


Nil ; 0 ; 5 ; -5
zeros
replicate :: n: Nat → x: a → {List a | len ν = n}
Cons
⊢ ?? :: {List Neg | len ν ≥ 5}
? :: _ → _ → {List Neg | len ν ≥ 5}
?? :: Nat
?? :: Neg
replicate :: n: Nat → x: Neg → {List Neg | len ν = n}

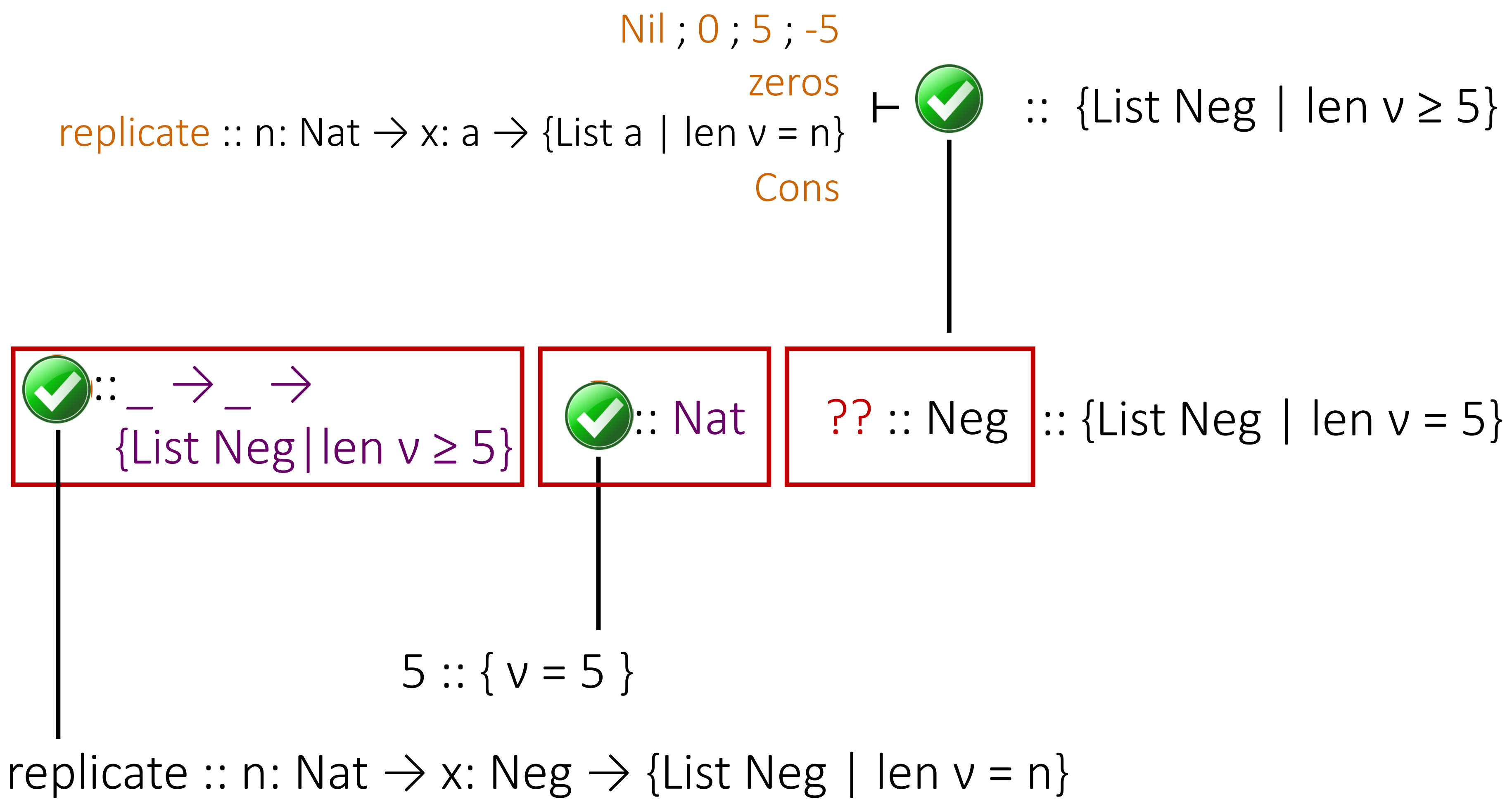
Round-trip type checking


Nil ; 0 ; 5 ; -5
zeros
replicate :: n: Nat → x: a → {List a | len v = n}
Cons
⊢ ?? :: {List Neg | len v ≥ 5}
:: _ → _ → {List Neg | len v ≥ 5}
:: Nat
0 :: { v = 0 }
replicate :: n: Nat → x: Neg → {List Neg | len v = n}

Round-trip type checking

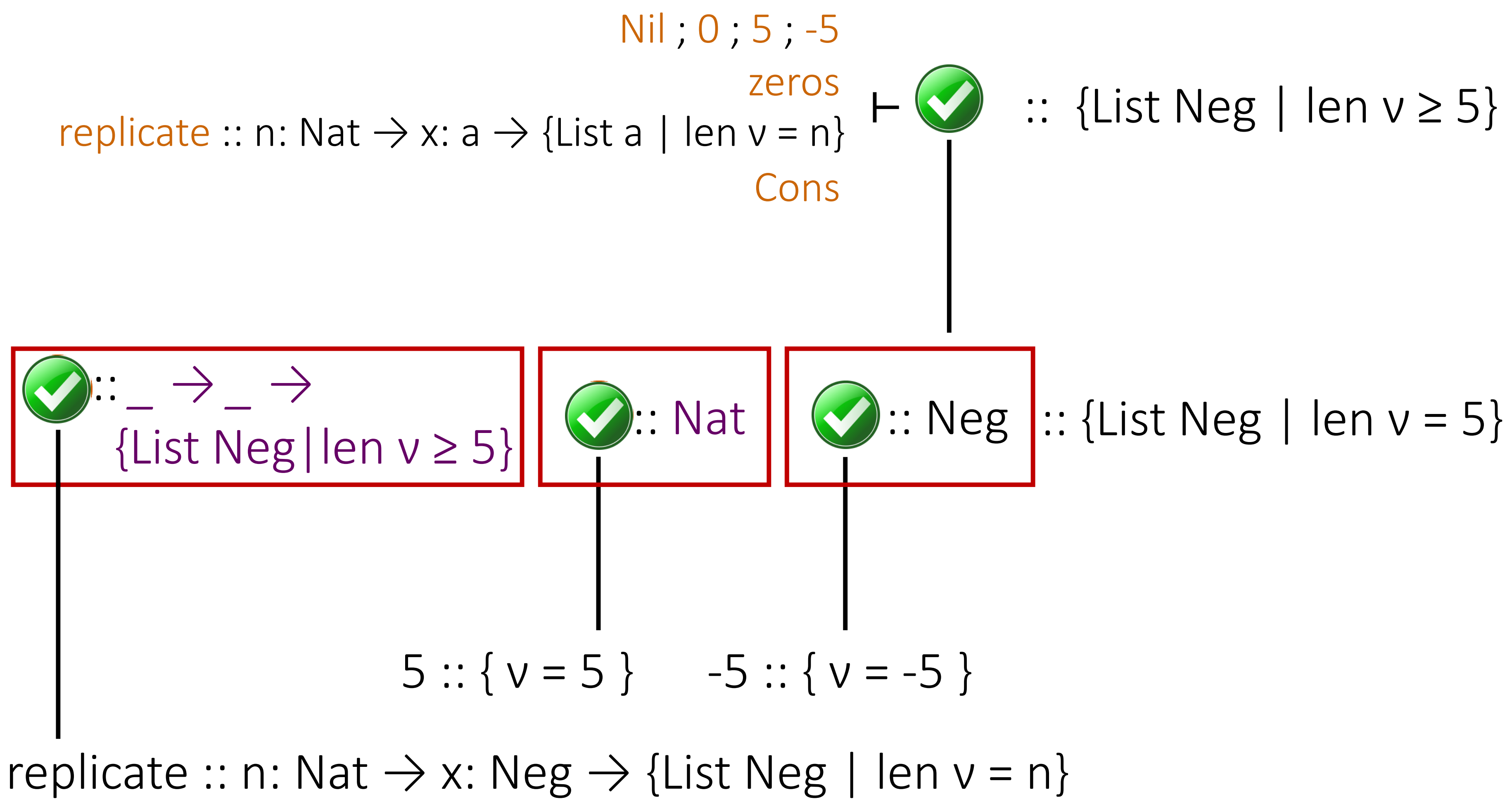

Nil ; 0 ; 5 ; -5
zeros
replicate :: n: Nat → x: a → {List a | len v = n}
Cons
⊢ ?? :: {List Neg | len v ≥ 5}
:: _ → _ →
{List Neg | len v ≥ 5}
:: Nat
?? :: Neg
:: {List Neg | len v = 0}
0 :: { v = 0 }
replicate :: n: Nat → x: Neg → {List Neg | len v = n}

Round-trip type checking



Nil ; 0 ; 5 ; -5
zeros
replicate :: n: Nat → x: a → {List a | len v = n}
Cons
⊢ ✓ :: {List Neg | len v ≥ 5}
✓ :: _ → _ → {List Neg | len v ≥ 5}
✓ :: Nat
?? :: Neg
:: {List Neg | len v = 5}
5 :: { v = 5 }
replicate :: n: Nat → x: Neg → {List Neg | len v = n}

Round-trip type checking


Nil ; 0 ; 5 ; -5
zeros
replicate :: n: Nat → x: a → {List a | len v = n}
Cons
⊢ ✓ :: {List Neg | len v ≥ 5}
✓ :: _ → _ → {List Neg | len v ≥ 5}
✓ :: Nat
✓ :: Neg
:: {List Neg | len v = 5}
5 :: { v = 5 }
-5 :: { v = -5 }
replicate :: n: Nat → x: Neg → {List Neg | len v = n}

Condition abduction

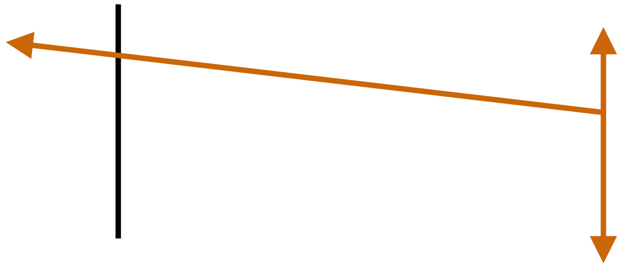
$$\begin{array}{r l}
\text{Nil} ; 0 ; -5 ; n :: \text{Nat} & \\
(\leq) ; (\neq) & \vdash \;\; ?? \;\; :: \{\text{List Neg} \mid \text{len } \nu = n\} \\
P &
\end{array}$$

Condition abduction

Nil ; 0 ; -5 ; n :: Nat

(≤) ; (≠) ⊢ ?? :: {List Neg | len ν = n}

n ≤ 0



Nil :: {List Neg | len ν = 0}

Condition abduction

$$\begin{array}{r l}
\text{Nil} ; 0 ; -5 ; n :: \text{Nat} & \\
(\le) ; (\ne) & \vdash ✓ :: \{\text{List Neg} \mid \text{len } v = n\} \\
n \le 0 &
\end{array}$$

if n ≤ 0 **then** Nil **else** $\boxed{\Gamma;\neg(n \le 0) \vdash ?? :: \{\text{List Neg} \mid \text{len } v = n\}}$

Liquid abduction

$$\begin{tikzcd}[column sep=small] n \geq 0 & \wedge & \text{len } v = 0 & \wedge \quad P \quad \Rightarrow \quad \text{len } v = n \\ n :: \text{Nat} \arrow[u] & & \\ & & \text{Nil} :: \{\text{List } a \mid \text{len } v = 0\} \arrow[uu] \end{tikzcd}$$

Liquid abduction

$$n \geq 0 \;\wedge\; \text{len } v = 0 \;\wedge\; \underset{\substack{\big|\\ \bigstar \leq \bigstar \\ \bigstar \neq \bigstar}}{\color{red}{P}} \;\wedge\; \neg(\text{len } v = n)$$

Liquid abduction

$$n \geq 0 \;\wedge\; \text{len } v = 0 \;\wedge\; P \;\Rightarrow\; \text{len } v = n$$

$n \leq 0$

$n \leq -5$

$-5 \leq n$

$n \neq 0$

$n \neq -5$

Liquid abduction

$$\text{UNSAT core}\left[\; n \geq 0 \;\land\; \text{len } v = 0 \;\land\; \underset{\Big|}{P} \;\land\; \neg(\text{len } v = n) \;\right]$$

$$n \leq 0$$

$$n \leq -5$$

$$-5 \leq n$$

$$n \neq 0$$

$$n \neq -5$$

Liquid abduction

UNSAT core $$\left[\boxed{n \geq 0} \land \boxed{\text{len } v = 0} \land P \land \boxed{\neg(\text{len } v = n)} \right]$$

$$\boxed{n \leq 0}$$

$$n \leq -5$$

$$-5 \leq n$$

$$n \neq 0$$

$$n \neq -5$$

Evaluation

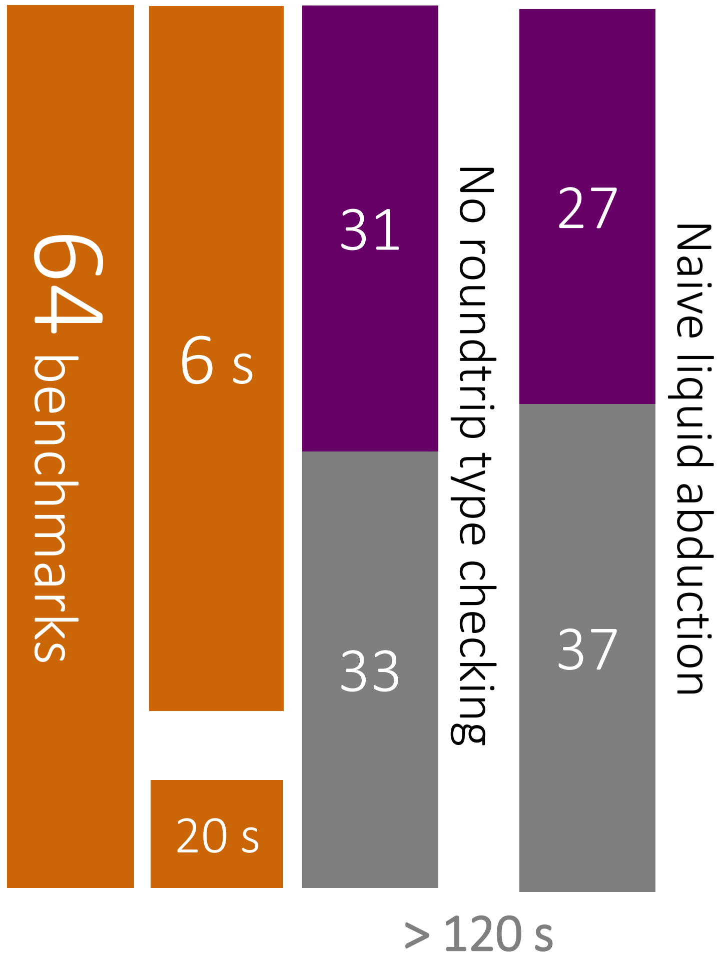
Lists
take, drop, delete, zip with, reverse, de-duplicate, fold, length/append with fold, ...

Sorting
insertion s., selection s., merge s., quick s.

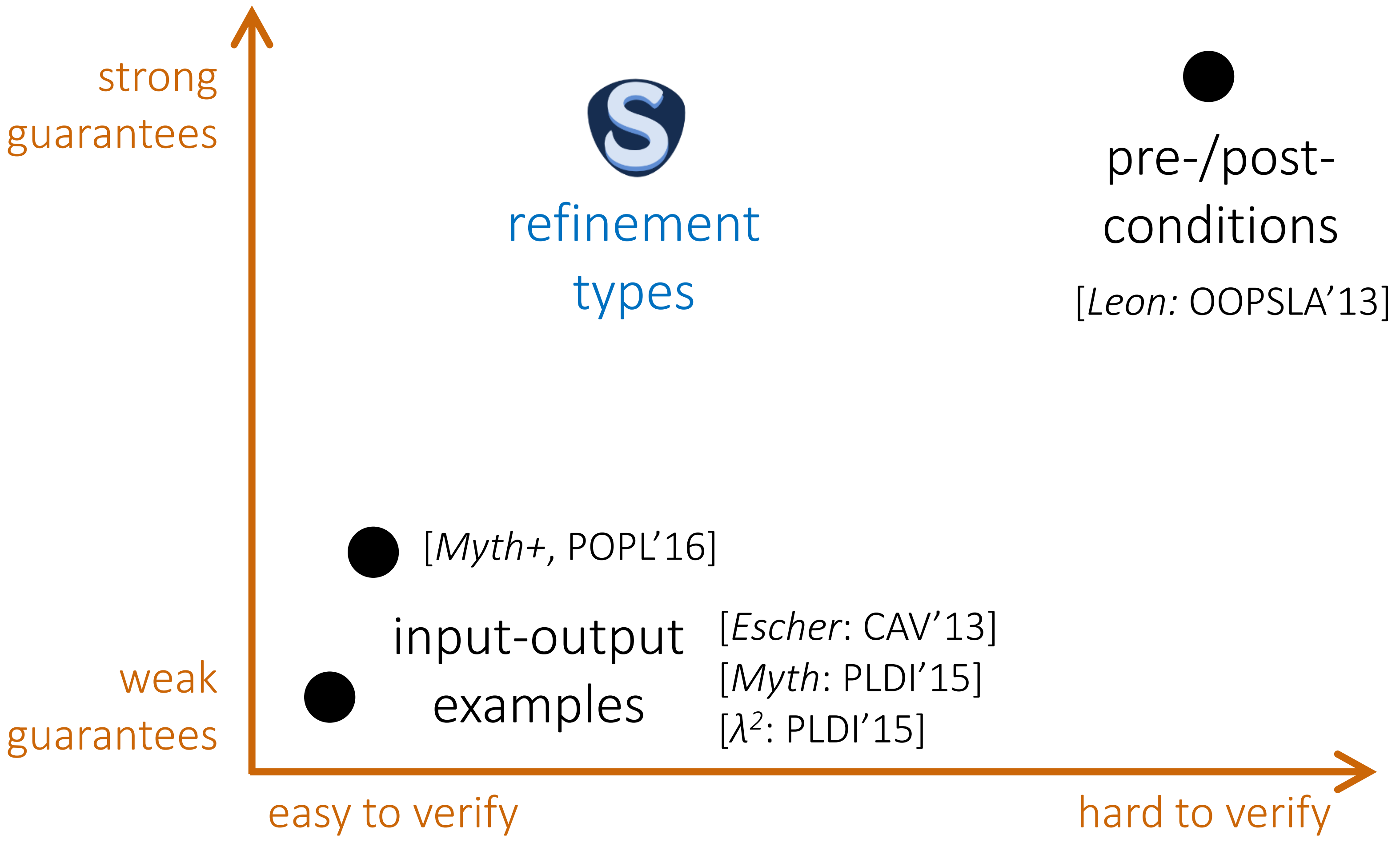
Binary Search Trees
member, insert, delete

Custom datatypes
AST desugaring, address book

Balanced trees
RBT & AVL insertion, AVL deletion



Synthesis of recursive programs



http://tiny.cc/synquid