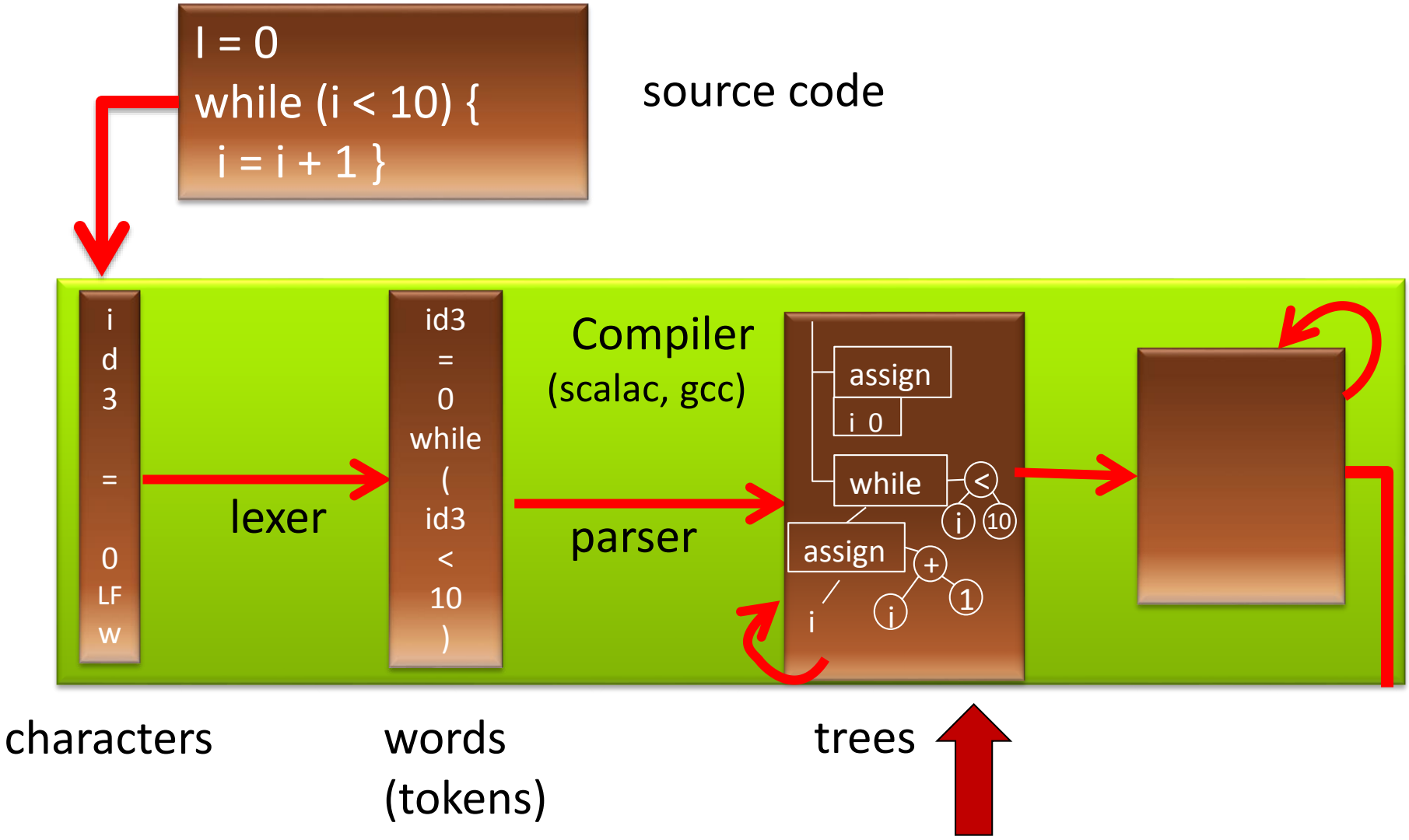

I = 0
while (i < 10) {
i = i + 1 }
source code
i
d
3
=
0
LF
w
lexer
id3
=
0
while
(
id3
<
10
)
Compiler
(scalac, gcc)
parser
assign
i 0
while
<
i
10
assign
+
i
1
i
characters
words
(tokens)
trees
Type Checking

Evaluating an Expression

scala prompt:

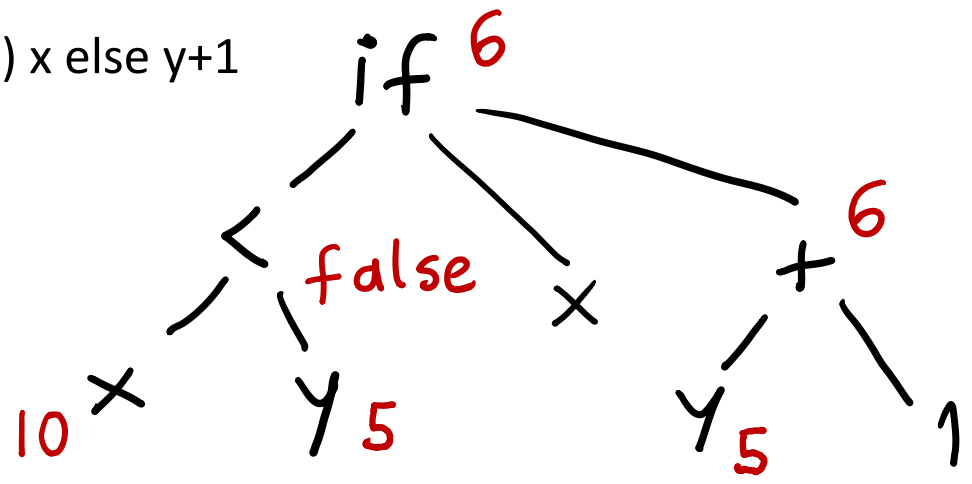
```
>def min1(x : Int, y : Int) : Int = { if (x < y) x else y+1 }
min1: (x: Int,y: Int)Int
>min1(10,5)
res1: Int = 6
```

How can we think about this evaluation?

x → 10

y → 5

if (x < y) x else y+1



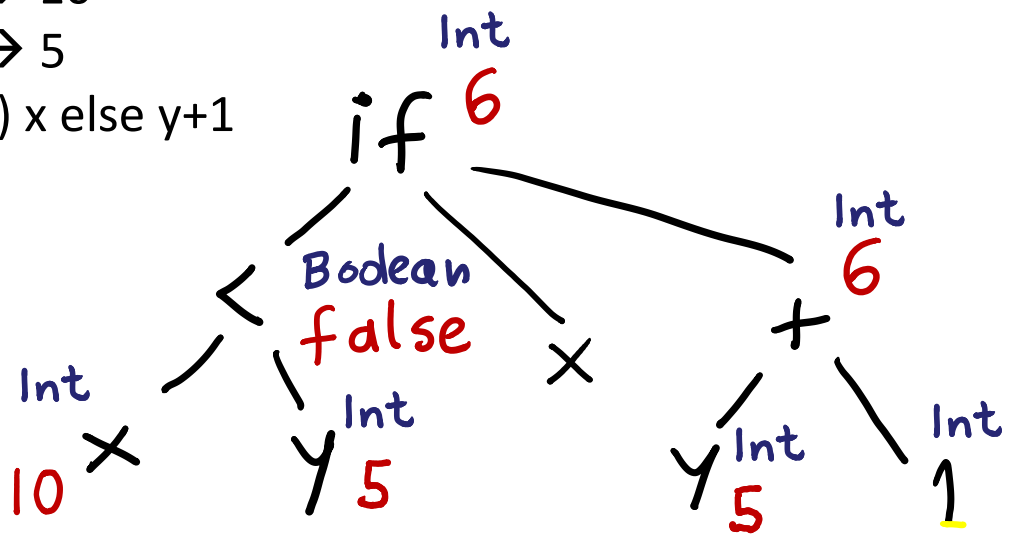
Computing types using the evaluation tree

scala prompt:

```
>def min1(x : Int, y : Int) : Int = { if (x < y) x else y+1 }
min1: (x: Int,y: Int)Int
>min1(10,5)
res1: Int = 6
```

How can we think about this evaluation?

x : Int → 10
y : Int → 5
if (x < y) x else y+1



We can compute types without values

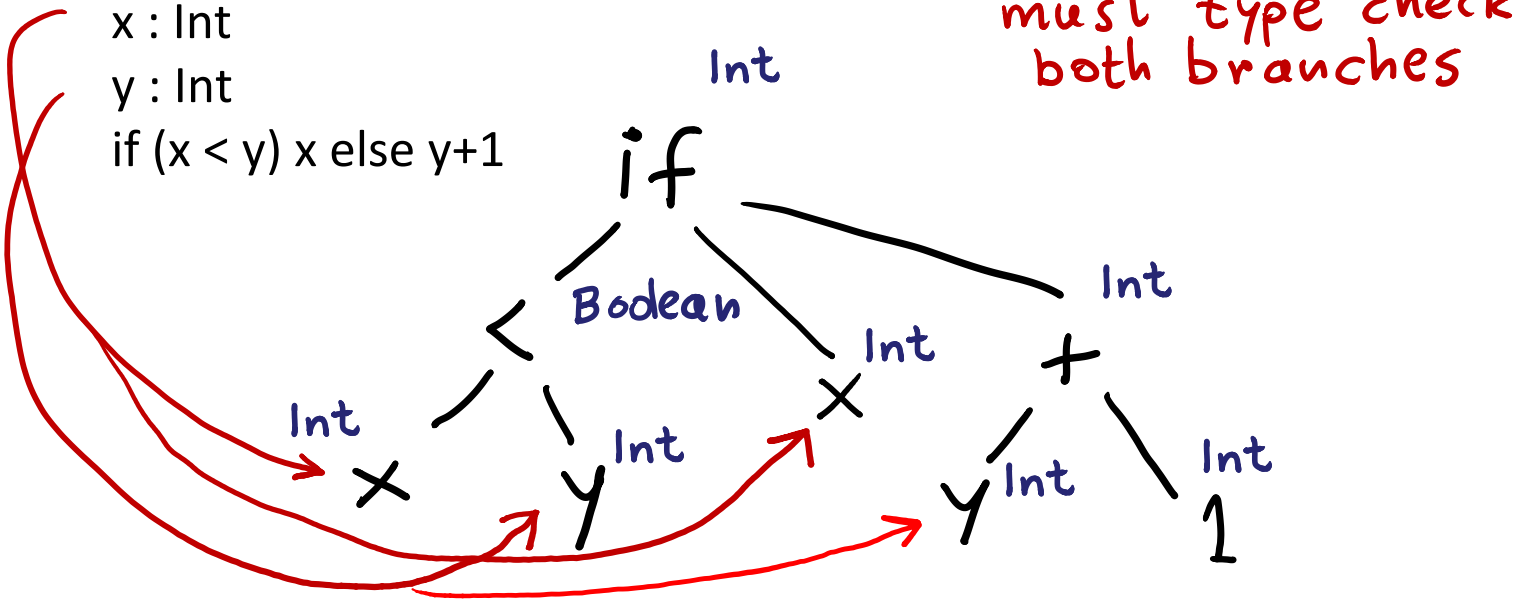
scala prompt:

```
>def min1(x : Int, y : Int) : Int = { if (x < y) x else y+1 }
min1: (x: Int,y: Int)Int
>min1(10,5)
res1: Int = 6
```

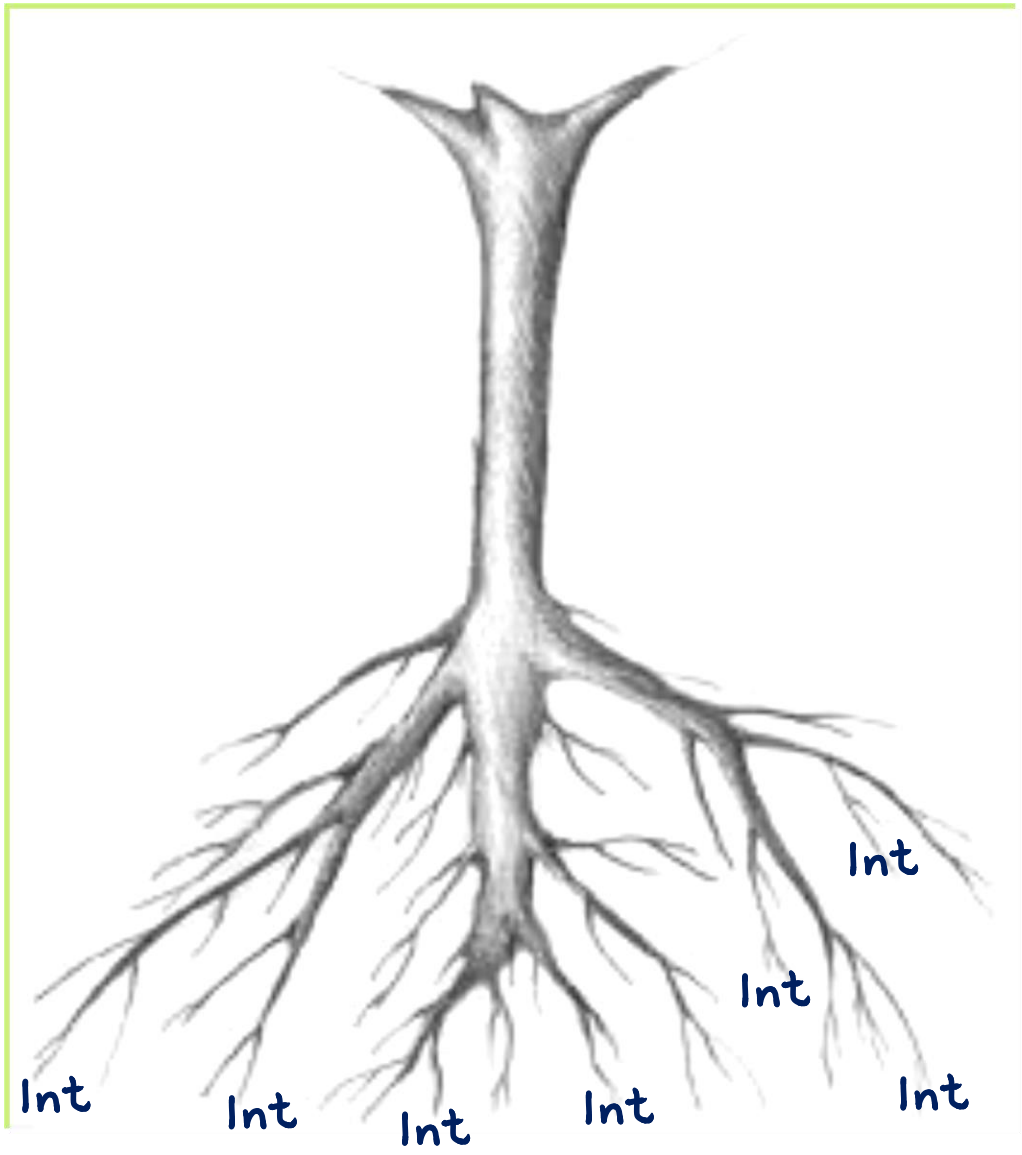
How can we think about this evaluation?

x : Int
y : Int
if (x < y) x else y+1

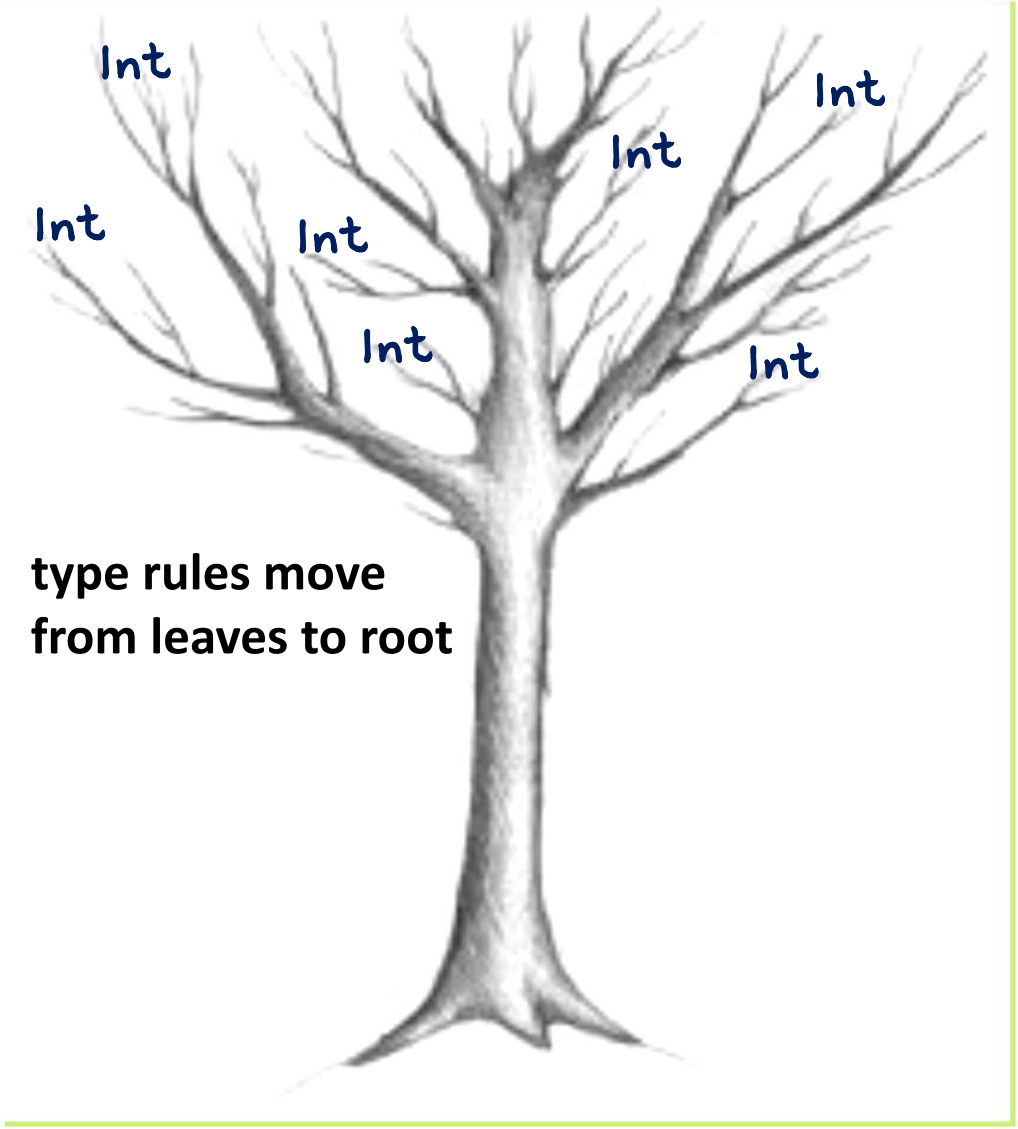
must type check both branches



We do not like trees upside-down



Leaves are Up



Type Judgements and Type Rules

- e type checks to T under Γ (type environment)

$$\Gamma \vdash e : T$$

  - Types of constants are predefined
  - Types of variables are specified in the source code
- If e is composed of sub-expressions

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

type check from leaves

Type Judgements and Type Rules

$$\Gamma \vdash e : T$$

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

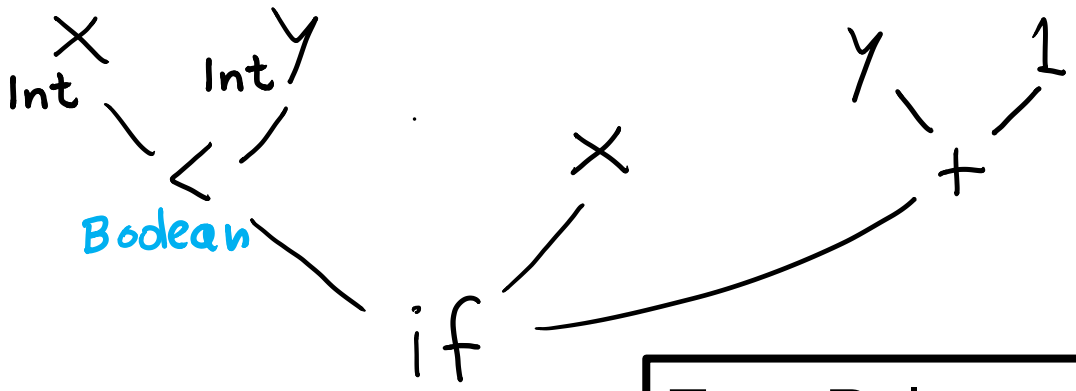
$$\frac{\Gamma \vdash e_1 : T_1 \quad \cdots \quad \Gamma \vdash e_n : T_n}{\Gamma \vdash e : T}$$

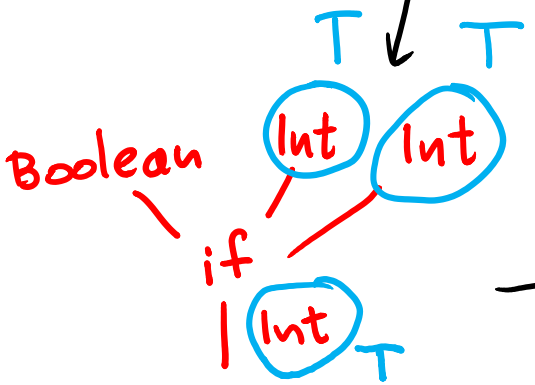
If e_1 type checks in gamma and has type T_1 and ...
and e_n type checks in gamma and has type T_n
then e type checks in gamma and has type T

type rule

Type Rules as Local Tree Constraints

x : Int
y : Int





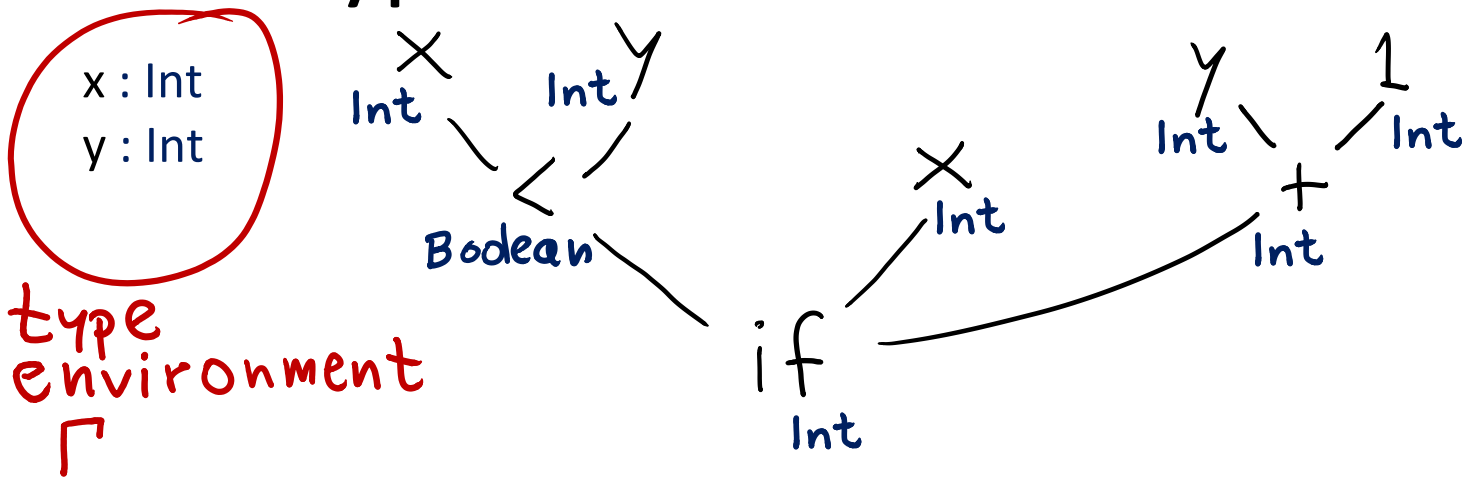
Type Rules

$$\frac{e_1 : \text{Int} \qquad e_2 : \text{Int}}{e_1 < e_2 : \text{Boolean}}$$

for every type T, if
b has type Boolean, and ...
then

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

Type Rules with Environment



Type Rules

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

$$\frac{}{\Gamma \vdash \text{IntConst}(k) : \text{Int}}$$

$$\frac{\Gamma \vdash e_1 : \text{Int} \quad \Gamma \vdash e_2 : \text{Int}}{\Gamma \vdash (e_1 < e_2) : \text{Boolean}}$$

...(then) in the (same) environment Γ the expression $e_1 < e_2$ has type Bool.

$$\frac{\Gamma \vdash e_1 : \text{Int} \quad \Gamma \vdash e_2 : \text{Int}}{\Gamma \vdash (e_1 + e_2) : \text{Int}}$$

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

Type Checker Implementation Sketch

```
def typeCheck(Γ : Map[ID, Type],  e : ExprTree) : TypeTree = {
 e match {
   case Var(id) => {  ??    }
   case If(c,e1,e2) => { ?? }
   ...
}}


   case Var(id) => { Γ(id) match
     case Some(t) => t
     case None => error(UnknownIdentifier(id,id.pos))
   }
```

Type Checker Implementation Sketch

- **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)

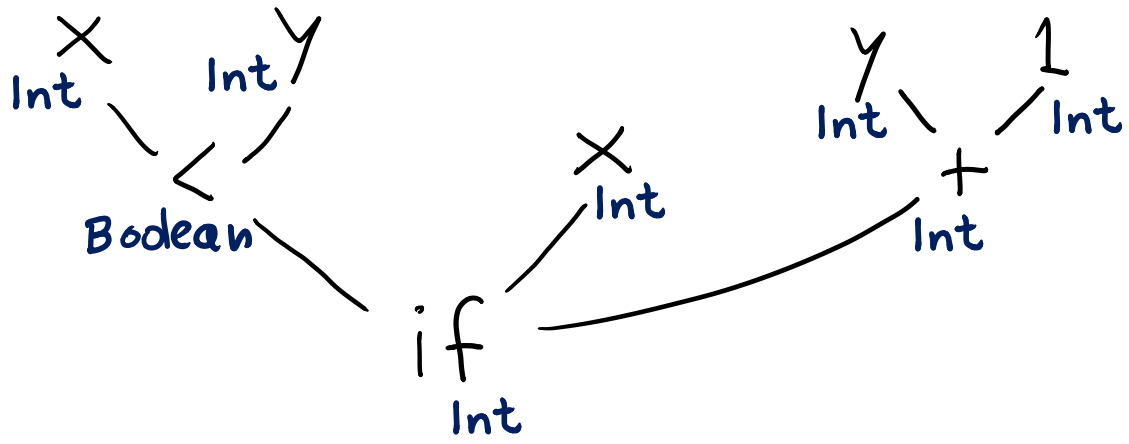
  if (t1 != t2) error(IfBranchesShouldHaveSameType(e.pos))

  t1

  }

Derivation Using Type Rules

x : Int
y : Int



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

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

$$\Gamma \vdash (\text{if } (x<y) \text{ } x \text{ else } y+1) : Int$$

Type Rule for Function Application

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

Type Rule for Function Application [Cont.]

We can treat operators as variables that have function type

$+ : \text{Int} \times \text{Int} \rightarrow \text{Int}$

$< : \text{Int} \times \text{Int} \rightarrow \text{Boolean}$

$\&\& : \text{Boolean} \times \text{Boolean} \rightarrow \text{Boolean}$

We can replace many previous rules with application rule:

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

$$\frac{\Gamma \vdash e_1 : \text{Bool} \quad \Gamma \vdash e_2 : \text{Bool} \quad \Gamma \vdash \&\& : (\text{Bool} \times \text{Bool}) \rightarrow \text{Bool}}{\Gamma \vdash e_1 \;\&\&\; e_2 : \text{Bool}}$$

Computing the Environment of a Class

$\Gamma_0 = \{$

```
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))
  }
}
```

(data, Int),

(name, String),

(m, Int x Int → Boolean),

(n, Int → Int),

(p, Int → Int)

}

We can 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 x Int → Boolean),
(n, Int → Int),
(p, Int → Int) }

← Γ_0

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

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

Type Checking Expression in a Body

$\Gamma_0 = \{$

(data, Int),
(name, String),
(m, Int x Int → Boolean),
(n, Int → Int),
(p, Int → 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) : Boolean = {
    var k:Int = r + 2
    m(k, n(k))
  }
}
```

$\leftarrow \Gamma_0$

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

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

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

$$\frac{\Gamma_2 \vdash k: \text{Int} \quad \dfrac{\Gamma_2 \vdash n: \text{Int} \to \text{Int} \quad \Gamma_2 \vdash k: Int}{\Gamma_2 \vdash n(\text{k}): \text{Int}} \quad \Gamma_2 \vdash \text{m}: \text{Int} \times \text{Int} \to \text{Bool}}{\Gamma_2 \vdash m\big(k, n(k)\big): \text{Bool}}$$

Type Rule for Method Definitions

$\text{def } m(x_1 : T_1, \cdots, x_n : T_n) : T = e$

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

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 }

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

Blocks with Declarations in the Middle

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

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

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

$$\frac{\Gamma \vdash S_1 : \text{void} \qquad \Gamma \vdash \{t_2; \ldots; t_n\} : T}{\Gamma \vdash \{S_1; t_2; \ldots; t_n\} : T}$$ 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 a Method Call

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

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

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

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 }
  }
}
```

Γ_0 = {
(z, Boolean),
(u, Int),
(f, Boolean → 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:

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

Solution

$$\dfrac{\dfrac{(\texttt{u},\texttt{Int}) \in \Gamma}{\Gamma \vdash \texttt{u} : \texttt{Int}} \quad \vdash 0 : \texttt{Int}}{\Gamma \vdash \texttt{u > 0} : \texttt{Boolean}} \qquad \dfrac{\dfrac{(u,\texttt{Int}) \in \Gamma}{\Gamma \vdash \texttt{u} : \texttt{Int}} \quad \vdash 1 : \texttt{Int}}{\Gamma \vdash \texttt{u=u-1} : \texttt{void}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{(\texttt{y},\texttt{Boolean}) \in \Gamma'}{\Gamma' \vdash \texttt{y} : \texttt{Boolean}}}{\Gamma' \vdash \texttt{!y} : \texttt{Boolean}} \quad \dfrac{(\texttt{f},\texttt{Boolean} \to \texttt{Int}) \in \Gamma'}{\Gamma' \vdash \texttt{f} : \texttt{Boolean} \to \texttt{Int}}}{\Gamma' \vdash \texttt{f(!y)} : \texttt{Int}} \quad \dfrac{(\texttt{z},\texttt{Int}) \in \Gamma'}{\Gamma' \vdash \texttt{z} : \texttt{Int}} \quad \vdash 3 : \texttt{Int}}{\Gamma' \vdash \texttt{\{z=f(!y)+3\}} : \texttt{void}} \quad \dfrac{(\texttt{z},\texttt{Int}) \in \Gamma'}{\Gamma' \vdash \texttt{\{z+z\}} : \texttt{Int}}}{\Gamma' = \Gamma \oplus (\texttt{z},\texttt{Int}) \vdash \texttt{\{z=f(!y)+3;z+z\}} : \texttt{Int}}}{\Gamma \vdash \texttt{\{var z:Int;z=f(!y)+3;z+z\}} : \texttt{Int}}$$

$$\dfrac{\Gamma \vdash \texttt{u > 0} : \texttt{Boolean} \qquad \texttt{\{u=u-1;var z:Int;z=f(!y)+3;z+z\}} : \texttt{Int} \qquad \vdash 0 : \texttt{Int}}{\Gamma \vdash \texttt{if (u > 0) \{ u = u - 1; var z :Int; z = f(!y) + 3; z + z \} else \{ 0 \} :Int}}$$

Overloading of Operators

Int x Int $\rightarrow$ 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 $\rightarrow$ String

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