Lecture 15
Refinement Types and Type-Driven Synthesis

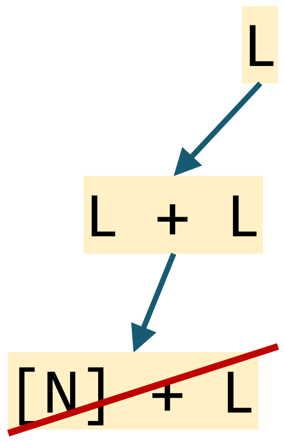
Nadia Polikarpova

Motivation

Goal: use deductive reasoning for top-down propagation

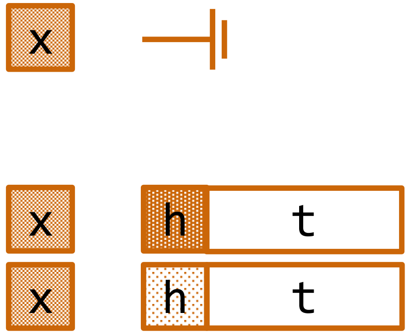
- prune unverifiable candidates early
- need synthesis-friendly verification technique!

Observation: type checkers are good at rejecting incomplete programs!

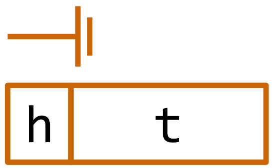


Running example

```
// Insert x into a sorted list xs
insert :: x:e  →  xs:List e  →  List e
insert x xs =
  match xs with
    Nil →
    Cons h t →
      if x ≤ h
        then Cons x xs
        else Cons h (insert x t)
```




```
data List e where
  Nil :: List e
  Cons :: h:e  →  t:List e  →  List e
```



Rejecting incomplete programs

[Pierce, Turner. TPLS'00]

```
// Insert x into a sorted list xs
insert :: x:e  →  xs:List e  →  List e
insert x xs =
  match xs with
    Nil → Cons xs ...
    ...
```

bidirectional type-checking!

Expected
e
and got
List e

Motivation

Goal: use deductive reasoning for top-down propagation

- prune unverifiable candidates early
- need synthesis-friendly verification technique!

Observation: type checkers are good at rejecting incomplete programs!

Idea: can we use types as behavioral constraints for synthesis?

L

L + L

~~[N] + L~~

Conventional types are not enough

```
// Insert x into a sorted list xs
insert :: x:e  →  xs:List e  →  List e
insert x xs =
  match xs with
    Nil → Nil
    Cons h t →
      if x ≤ h
        then Cons x xs
        else Cons h (insert x t)
```

Refinement types

[Rondon et al.'08, Kawaguchi et al.'09]

```
Nat
```

base types

```
max :: x: Int → y: Int → { v: Int | x ≤ v ∧ y ≤ v }
```

dependent function types

```
xs :: { v: List Nat }
```

polymorphic datatypes

```
data List α where
    Nil  ::  { List α | len v = 0 }
    Cons  ::  x: α  → { List α | len v = len xs + 1 }
```

```
measure len :: List α → Int
    len Nil = 0
    len (Cons _ xs) = len xs + 1
```

Refinement types

$$e \ ::= \text{true} \mid \text{false} \mid n \mid e + e$$
$$\mid x \mid e\ e \mid \lambda x{:}T.\, e$$

Terms

$$T \ ::= \{\nu{:}\ \mathrm{B} \mid \mathrm{e}\}$$ (basic types)
$$\mid x{:}T_1 \rightarrow T_2$$ (function types)
$$\mid \alpha$$ (type variables)

Types

$$S \ ::= T \mid \forall \alpha.\, S$$

Type schemas

T-num
$$\frac{(n = 0, 1, \dots)}{\Gamma \vdash n :: \{\nu{:}\, \mathrm{Int} \mid \nu = n\}}$$

T-var
$$\frac{(x{:}T \in \Gamma)}{\Gamma \vdash x :: \{\nu{:}\, T \mid \nu = x\}}$$

T-app
$$\frac{\Gamma \vdash e_1 :: x{:}T \rightarrow T' \qquad \Gamma \vdash e_2 :: T}{\Gamma \vdash e_1\ e_2 :: T'[x \mapsto e_2]}$$

Example

Let's check that $\Gamma \vdash \text{double } 5 :: \text{Nat}$

- $\text{Nat} = \{\nu : \text{Int} \mid \nu \geq 0\}$
- $\Gamma = [\text{double}:\ x : \text{Int} \rightarrow\ \{\nu : \text{Int} \mid \nu = 2 * x\}]$

T-num $$\frac{(n = 0, 1, \dots)}{\Gamma \vdash n :: \{\nu : \text{Int} \mid \nu = n\}}$$

T-abs $$\frac{\Gamma; x : T \vdash e :: T'}{\Gamma \vdash \lambda x : T.\, e :: T \rightarrow T'}$$

T-var $$\frac{(x : T \in \Gamma)}{\Gamma \vdash x :: \{\nu : T \mid \nu = x\}}$$

T-app $$\frac{\Gamma \vdash e_1 :: x : T \rightarrow T' \qquad \Gamma \vdash e_2 :: T}{\Gamma \vdash e_1\ e_2 :: T'[x \mapsto e_2]}$$

We need subtyping!

Subtyping

Intuitively, T' is a subtype of T if all values of type T' also belong to T

- written $T' <: T$
- e.g. $\mathrm{Nat} <: \mathrm{Int}$ or $\{\nu\colon \mathrm{Int} \mid \nu = 5\} <: \mathrm{Nat}$

Defined via inference rules:

$$\text{Sub-base}\;\frac{[\![\Gamma]\!] \land e' \Rightarrow e}{\Gamma \vdash \{v\colon B \mid e'\} <: \{v\colon B \mid e\}}$$

$$\text{Sub-fun}\;\frac{\Gamma \vdash T_1 <: T_1' \qquad \Gamma; x\colon T_1 \vdash T_2' <: T_2}{\Gamma \vdash x\colon T'_1 \to T_2' <: x\colon T_1 \to T_2}$$

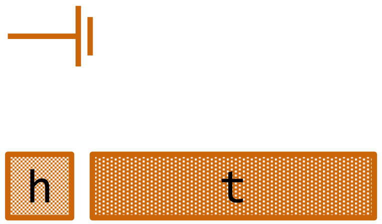
Conventional types are not enough

```
// Insert x into a sorted list xs
insert :: x:e  →  xs:List e  →  List e
insert x xs =
  match xs with
    Nil → Nil
    Cons h t →
      if x ≤ h
        then Cons x xs
        else Cons h (insert x t)
```

Refinement types

```
data SList e where
  Nil :: SList e
  Cons :: h:e →
          t:SList {ν:e | ν ≥ h} →
          SList e
```

sorted lists



Refinement types as specs

[Rondon et al. PLDI'08]

```
// Insert x into a sorted list xs
insert :: x:e  →  xs:SList e  →
          {ν:SList e | elems ν = elems xs ∪ {x}}
insert x xs =
  match xs with
    Nil → Nil
    Cons h t →
      if x ≤ h
        then Cons x xs
        else Cons h (insert x t)
```

Expected
{ν:SList e|elems ν = elems xs ∪ {x}}
and got
{ν:SList e|elems xs ⊆ elems ν}

Incomplete programs?

```
// Insert x into a sorted list xs
insert :: x:e  →  xs:SList e  →
          {v:SList e | elems v = elems xs ∪ {x}}
insert x xs =
  match xs with
    Nil → Nil
    Cons h t → ...
```

Bidirectional type checking

```
{ν:SList e | elems ν = {x}}
```

```
insert x xs =
  match xs with
    Nil → Nil
    Cons h t → ...
```

Round-trip type checking

{ν:e | ν ≥ h}

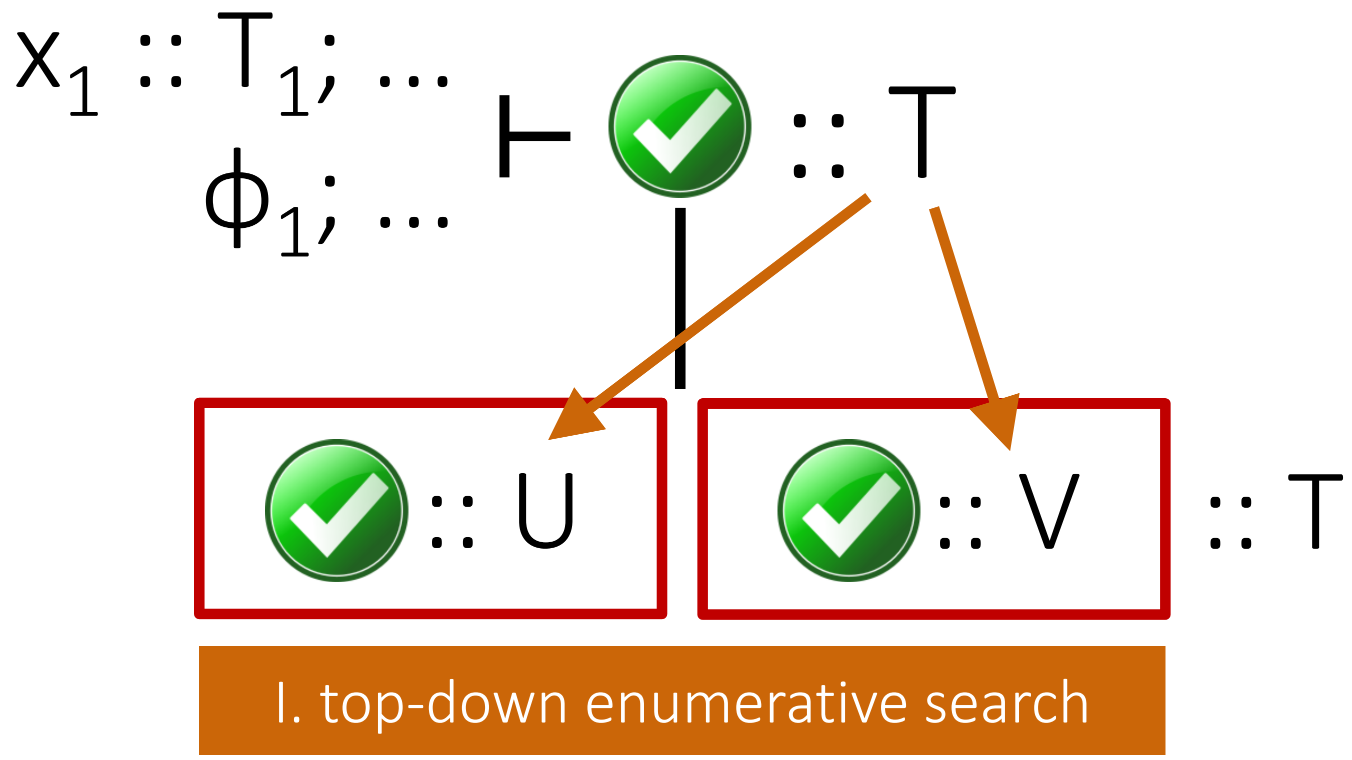
```
insert x xs =
  match xs with
    Nil → Cons x Nil
    Cons h t →
      Cons h (insert x ...)
```

h (insert x ...)

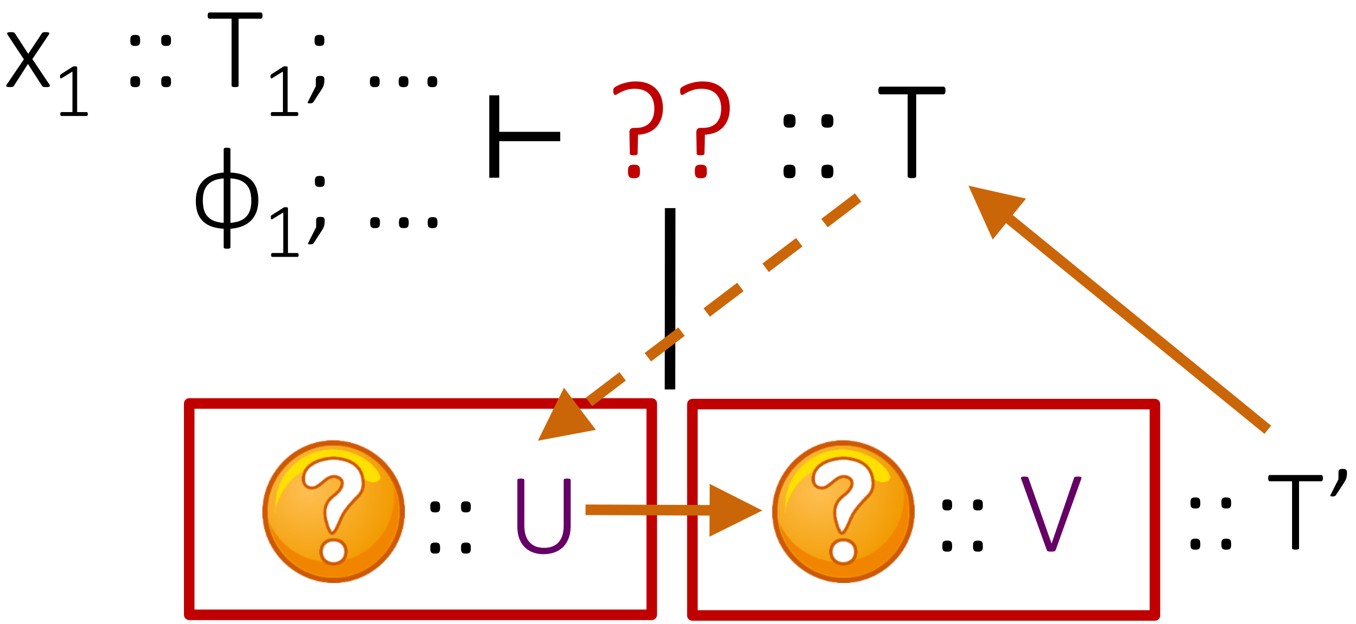
Type-driven Synthesis

http://tiny.cc/synquid

Synthesis from refinement types



Synthesis from refinement types



I. top-down enumerative search

II. round-trip type checking

Synthesis from refinement types

$$\begin{matrix} x_1 :: T_1; \ldots \\ \phi_1; \ldots \end{matrix} \vdash {\color{red}??} :: T$$

if $\boxed{{\color{red}??} :: \{Bool \mid v = P\}}$ then $\boxed{{\color{red}P} \vdash \; :: T}$ else $\boxed{\neg P \vdash {\color{red}??} :: T}$

I. top-down enumerative search

II. round-trip type checking

III. condition abduction

Round-trip type checking

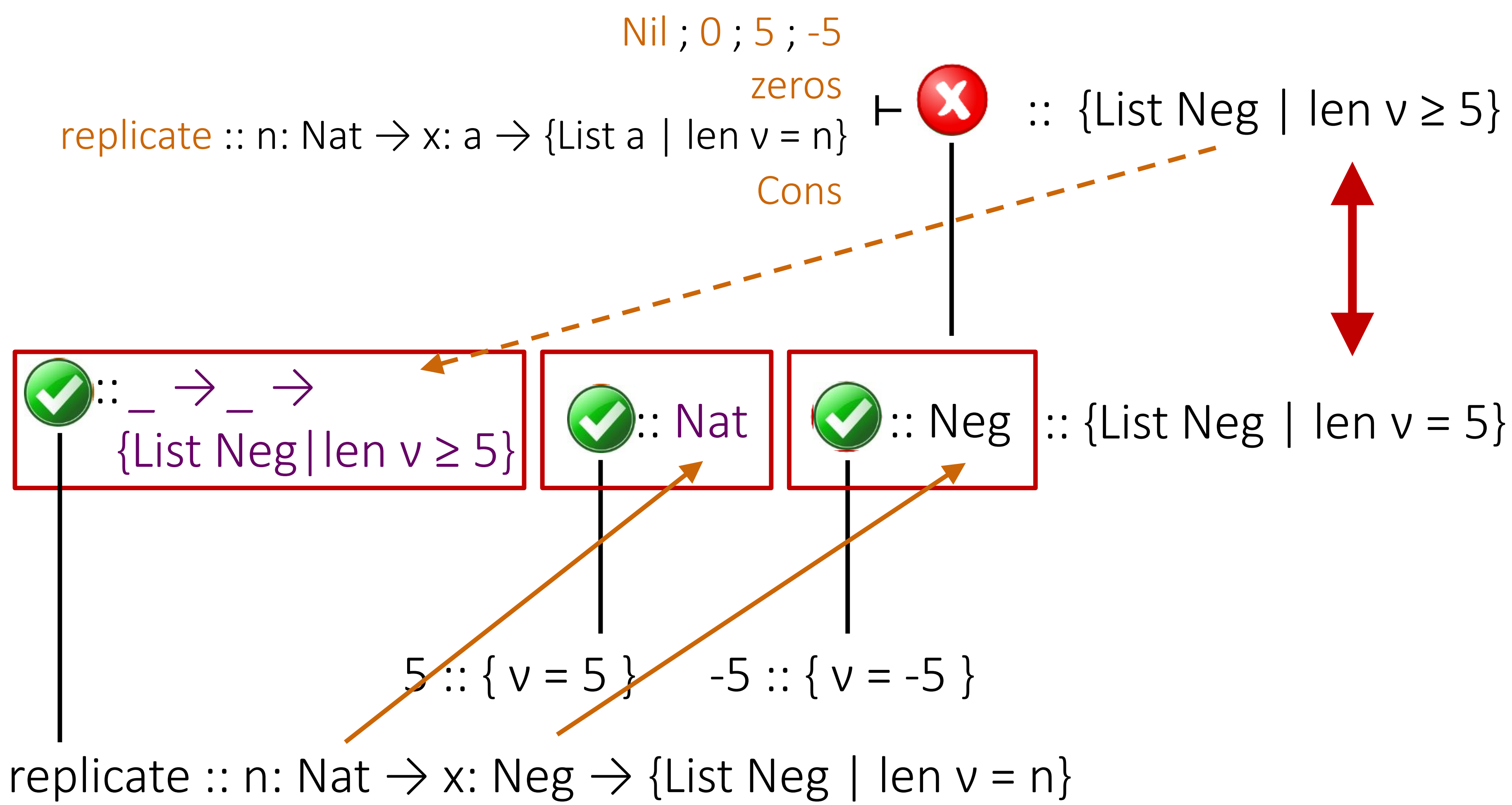
Nil :: {List a | l
zeros :: n:Nat → {l

$\Gamma \vdash$ :: {List Neg | len v ≥ 5}

:: _ → {List Neg | len v ≥ 5} √ l :: {List Neg|len v = 0}

zeros :: n:Nat → {List Zero | len v = n}

Round-trip type checking



Condition abduction

$\mathit{Nil} \; ; \; 0 \; ; \; -5 \; ; \; n :: \mathit{Nat}$

$(\leq) \; ; \; (\neq) \;\; \vdash \;\; :: \; \{\mathit{List\ Neg} \mid \mathit{len}\ v = n\}$

$n \leq 0$

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

Liquid abduction

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

$n :: \text{Nat}$

$\text{Nil} :: \{\text{List } a \mid \text{len } v = 0\}$

$\boxed{n \leq 0}$

$n \leq -5$

$-5 \leq n$

$\boxed{n \neq 0}$

$n \neq -5$