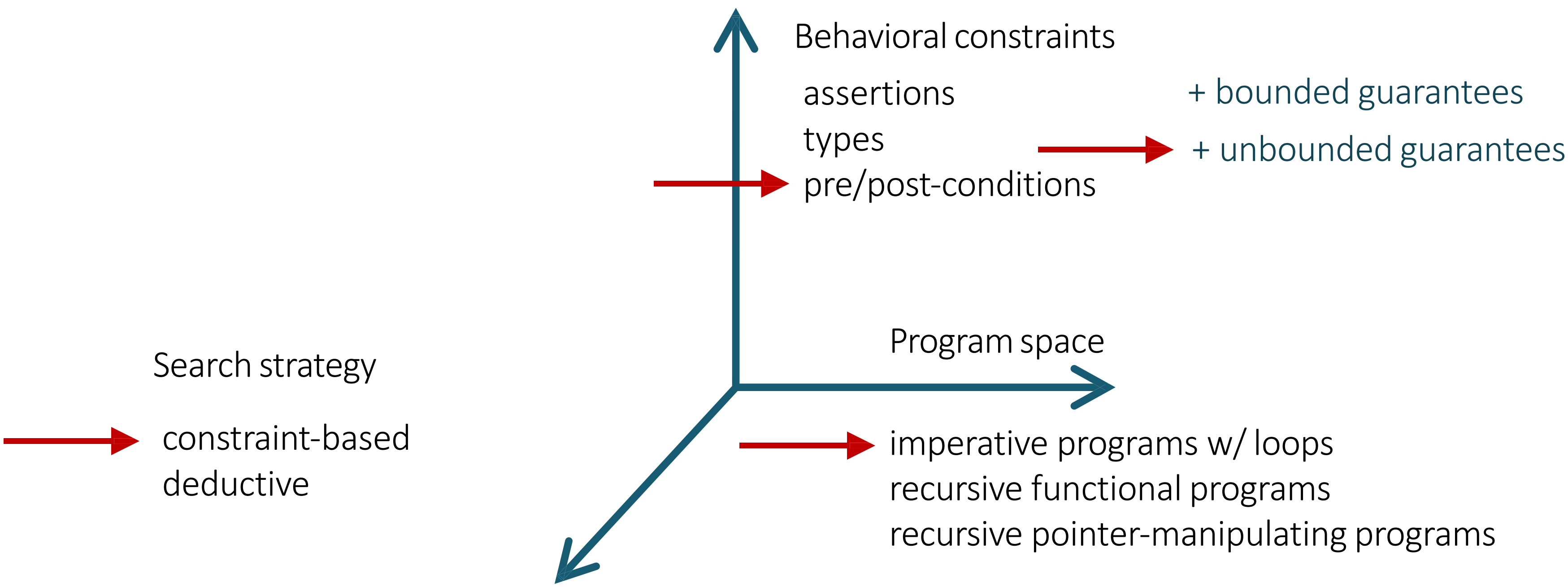
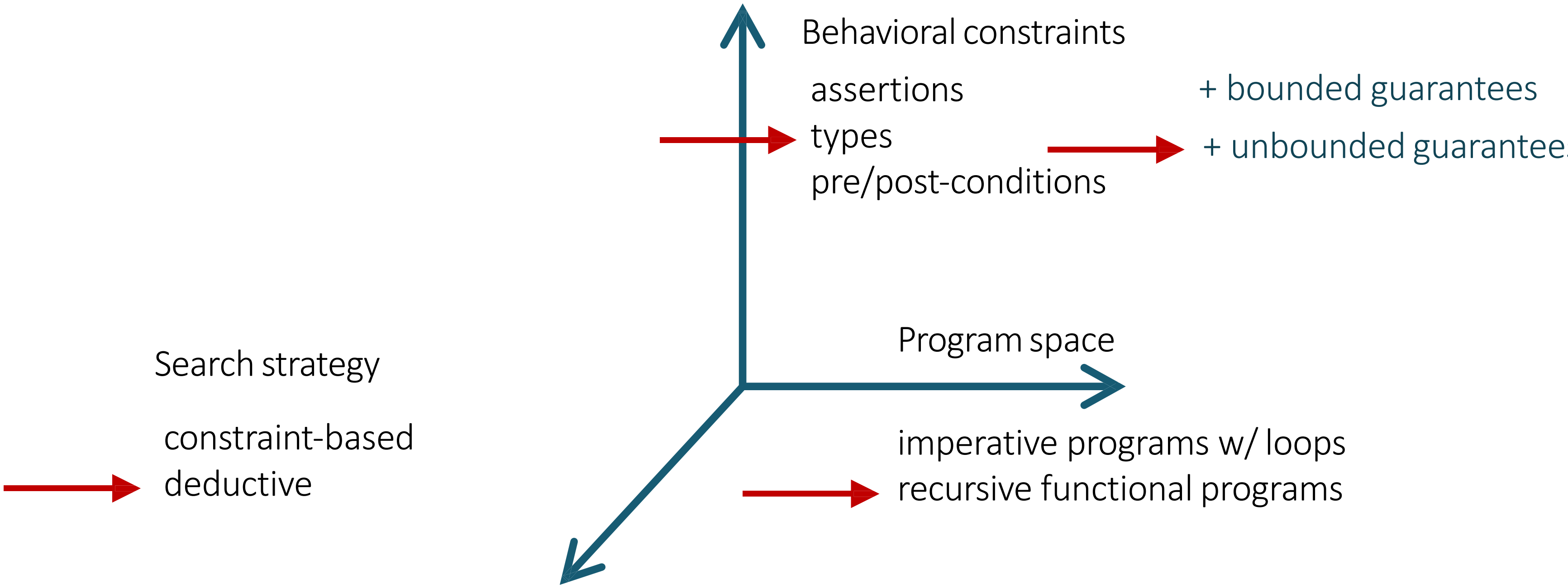
Lecture 12
Type Systems

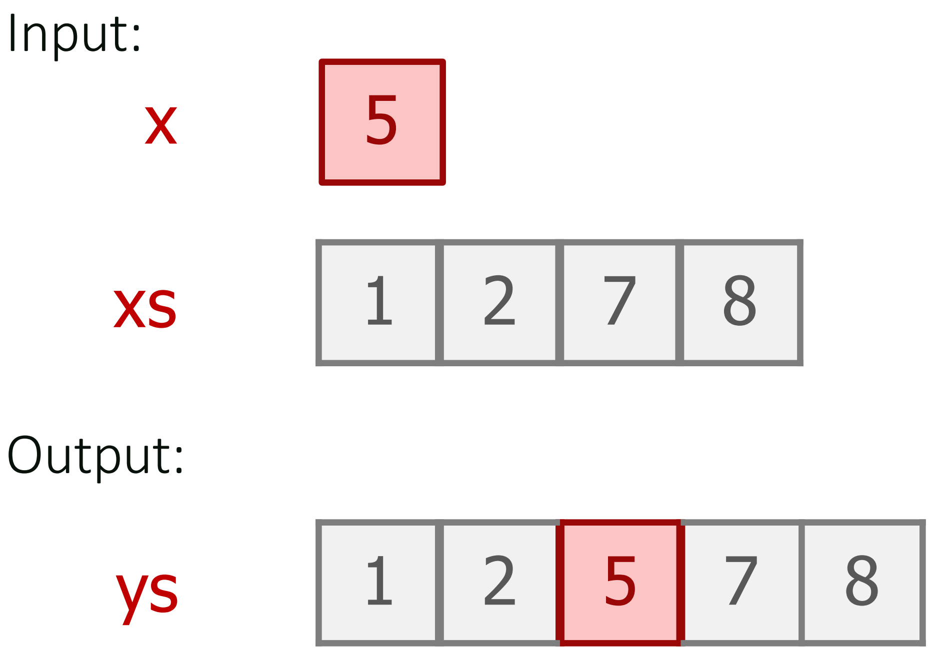
Last week


Behavioral constraints
assertions
types
pre/post-conditions
+ bounded guarantees
+ unbounded guarantees
Program space
imperative programs w/ loops
recursive functional programs
recursive pointer-manipulating programs
Search strategy
constraint-based
deductive

This week


Behavioral constraints
assertions
types
pre/post-conditions
+ bounded guarantees
+ unbounded guarantees
Program space
imperative programs w/ loops
recursive functional programs
Search strategy
constraint-based
deductive

Example: insert into a sorted list



In a functional language

```
insert x xs =
  match xs with
    Nil →
      Cons x Nil
    Cons h t →
      if x ≤ h
        then Cons x xs
        else Cons h (insert x t)
```

Specification for insert

Input:

x

xs: sorted list

Output:

ys: sorted list

elems ys = elems xs ∪ {x}

How can I express this formally?

Types!

How can I verify this for all inputs?

Type checking!

Agenda

Today:

- Simple types and how to check them
- Refinement types and how to check them

Later:

- Specification for insert as a refinement type
- How to use refinement type checking for synthesis?

What is a type system?

Formalization of a typing discipline of a language

- independently of a particular type checking algorithm
- if a type checking algorithm exists, type system is *decidable*

Deductive system for proving facts about programs and types

- defined using *inference rules* over *judgments*

environment / context
(declares free variables of $\mathfrak{J}$)

$$\Gamma \vdash \mathfrak{J}$$

assertion
for example:

typically:

$x_1 : T_1, \ldots, x_n : T_n$

$e :: T$	"e has type T"
T	"T is well-formed"
$T' <: T$	"T' is a subtype of T"

Simple type system

$e ::= \text{true} \mid \text{false} \mid n \mid e + e$ Syntax of terms (programs)

$T ::= \text{Bool} \mid \text{Int}$ Syntax of types

Inference Rules

T-true $\dfrac{}{\Gamma \vdash \text{true} :: \text{Bool}}$

T-false $\dfrac{}{\Gamma \vdash \text{false} :: \text{Bool}}$

T-num $\dfrac{(n = 0, 1, \ldots)}{\Gamma \vdash n :: \text{Int}}$

label → T-plus $\dfrac{\Gamma \vdash e_1 :: \text{Int} \qquad \Gamma \vdash e_2 :: \text{Int}}{\Gamma \vdash e_1 + e_2 :: \text{Int}}$ ← premises ← conclusion

Type derivations

$\emptyset \vdash 1 + 2 :: \text{Int}$ is a valid judgment, because....

$$\text{T-plus}\;\dfrac{\text{T-num}\;\dfrac{}{\emptyset \vdash 1 :: \text{Int}} \qquad \text{T-num}\;\dfrac{}{\emptyset \vdash 2 :: \text{Int}}}{\emptyset \vdash 1 + 2 :: \text{Int}}$$

We say that $1 + 2$ is *well-typed* (and has type Int)

Type derivations

$\emptyset \vdash 1 + \mathrm{true} :: \ \mathrm{Int}$ is not a valid judgment, because....

$$
\dfrac{\dfrac{}{\emptyset \vdash 1 \ :: \mathrm{Int}}\ \text{T-num} \qquad \emptyset \vdash \mathrm{true} :: \ \mathrm{Int}}{\emptyset \vdash 1 + \mathrm{true} :: \ \mathrm{Int}}\ \text{T-plus}
$$

We say that $1 + \mathrm{true}$ is *ill-typed* (or *not typable*)

Type checking vs inference

The problem of discovering the derivation of $\Gamma \vdash e :: T$ is called *type checking*

The problem of discovering the type T such that there exists a derivation of $\Gamma \vdash e :: T$ is called *type inference*

If we have a mechanism for inference, we can also do checking

Function types

$e \; ::= \text{true} \mid \text{false} \mid n \mid e + e$ Syntax of terms (programs)

$\quad \mid x \mid e\; e \mid \lambda x.\, e$ (variable, application, lambda abstraction)

$T \; ::= \text{Bool} \mid \text{Int}$ (basic types) Syntax of types

$\quad \mid T_1 \rightarrow T_2$ (function types)

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

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

T-app $$\frac{\Gamma \vdash e_1 :: T \rightarrow T' \qquad \Gamma \vdash e_2 :: T}{\Gamma \vdash e_1\, e_2 :: T'}$$

Exercise 1

Infer the type of $\lambda x.\, inc\ x$ in $\Gamma = [inc\colon \mathrm{Int} \to \mathrm{Int}]$ using the rules

T-num $\dfrac{(n = 0, 1, \ldots)}{\Gamma \vdash n :: \mathrm{Int}}$

T-plus $\dfrac{\Gamma \vdash e_1 :: \mathrm{Int} \qquad \Gamma \vdash e_2 :: \mathrm{Int}}{\Gamma \vdash e_1 + e_2 :: \mathrm{Int}}$

T-var $\dfrac{(x\colon T \in \Gamma)}{\Gamma \vdash x :: T}$

T-abs $\dfrac{\Gamma; x\colon T \vdash e :: T'}{\Gamma \vdash \lambda x\colon T.\, e :: T \to T'}$

T-app $\dfrac{\Gamma \vdash e_1 :: T \to T' \qquad \Gamma \vdash e_2 :: T}{\Gamma \vdash e_1\, e_2 :: T'}$

Inference algorithm

Goal: compute the type of term from the types of subterms

$$\dfrac{\dfrac{}{\Gamma \vdash inc\ ::}\ \text{T-var} \qquad \dfrac{}{\Gamma \vdash 1\ ::}\ \text{T-num}}{\Gamma \vdash inc\ 1\ ::}\ \text{T-app}$$

$\Gamma = [inc\colon \mathrm{Int} \to \mathrm{Int}]$

Inference algorithm

Problem: to compute the types of this term,
we had to *guess* the type of x:

$$\text{T-abs}\;\dfrac{\text{T-app}\;\dfrac{\text{T-var}\;\dfrac{}{\Gamma, x: ? \vdash inc ::} \qquad \dfrac{}{\Gamma, x: ? \vdash x ::}\;\text{T-var}}{\Gamma, x: ? \vdash inc\ x ::}}{\Gamma \vdash \lambda x.\ inc\ x\ ::}$$

Solution: constraint-based type inference

- aka Hindley-Milner type inference

Constraint-based type inference

[Hindley'69][Milner'78]

Idea: separate inference into constraint generation and constraint solving

1. Whenever you need to guess a type, generate a *type variable*
2. Whenever two types must match, generate a *unification constraint*
3. Solve unification constraints to assign types to type variables

Example

Type derivation

$$\dfrac{\dfrac{}{\Gamma, x{:}\ \alpha \vdash inc :: \mathrm{Int} \to \mathrm{Int}}\ \text{T-var} \qquad \dfrac{}{\Gamma, x{:}\ \alpha \vdash x :: \alpha}\ \text{T-var}}{\dfrac{\Gamma, x{:}\ \alpha \vdash inc\ x ::}{\Gamma \vdash \lambda x.\ inc\ x ::}\ \text{T-abs}}\ \text{T-app}$$

$$\Gamma = [inc{:}\ \mathrm{Int} \to \mathrm{Int}]$$

Example

Type derivation

$$\dfrac{\dfrac{}{\Gamma, x\!: \alpha \vdash inc :: \mathrm{Int} \to \mathrm{Int}}\,\text{T-var} \qquad \dfrac{}{\Gamma, x\!: \alpha \vdash x :: \alpha}\,\text{T-var}}{\dfrac{\Gamma, x\!: \alpha \vdash inc\ x :: \mathrm{Int}}{\Gamma \vdash \lambda x.\, inc\ x ::}\,\text{T-abs}}\,\text{T-app}$$

Type assignment

$$\alpha \;\to\; \mathrm{Int}$$

Unification constraints

$$\alpha \sim \mathrm{Int}$$

$$\Gamma = [inc\!: \mathrm{Int} \to \mathrm{Int}]$$

Example

Type derivation

$$\dfrac{\dfrac{}{\Gamma, x{:}\ \alpha \vdash inc :: \text{Int} \to \text{Int}}\ \text{T-var} \qquad \dfrac{}{\Gamma, x{:}\ \alpha \vdash x :: \alpha}\ \text{T-var}}{\dfrac{\Gamma, x{:}\ \alpha \vdash inc\ x :: \text{Int}}{\Gamma \vdash \lambda x.\ inc\ x :: \alpha \to \text{Int}}\ \text{T-abs}}\ \text{T-app}$$

Type assignment

$$\alpha \ \to\ \text{Int}$$

Unification constraints

$$\alpha \sim \text{Int}$$

$$\Gamma = [inc{:}\ \text{Int} \to \text{Int}]$$

Example

Type derivation

$$\dfrac{\dfrac{}{\Gamma, x{:}\,\alpha \vdash inc :: \mathrm{Int} \to \mathrm{Int}}\;\text{T-var} \qquad \dfrac{}{\Gamma, x{:}\,\alpha \vdash x :: \alpha}\;\text{T-var}}{\dfrac{\Gamma, x{:}\,\alpha \vdash inc\ x :: \mathrm{Int}}{\Gamma \vdash \lambda x.\, inc\ x :: \alpha \to \mathrm{Int}}\;\text{T-abs}}\;\text{T-app}$$

$$\mathrm{Int} \to \mathrm{Int}$$

Type assignment

$$\alpha \;\to\; \mathrm{Int}$$

Unification constraints

$$\alpha \sim \mathrm{Int}$$

$$\Gamma = [inc{:}\ \mathrm{Int} \to \mathrm{Int}]$$

Bidirectional type-system

[Pierce, Turner'00]

Rules differentiate between type inference and checking

$$\Gamma \vdash e \uparrow T \qquad\qquad \Gamma \vdash e \downarrow T$$

"e generates T in Γ" "e checks against T in Γ"

I-var $\dfrac{(x{:}\, T \in \Gamma)}{\Gamma \vdash x \uparrow T}$ C-abs $\dfrac{\Gamma; x{:}\, T_1 \vdash e \downarrow T_2}{\Gamma \vdash \lambda x.\, e \downarrow T_1 \rightarrow T_2}$ C-I $\dfrac{\Gamma \vdash e \uparrow T' \quad \Gamma \vdash T \sim T'}{\Gamma \vdash e \downarrow T}$

Bidirectional type-system

Type derivation

$$\dfrac{\dfrac{}{\Gamma, x{:}\,\mathrm{Int} \vdash inc \uparrow \mathrm{Int} \to \mathrm{Int}}\text{T-var} \qquad \dfrac{}{\Gamma, x{:}\,\mathrm{Int} \vdash x \uparrow \mathrm{Int}}\text{T-var}}{\dfrac{\Gamma, x{:}\,\mathrm{Int} \vdash inc\ x \downarrow \mathrm{Int}}{\Gamma \vdash \lambda x.\, inc\ x \downarrow \mathrm{Int} \to \mathrm{Int}}\text{T-abs}}\text{T-app}$$

Type assignment

Unification constraints

$$\mathrm{Int} \sim \mathrm{Int}$$

$$\Gamma = [inc{:}\ \mathrm{Int} \to \mathrm{Int}]$$

Polymorphism (aka "generics")

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

Terms

$$T ::= \text{Bool} \mid \text{Int}$$ (basic types)
$$\mid T_1 \rightarrow T_2$$ (function types)
$$\mid \alpha$$ (type variables)

Types

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

Type schemas

T-gen
$$\frac{\Gamma; \alpha \vdash e :: S}{\Gamma \vdash e :: \forall \alpha.\, S}$$

T-inst
$$\frac{\Gamma \vdash e :: \forall \alpha.\, S \qquad \Gamma \vdash T}{\Gamma \vdash e :: S[\alpha \mapsto T]}$$

Exercise 3

Let's infer the type of $id\;5$ in Γ
where $\Gamma = [\mathrm{id} : \forall\alpha.\alpha \rightarrow \alpha]$
using the following rules:

T-num $\dfrac{(n = 0, 1, \ldots)}{\Gamma \vdash n :: \mathrm{Int}}$

T-var $\dfrac{(x : T \in \Gamma)}{\Gamma \vdash x :: T}$

T-app $\dfrac{\Gamma \vdash e_1 :: T \rightarrow T' \quad \Gamma \vdash e_2 :: T}{\Gamma \vdash e_1\, e_2 :: T'}$

T-gen $\dfrac{\Gamma; \alpha \vdash e :: S}{\Gamma \vdash e :: \forall\alpha. S}$

T-inst $\dfrac{\Gamma \vdash e :: \forall\alpha. S \quad \Gamma \vdash T}{\Gamma \vdash e :: S[\alpha \mapsto T]}$

Agenda

Today:

- Simple types and how to check them
- Refinement types and how to check them
- Specification for insert as a refinement type

Thursday:

- How to use refinement type checking for synthesis?

Types as specifications

```
insert :: ∀ a . a → List a → List a
```

Conventional types are not enough

```
// Insert x into a sorted list xs
insert :: x:a  →  xs:List a  →  List a
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

Problem: intersection of strictly sorted lists

- example: intersect [4, 8, 15, 16, 23, 42] [8, 16, 32, 64] ⟶ [8, 16]

Also: we want a guarantee that it's correct on all inputs!

```
insert :: x:a →
  ys:SList a  →
  {ν:SList a | elems ν = elems xs ∩
                         elems ys}
```

Refinement types

[Rondon et al.'08]

```
Nat = { v: Int | 0 ≤ v }
```

base types

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

dependent function types

```
xs :: { v: List Nat | len v=5 }
```

polymorphic datatypes

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

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

```
data SList α where
    Nil  ::  { List α | len v = 0 }
    Cons  ::  x: α -> xs: SList {α | _v >= x} -> {List α|
                    len v = len xs + 1}
```

Refinement types

binary search tree

red nodes have
black children

```
data RBT a where
  Empty :: RBT a
  Node  :: x: a ->
    black: Bool ->
    left:  { RBT {a | _v < x} | !black ==> isBlack _v } ->
    right: { RBT {a | x < _v} | !black ==> isBlack _v  &&
                blackHeight _v == blackHeight left } ->
    RBT a

insert :: x: a -> t: RBT a -> {RBT a | elems _v == elems t + [x]}
insert = ??
```

same number of
black nodes on
every path to leaves

Refinement types

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

Terms

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

Types

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

Type schemas

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

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

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

Example

Let’s check that $\Gamma \vdash \text{inc } 5 :: \text{Nat}$

- $\text{Nat} = \{\nu : \text{Int} \mid \nu \geq 0\}$
- $\Gamma = [\text{inc}: y : \text{Int} \to \{\nu : \text{Int} \mid \nu = y + 1\}]$

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

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

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

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

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 $\{\nu : \mathrm{Int} \mid \nu = 5\} <: \mathrm{Nat}$

Defined via inference rules:

$$\text{Sub-base}\ \frac{[\![\Gamma]\!] \wedge e' \Rightarrow e}{\Gamma \vdash \{\nu : B \mid e'\} <: \{\nu : B \mid e\}}$$

$$\text{Sub-fun}\ \frac{\Gamma \vdash T_1 <: T'_1 \qquad \Gamma; x : T_1 \vdash T'_2 <: T_2}{\Gamma \vdash x : T'_1 \to T'_2 <: x : T_1 \to T_2}$$

Refinement type inference

[Rondon et al.'08]

Idea: separate inference into (subtyping) constraint generation and (subtyping) constraint solving

1. Whenever you need to guess a type, generate a *type variable*
2. Whenever two types must match, generate a *subtyping constraint*
3. Solve subtyping constraints to assign refined types to type variables

$$\Gamma \vdash \lambda x.\,\mathrm{inc}\ x :: \mathrm{Nat} \to \mathrm{Nat}$$

Example

Type derivation

$$\dfrac{\dfrac{\text{T-var}\quad}{\Gamma, x{:}\,\alpha \vdash inc :: y{:}\,\mathrm{Int} \to \{\nu{:}\,\mathrm{Int} \mid \nu = y + 1\}} \qquad \dfrac{}{\Gamma, x{:}\,\alpha \vdash x :: \alpha}\ \text{T-var}}{\dfrac{\Gamma, x{:}\,\alpha \vdash inc\ x :: \{\nu{:}\,\mathrm{Int} \mid \nu = x + 1\}}{\Gamma \vdash \lambda x.\, inc\ x :: x{:}\alpha \to \{\nu{:}\,\mathrm{Int} \mid \nu = x + 1\}}\ \text{T-abs}}\ \text{T-app}$$

$$\mathrm{Nat} \to \mathrm{Nat}$$

Subtyping constraints

$$\alpha <:\ \mathrm{Int}$$

$$\cancel{x{:}\alpha \to \{\nu{:}\,\mathrm{Int} \mid \nu = x + 1\} <: \mathrm{Nat} \to \mathrm{Nat}}$$

$$\mathrm{Nat} <: \alpha$$

$$x{:}\,\mathrm{Nat} \vdash \{\nu{:}\,\mathrm{Int} \mid \nu = x + 1\} <: \mathrm{Nat}$$

Type assignment

$$\alpha \ \to \{\nu{:}\,\mathrm{Int} \mid P\}$$

Horn clauses

$$P \Rightarrow true$$

$$\nu \geq 0 \Rightarrow P$$

$$x \geq 0 \wedge \nu = x + 1 \Rightarrow \nu \geq 0$$

Ask Z3: $P \to true$

$$\Gamma = [inc{:}\ y{:}\,\mathrm{Int} \to \{\nu{:}\,\mathrm{Int} \mid \nu = y + 1\}]$$

Bidirectional type-checking

[Polikarpova et al.'16]

Type derivation

$$\dfrac{\dfrac{\dfrac{}{\Gamma, x: \mathrm{Nat} \vdash inc \uparrow y: \mathrm{Int} \to \{\nu: \mathrm{Int} \mid \nu = y + 1\}}\ \text{T-var} \qquad \dfrac{}{\Gamma, x: \mathrm{Nat} \vdash x \uparrow \mathrm{Nat}}\ \text{T-var}}{\Gamma, x: \mathrm{Nat} \vdash inc\ x \downarrow \mathrm{Nat}}\ \text{T-app}}{\Gamma \vdash \lambda x.\ inc\ x \downarrow \mathrm{Nat} \to \mathrm{Nat}}\ \text{T-abs}$$

Horn clauses

$$\nu \geq 0 \Rightarrow true$$

$$x \geq 0 \wedge \nu = x + 1 \Rightarrow \nu \geq 0$$

Subtyping constraints

$$\mathrm{Nat} <: \mathrm{Int}$$

$$x: \mathrm{Nat} \vdash \{\nu: \mathrm{Int} \mid \nu = x + 1\} <: \mathrm{Nat}$$

$$\Gamma = [\mathrm{inc}: y: \mathrm{Int} \to \{\nu: \mathrm{Int} \mid \nu = y + 1\}]$$

Recursion

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

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

$$\text{fix } f. \lambda n. \text{ if } n \leq 1 \text{ then } 1 \text{ else } n * (f\ (n - 1))$$

T-fix $$\frac{\Gamma, f : S \vdash e :: S}{\Gamma \vdash \text{fix } f. e :: S}$$

Example: factorial

$$\text{fix } f.\ \lambda n.\ \text{if } n \leq 1 \text{ then } 1 \text{ else } n * (f\ (n-1))$$

$$\text{T-app}\ \frac{\ldots \vdash f :: \text{Nat} \to \text{Nat} \qquad \ldots \vdash n - 1 :: \{\text{Int} \mid \nu = n - 1\}}{\ldots \vdash f\ (n-1) :: \text{Nat}}$$

$$\ldots$$

$$\text{T-if}\ \frac{\ldots \vdash n \leq 1 :: \text{Bool} \qquad \ldots \vdash 1 :: \text{Nat} \qquad \ldots, n > 1 \vdash n * \big(f\ (n-1)\big) :: \text{Nat}}{f : \text{Nat} \to \text{Nat}, n : \text{Nat} \vdash \text{if } n \leq 1 \text{ then } 1 \text{ else } n * \big(f\ (n-1)\big) :: \text{Nat}}$$

$$\text{T-abs}\ \frac{f : \text{Nat} \to \text{Nat}, n : \text{Nat} \vdash \text{if } n \leq 1 \text{ then } 1 \text{ else } n * \big(f\ (n-1)\big) :: \text{Nat}}{f : \text{Nat} \to \text{Nat} \vdash \lambda n \ldots :: \text{Nat} \to \text{Nat}}$$

$$\text{T-fix}\ \frac{f : \text{Nat} \to \text{Nat} \vdash \lambda n \ldots :: \text{Nat} \to \text{Nat}}{\emptyset \vdash \text{fix } f.\ \lambda n \ldots :: \text{Nat} \to \text{Nat}}$$

$$n > 1 \vdash \{\text{Int} \mid \nu = n - 1\} <: \text{Nat}$$