Exercise

Determine the output of the following program assuming static and dynamic scoping. Explain the difference, if there is any.

```
object MyClass {
  val x = 5
  def foo(z: Int): Int = { x + z }
  def bar(y: Int): Int = {
    val x = 1; val z = 2
    foo(y)
  }
  def main() {
    val x = 7
    println(foo(bar(3)))
  }
}
```

foo(3) ⇝ 4

4

static: 13

dynamic: 11

type judgement relation

$$\Gamma \vdash e : T$$

if the (free) variables of e have types given by gamma,
then e (correctly) type checks and has type T

$$\frac{\Gamma \vdash e_1 : T_1 \quad \dots \quad \Gamma \vdash e_n : T_n}{\Gamma \vdash e : T}$$

If e_1 type checks in gamma and has type T_1 and ...
and e_n type checks in gamma and has type T_n
then e type checks in gamma and has type T

type rule

Type Checker Implementation Sketch

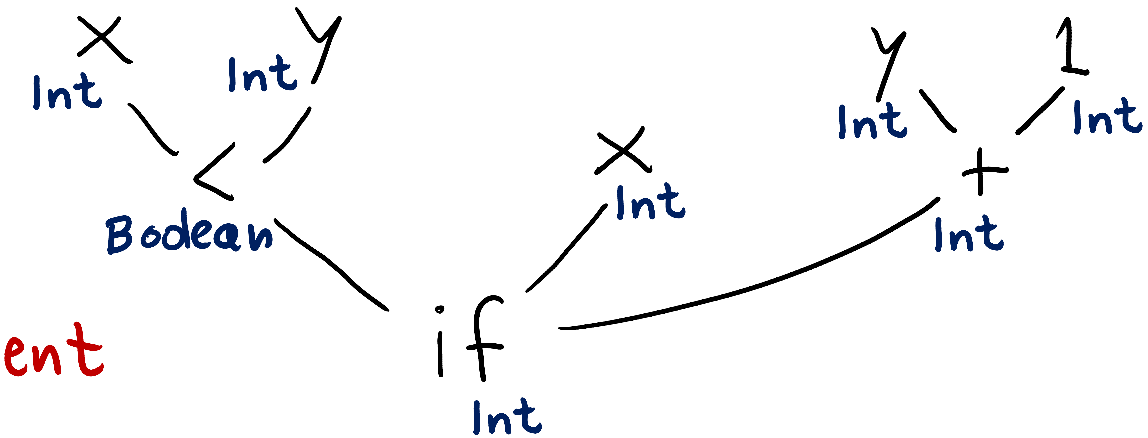
$\Gamma \vdash e : T$

```
def typeCheck(Γ : Map[ID, Type],  e : ExprTree) : TypeTree = {
 e match {
   case Var(id) => { Γ(id) match
     case Some(t) => t
     case None => error(UnknownIdentifier(id,id.pos))
   }
   case If(c,e1,e2) => {
     val tc = typeCheck(Γ,c)
     if (tc != BooleanType) error(IfExpectsBooleanCondition(e.pos))
     val t1 = typeCheck(Γ, e1); val t2 = typeCheck(Γ, e2)
     if (t1 != t2) error(IfBranchesShouldHaveSameType(e.pos))
     t1
   }
   ...
}}
```

Derivation Using Type Rules

x : Int
y : Int

type environment
Γ



Let $\Gamma = \{(x, \text{Int}), (y, \text{Int})\}$

$$\dfrac{\dfrac{\dfrac{(x,\text{Int})\in\Gamma}{\Gamma\vdash x:\text{Int}} \qquad \dfrac{(y,\text{Int})\in\Gamma}{\Gamma\vdash y:\text{Int}}}{\Gamma\vdash (x<y):\text{Boolean}} \qquad \dfrac{(x,\text{Int})\in\Gamma}{\Gamma\vdash x:\text{Int}} \qquad \dfrac{\dfrac{(y,\text{Int})\in\Gamma}{\Gamma\vdash y:\text{Int}} \qquad \dfrac{}{\Gamma\vdash 1:\text{Int}}}{\Gamma\vdash (y+1):\text{Int}}}{\Gamma\vdash (\text{if}(x<y)\ x\ \text{else}\ y+1):\text{Int}}$$

Type Rule for Function Application

We can treat operators as variables that have function type

$$+ : Int \times Int \to Int$$
$$< : Int \times Int \to Boolean$$
$$\&\& : Boolean \times Boolean \to Boolean$$

We can replace many previous rules with application rule:

$$\frac{\Gamma \vdash e_1 : T_1 \quad \dots \quad \Gamma \vdash e_n : T_n \quad \Gamma \vdash f : ((T_1 \times \cdots \times T_n) \to T)}{\Gamma \vdash f(e_1, \dots, e_n) : T}$$

$$\frac{\Gamma \vdash b_1 : Boolean \quad \Gamma \vdash b_2 : Boolean \quad \Gamma \vdash \&\& : Boolean \times Boolean \to Boolean}{\Gamma \vdash (b_1 \ \&\& \ b_2) : Boolean}$$

Computing the Environment of a Class

```
object World {
  var data : Int
  var name : String
  def m(x : Int, y : Int) : Boolean { ... }
  def n(x : Int) : Int {
    if (x > 0) p(x – 1) else 3
  }
  def p(r : Int) : Int = {
     var k = r + 2
     m(k, n(k))
  }
}
```

$$\Gamma_0 = \{ (data, Int), (name, String), (m, Int \times Int \to Boolean), (n, Int \to Int), (p, Int \to Int) \}$$

Type check each function m,n,p in this global environment

Extending the Environment

$\Gamma_0 = \{$

```
class World {
  var data : Int
  var name : String
  def m(x : Int, y : Int) : Boolean { ... }
  def n(x : Int) : Int {
    if (x > 0) p(x – 1) else 3
  }
  def p(r : Int) : Int = {
     var k:Int = r + 2
     m(k, n(k))
  }
}
```

$(data, Int),$
$(name, String),$
$(m, Int \times Int \to Boolean),$
$(n, Int \to Int),$
$(p, Int \to Int)\ \}$

$\leftarrow \Gamma_0$

$\leftarrow \Gamma_1 = \Gamma_0 \oplus \{(r, Int)\}$

$\leftarrow \Gamma_2 = \Gamma_1 \oplus \{(k, Int)\} = \Gamma_0 \cup \{(r, Int), (k, Int)\}$

Type Checking Expression in a Body

$\Gamma_0 = \{$
$(data, Int),$
$(name, String),$
$(m, Int \times Int \rightarrow Boolean),$
$(n, Int \rightarrow Int),$
$(p, Int \rightarrow Int) \}$

```
class World {
  var data : Int
  var name : String
  def m(x : Int, y : Int) : Boolean { ... }
  def n(x : Int) : Int {
    if (x > 0) p(x – 1) else 3
  }
  def p(r : Int) : Int = {
    var k:Int = r + 2
    m(k, n(k))
  }
}
```

$\leftarrow \Gamma_0$

$\leftarrow \Gamma_1 = \Gamma_0 \oplus \{(r, Int)\}$

$\leftarrow \Gamma_2 = \Gamma_1 \oplus \{(k, Int)\}$

$$\dfrac{\Gamma_2 \vdash k : Int \qquad \dfrac{\Gamma_2 \vdash n : Int \rightarrow Int \qquad \Gamma_2 \vdash k : Int}{\Gamma_2 \vdash n(k) : Int} \qquad \Gamma_2 \vdash m : Int \times Int \rightarrow Boolean}{\Gamma_2 \vdash m(k, n(k)) : Boolean}$$

Remember Function Updates

$$\{(x,T_1),(y,T_2)\} \oplus \{(x,T_3)\} = \{(x,T_3),(y,T_2)\}$$

Type Rule for Method Bodies

$$\frac{\Gamma \oplus \{(x_1,T_1),\ldots,(x_n,T_n)\} \vdash e : T}{\Gamma \vdash (\text{def } m(x_1{:}T_1,\ldots,x_n{:}T_n) : T = e) : \text{OK}}$$

Type Rule for Assignments

$$\frac{(x,T) \in \Gamma \qquad \Gamma \vdash e : T}{\Gamma \vdash (x = e) : \text{void}}$$

Unit

Type Rules for Block: { var x_1:T_1 ... var x_n:T_n; s_1; ... s_m; e }

$$\frac{\Gamma \oplus \{(x_1,T_1),\ldots,(x_n,T_n)\} \qquad \begin{array}{l} \vdash s_1 : \text{void} \\ \ldots \\ \vdash s_n : \text{void} \\ \vdash e : T \end{array}}{\Gamma \vdash \{\text{var } x_1 : T_1; \ldots; \text{var } x_n : T_n; s_1; \ldots; s_n; e\} : T}$$

Blocks with Declarations in the Middle

$$\frac{\Gamma \vdash e : T}{\Gamma \vdash \{e\} : T}$$ just expression

$$\frac{}{\Gamma \vdash \{\} : void}$$ empty

$$\frac{\Gamma \oplus \{(x, T_1)\} \vdash \{t_2; \dots; t_n\} : T}{\Gamma \vdash \{var\ x : T_1; t_2; \dots; t_n\} : T}$$ declaration is first

$$\frac{\Gamma \vdash s_1 : void \qquad \Gamma \vdash \{t_2; \dots; t_n\} : T}{\Gamma \vdash \{s_1; t_2; \dots; t_n\} : T}$$ statement is first

Rule for While Statement

$$\frac{\Gamma \vdash b : \text{Boolean} \qquad \Gamma \vdash s : \text{void}}{\Gamma \vdash (\text{while } (b)\ s) : \text{void}}$$

Rule for Method Call

```
class T0 {
  ...
  def m(x1:T1, ..., xn:Tn):T = {
     ...
  }
  ...
}
```

$$\Gamma_{T_0} \vdash m : T_0 \times T_1 \times \ldots \times T_n \to T$$

$$\frac{\Gamma \vdash x : T_0 \qquad \Gamma \vdash (T_0.m) : T_0 \times T_1 \times \ldots \times T_n \to T \qquad \overset{\forall i \in \{1,2,\ldots,n\}}{\Gamma \vdash e_i : T_i}}{\Gamma \vdash x.m(e_1,\ldots,e_n) : T}$$

$$m(x, e_1, \ldots, e_n)$$

Example to Type Check

```
object World {
  var z : Boolean
  var u : Int
  def f(y : Boolean) : Int {
    z = y
    if (u > 0) {
      u = u - 1
      var z : Int
      z = f(!y) + 3
      z+z
    } else { 0 }
  }
}
```

rest

s_1

e

Γ_0 = {
(z, Boolean),
(u, Int),
(f, Boolean $\rightarrow$ Int) }

$\Gamma_1 = \Gamma_0 \oplus \{(y, \text{Boolean})\}$

$$\frac{\Gamma_1 \vdash z\!:\ \text{Boolean} \qquad \Gamma_1 \vdash y\!:\ \text{Boolean}}{\Gamma_1 \vdash (z{=}y)\!:\ \text{void}}$$

Exercise:

$$\frac{???}{\Gamma_1 \vdash \text{if } (u > 0)\ \{\ \text{body}\ \}\ \text{else}\ \{\ 0\ \}\ :\ \text{Int}}$$

Overloading of Operators

$+: T \times T \to T$

Int x Int → Int

$$\frac{\Gamma \vdash e_1\colon \text{Int} \qquad \Gamma \vdash e_2\colon \text{Int}}{\Gamma \vdash (e_1 + e_2)\colon \text{Int}}$$

Not a problem for type checking from leaves to root

String x String → String

$$\frac{\Gamma \vdash e_1\colon \text{String} \qquad \Gamma \vdash e_2\colon \text{String}}{\Gamma \vdash (e_1 + e_2)\colon \text{String}}$$

Arrays

Using array as an expression, on the right-hand side

$$\frac{\Gamma \vdash a:\ \mathrm{Array}(T) \qquad \Gamma \vdash i:\ \mathrm{Int}}{\Gamma \vdash \underline{a[i]:T}}$$

Assigning to an array

$$\frac{\Gamma \vdash a:\ \mathrm{Array}(T) \qquad \Gamma \vdash i:\ \mathrm{Int} \qquad \Gamma \vdash e:\ T}{\Gamma \vdash (a[i] = e):\ \mathrm{void}}$$

Example with Arrays

```
def next(a : Array[Int], k : Int) : Int = {
    a[k] = a[a[k]]
}
```

Γ

Given Γ = {(a, Array(Int)), (k, Int)}, check Γ |- a[k] = a[a[k]]: Int

$$\dfrac{\dfrac{\Gamma \vdash a:\ \mathrm{Array(Int)} \qquad \dfrac{\Gamma \vdash a:\ \mathrm{Array(Int)} \qquad \Gamma \vdash k:\ \mathrm{Int}}{\Gamma \vdash a[k]:\ \mathrm{Int}}}{\Gamma \vdash a[a[k]]:\mathrm{Int}} \qquad \Gamma \vdash a:\ \mathrm{Array(Int)} \qquad \Gamma \vdash k:\mathrm{Int}}{\Gamma \vdash (a[k] = a[a[k]]) : \cancel{\mathrm{Int}} : \mathrm{void}}$$

Type Rules (1)

$$\frac{(\mathrm{x}:\ \mathrm{T}) \in \Gamma}{\Gamma \vdash \mathrm{x}:\ \mathrm{T}}$$ variable

$$\frac{}{\mathrm{IntConst(k)}:\ \mathrm{Int}}$$ constant

$$\frac{\Gamma \vdash e_1 : T_1 \ \ \ldots \ \ \Gamma \vdash e_n : T_n \qquad \Gamma \vdash f : (T_1 \times \cdots \times T_n \to T)}{\Gamma \vdash f(e_1, \ldots, e_n) : T}$$ function application

$$\frac{\Gamma \vdash e_1:\ \mathrm{Int} \qquad \Gamma \vdash e_2:\ \mathrm{Int}}{\Gamma \vdash (e_1 + e_2):\ \mathrm{Int}}$$ plus

$$\frac{\Gamma \vdash e_1:\ \mathrm{String} \qquad \Gamma \vdash e_2:\ \mathrm{String}}{\Gamma \vdash (e_1 + e_2):\ \mathrm{String}}$$

$$\frac{\Gamma \vdash \mathrm{b}:\ \mathrm{Boolean} \qquad \Gamma \vdash e_1 :\ \mathrm{T} \qquad \Gamma \vdash e_2 :\ \mathrm{T}}{\Gamma \vdash (\mathrm{if(b)}\ e_1\ \mathrm{else}\ e_2) :\ \mathrm{T}}$$ if

$$\frac{\Gamma \vdash \mathrm{b}:\ \mathrm{Boolean} \qquad \Gamma \vdash \mathrm{s}:\ \mathrm{void}}{\Gamma \vdash (\mathrm{while(b)}\ \mathrm{s}):\ \mathrm{void}}$$

$$\frac{(\mathrm{x},\ \mathrm{T}) \in \Gamma \qquad \Gamma \vdash \mathrm{e}:\ \mathrm{T}}{\Gamma \vdash (\mathrm{x{=}e}):\ \mathrm{void}}$$

while

assignment

Type Rules (2)

$$\frac{\Gamma \vdash \text{e: T}}{\Gamma \vdash \{\text{e}\}\text{: T}} \qquad \frac{}{\Gamma \vdash \{\}\text{: void}}$$

$$\frac{\Gamma \oplus \{(x, T_1)\} \vdash \{t_2; \ \ldots \ ; t_n\}\text{: T}}{\Gamma \vdash \{\text{var } x : T_1; t_2; \ \ldots \ ; t_n\}\text{: T}}$$

$$\frac{\Gamma \vdash s_1\text{: void} \qquad \Gamma \vdash \{t_2; \ \ldots \ ; t_n\}\text{: T}}{\Gamma \vdash \{s_1; t_2; \ \ldots \ ; t_n\}\text{: T}}$$

block

$$\frac{\Gamma \vdash \text{a: Array(T)} \qquad \Gamma \vdash \text{i: Int}}{\Gamma \vdash \text{a[i]: T}}$$

array use

$$\frac{\Gamma \vdash \text{a: Array(T)} \qquad \Gamma \vdash \text{i: Int} \qquad \Gamma \vdash \text{e: T}}{\Gamma \vdash (\text{a[i] = e})\text{: void}}$$

array
assignment

Type Rules (3)

Γ^c - top-level environment of class C

```
class C {
  var x: Int;
  def m(p: Int): Boolean = {…}
}
```

var y: C

Γ^c = { (x, Int), (m, C x Int $\rightarrow$ Boolean)}

$$\frac{\Gamma \vdash e : C \quad \Gamma^C \vdash m : T \times T_1 \times \ \ldots \ \times T_n \rightarrow T_{n+1} \quad \Gamma \vdash e_i : T_i \quad 1 \leq i \leq n}{\Gamma \vdash e.m(e_1, \ \ldots \ , e_n) : T_{n+1}}$$

C

method invocation

$$\frac{\Gamma \vdash \mathrm{e{:}}\ \mathrm{C} \qquad \Gamma^C \vdash \mathrm{f{:}}\ \mathrm{T}}{\Gamma \vdash \mathrm{e.f{:}}\ \mathrm{T}}$$

field use

$$\frac{\Gamma \vdash \mathrm{e{:}}\ \mathrm{C} \qquad \Gamma^C \vdash \mathrm{f{:}}\ \mathrm{T} \qquad \Gamma \vdash \mathrm{x{:}}\ \mathrm{T}}{\Gamma \vdash (\mathrm{e.f} = \mathrm{x}){:}\ \mathrm{void}}$$

field assignment

Does this program type check?

```
class Rectangle {
  var width: Int
  var height: Int
  var xPos: Int
  var yPos: Int
  def area(): Int = {
    if (width > 0 && height > 0)
     width * height
   else 0
  }
  def resize(maxSize: Int) {
    while (area > maxSize) {
      width = width / 2
      height = height / 2
    }
  }
}
```

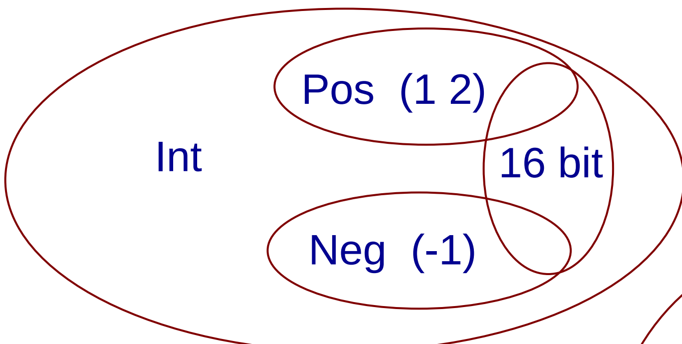
Meaning of Types

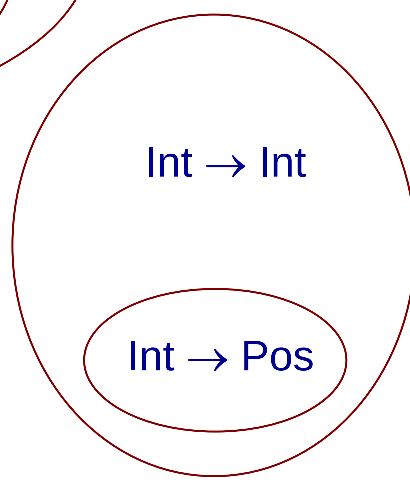
- Types can be viewed as named entities
  - explicitly declared classes, traits
  - their meaning is given by methods they have
  - constructs such as inheritance establish relationships between classes
- Types can be viewed as sets of values
  - Int = { ..., -2, -1, 0, 1, 2, ... }
  - Boolean = { false, true }
  - Int → Int = { f : Int -> Int | f is computable }

Types as Sets

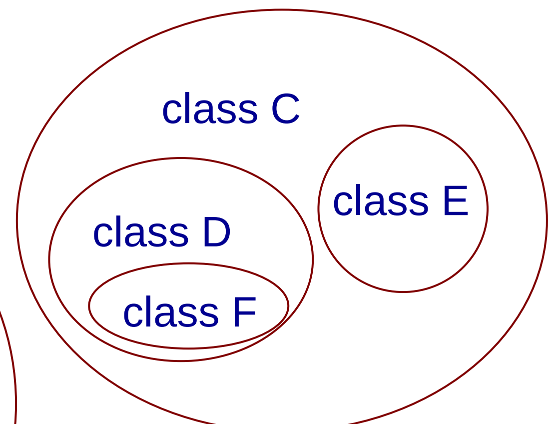
- Sets so far were disjoint

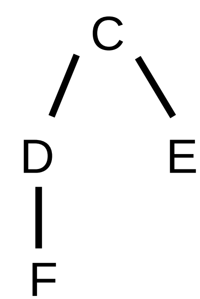
- Sets can overlap





C represents not only declared C,
but all possible extensions as well





F extends D,
D extends C

SUBTYPING

Subtyping

- Subtyping corresponds to subset
- Systems with subtyping have non-disjoint sets
- $T_1 <: T_2$ means T_1 is a subtype of T_2
  - corresponds to $T_1 \subseteq T_2$ in sets of values
- Rule for subtyping: analogous to set reasoning

$$\frac{\Gamma \vdash e : T_1 \qquad T_1 <: T_2}{\Gamma \vdash e : T_2} \qquad\qquad \frac{e \in T_1 \qquad T_1 \subseteq T_2}{e \in T_2}$$

Types for Positive and Negative Ints

Int = { ... , -2, -1, 0, 1, 2, ... }

Pos = { 1, 2, ... } (not including zero)

Neg = { ..., -2, -1 } (not including zero)

types

Pos <: Int
Neg <: Int

$$\frac{\Gamma \vdash \mathrm{x}\text{: } \mathrm{Pos} \qquad \Gamma \vdash \mathrm{y}\text{: } \mathrm{Pos}}{\Gamma \vdash \mathrm{x} + \mathrm{y}\text{: } \mathrm{Pos}}$$

$$\frac{\Gamma \vdash \mathrm{x}\text{: } \mathrm{Pos} \qquad \Gamma \vdash \mathrm{y}\text{: } \mathrm{Neg}}{\Gamma \vdash \mathrm{x} * \mathrm{y}\text{: } \mathrm{Neg}}$$

$$\frac{\Gamma \vdash \mathrm{x}\text{: } \mathrm{Pos} \qquad \Gamma \vdash \mathrm{y}\text{: } \mathrm{Pos}}{\Gamma \vdash \mathrm{x} / \mathrm{y}\text{: } \mathrm{Pos}}$$

sets

Pos $\subseteq$ Int
Neg $\subseteq$ Int

$$\frac{\mathrm{x} \in \mathrm{Pos} \qquad \mathrm{y} \in \mathrm{Pos}}{\mathrm{x} + \mathrm{y} \in \mathrm{Pos}}$$

$$\frac{\mathrm{x} \in \mathrm{Pos} \qquad \mathrm{y} \in \mathrm{Neg}}{\mathrm{x} * \mathrm{y} \in \mathrm{Neg}}$$

(y not zero)

$$\frac{\mathrm{x} \in \mathrm{Pos} \qquad \mathrm{y} \in \mathrm{Pos}}{\mathrm{x} / \mathrm{y} \in \mathrm{Pos}}$$

(x/y well defined)