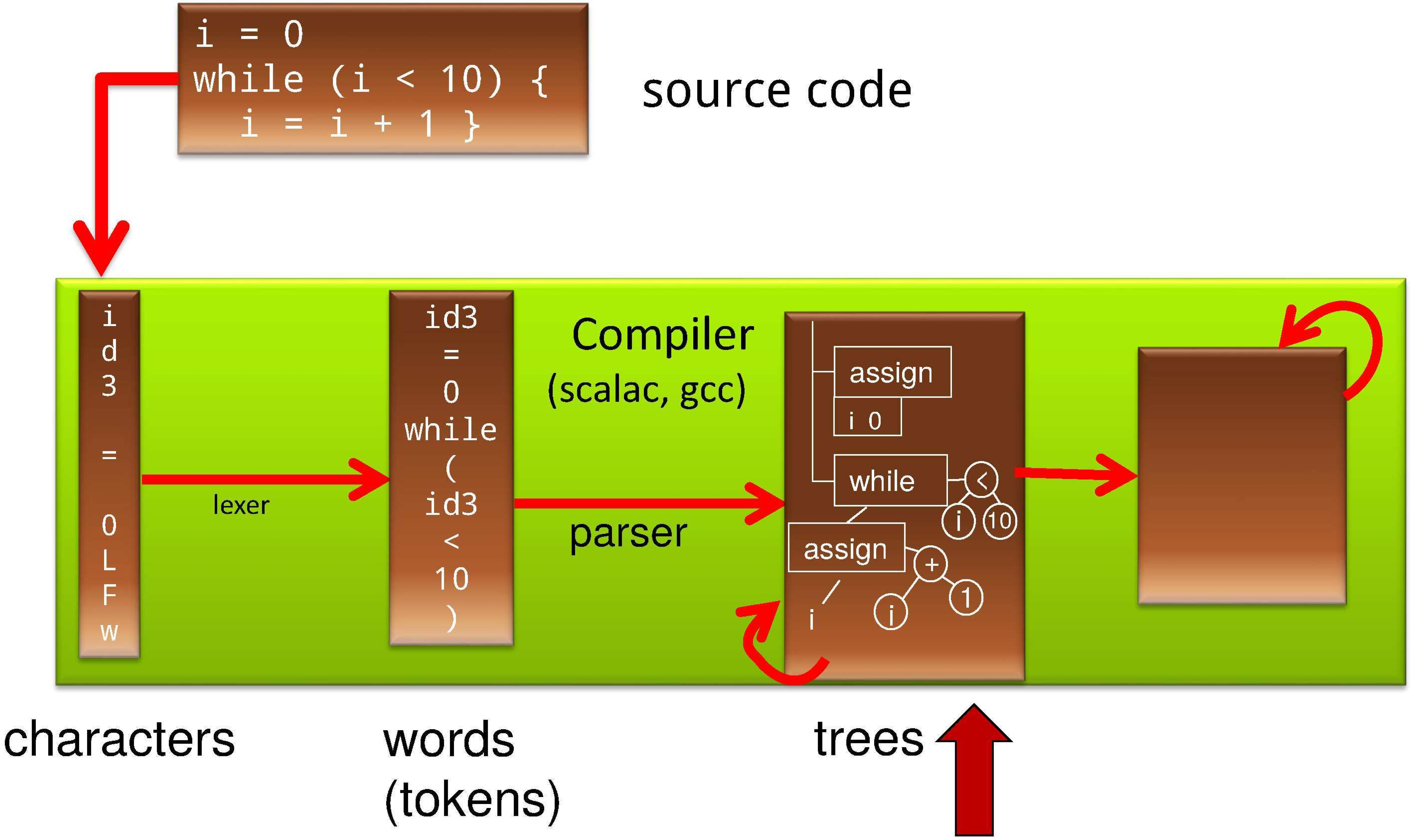


Type Checking

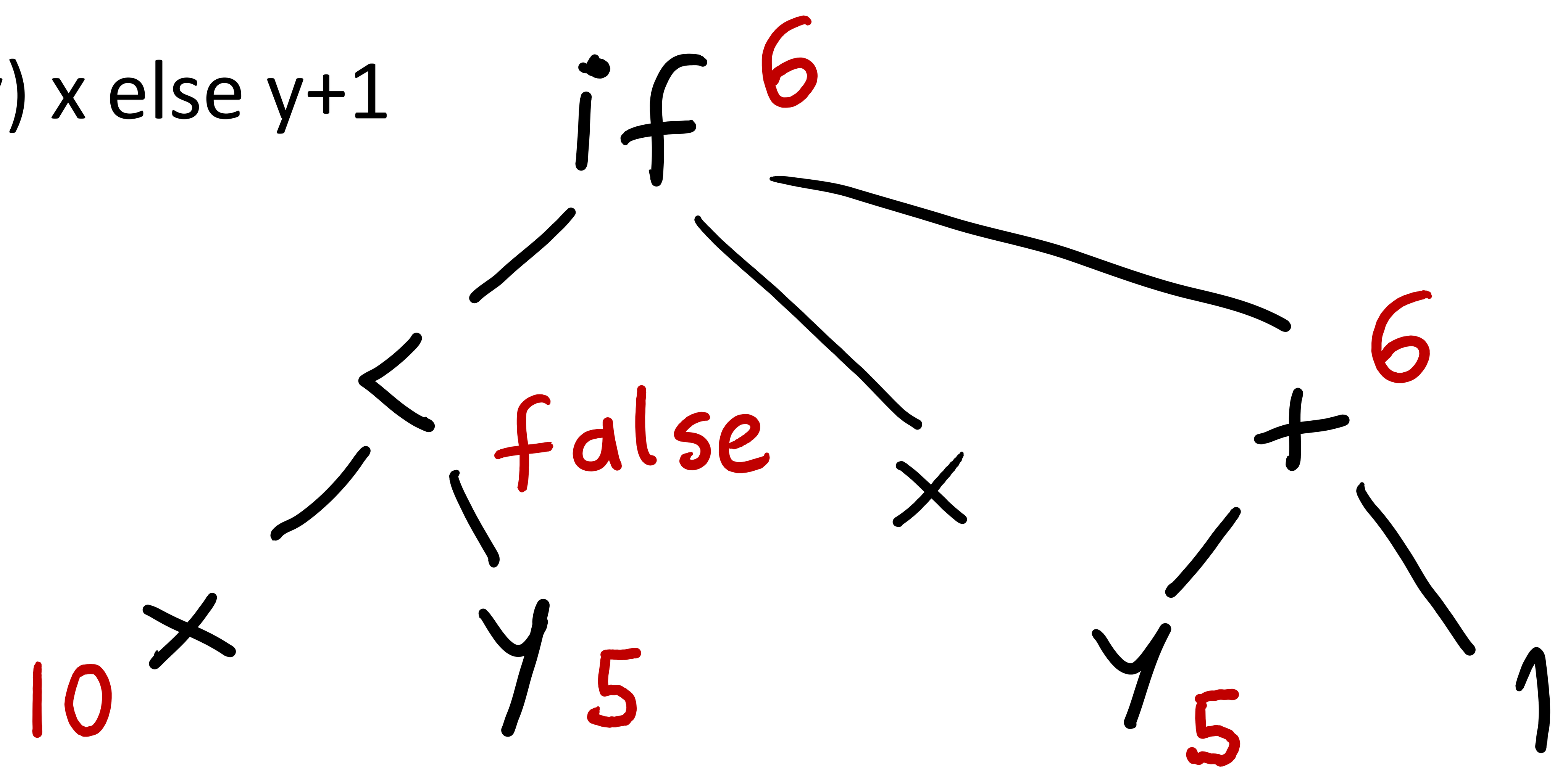
Evaluating an Expression

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 $\rightarrow$ 10
y $\rightarrow$ 5
if (x < y) x else y+1



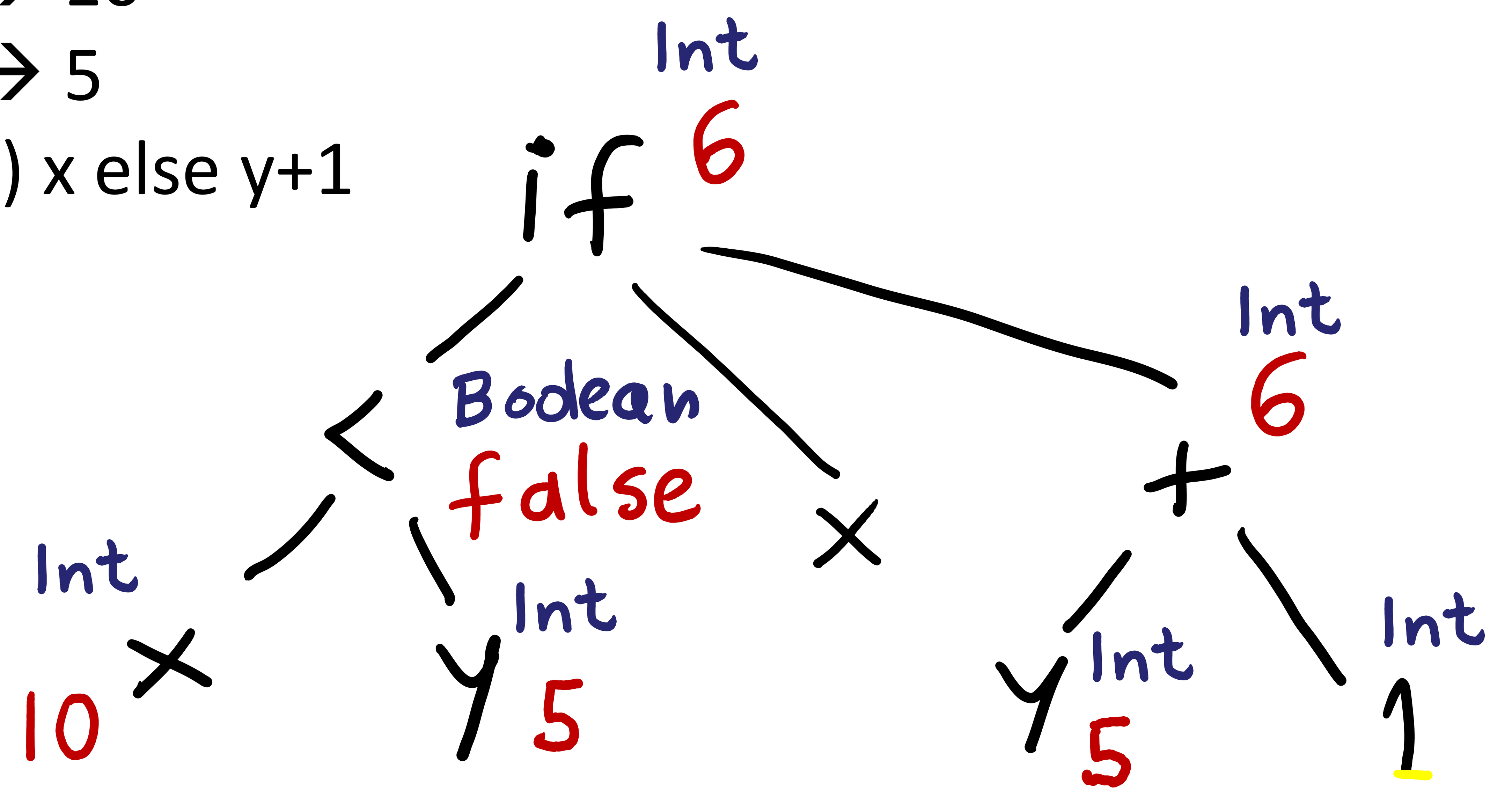
Computing types using the evaluation tree

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



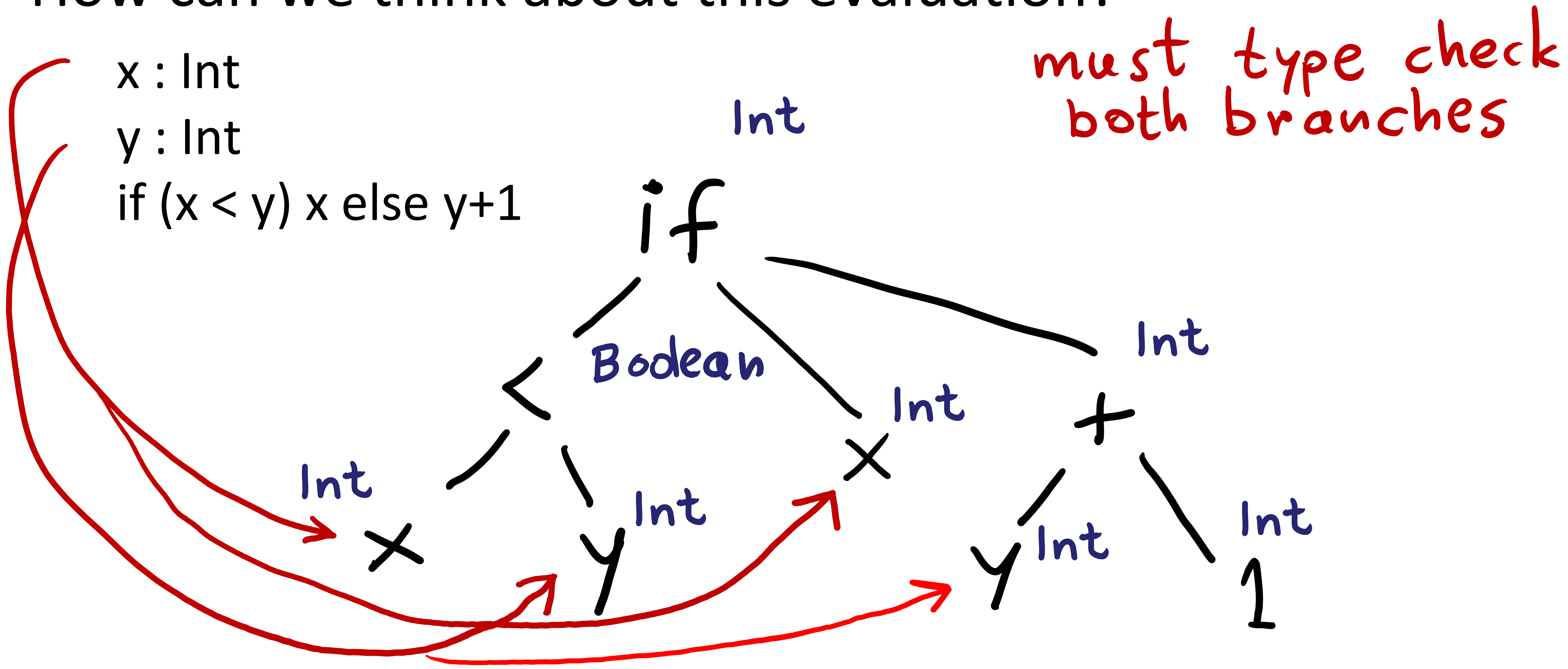
We can compute types without values

scala prompt:

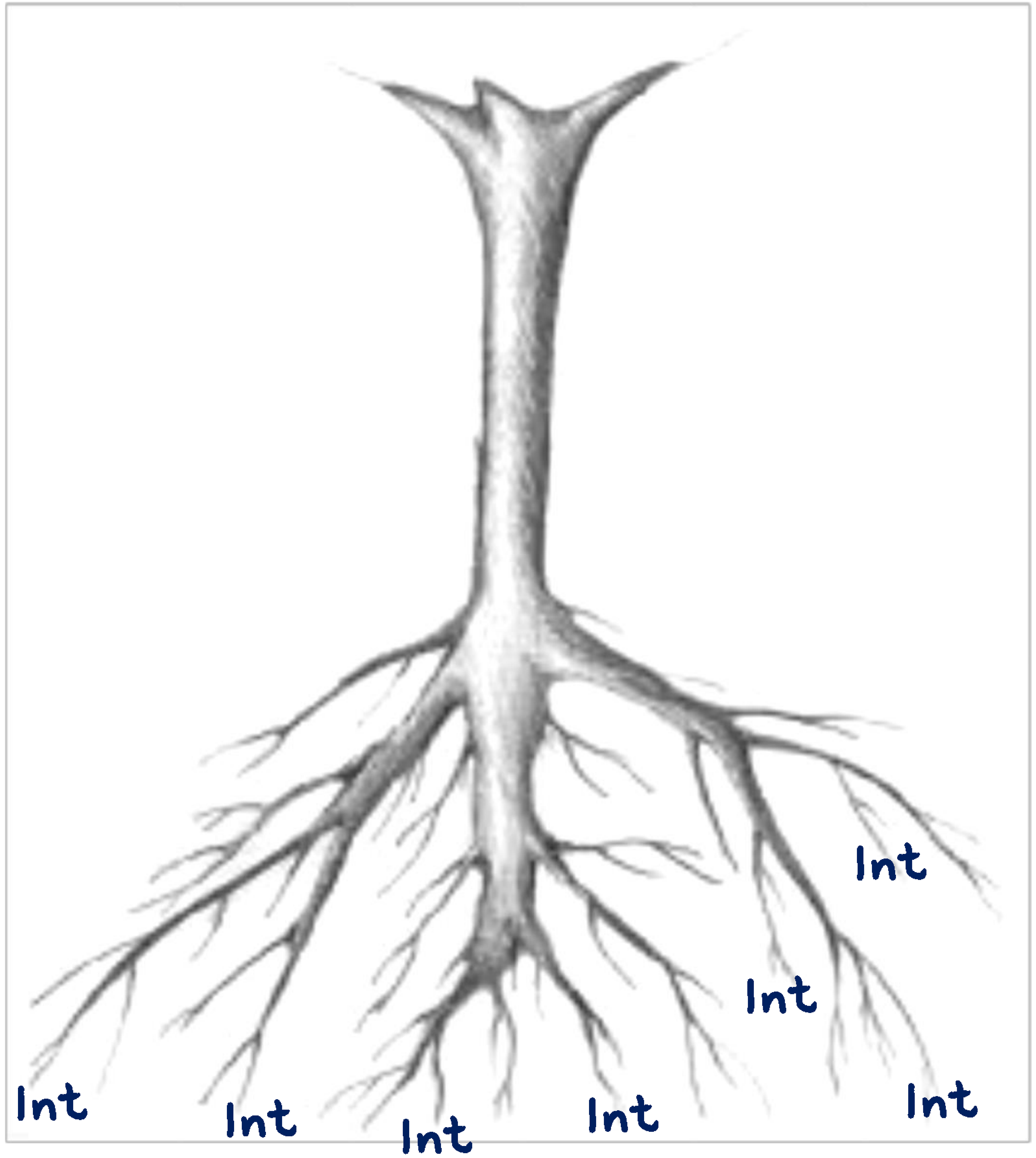
```
>def min1(x : Int, y : Int) : Int = { if (x < y) x else y+1 }
min1: (x: Int,y: Int)Int
>min1(10,5)
res1: Int = 6
```

How can we think about this evaluation?

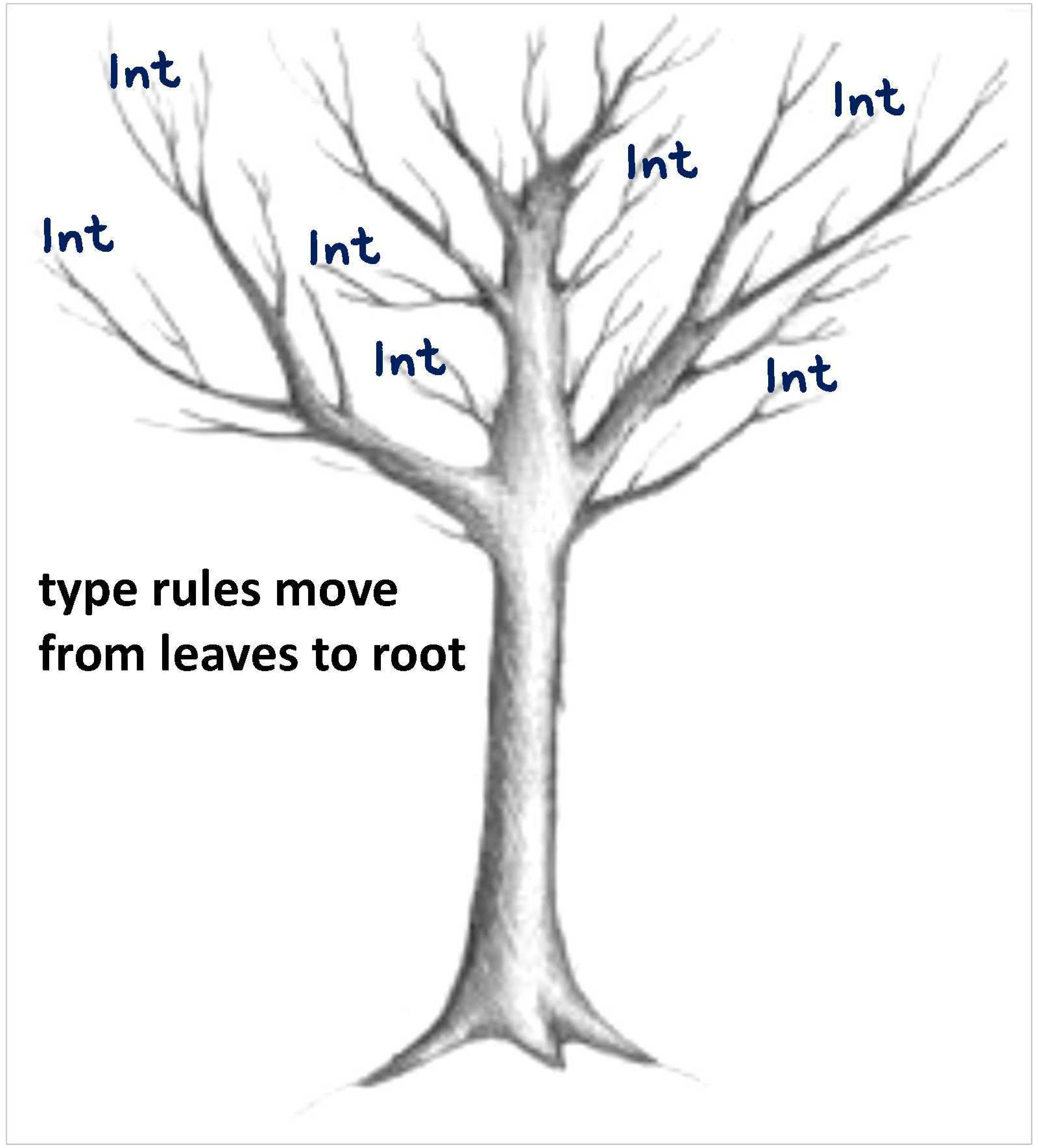
x : Int
y : Int
if (x < y) x else y+1



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

$$\Gamma \vdash e : T$$

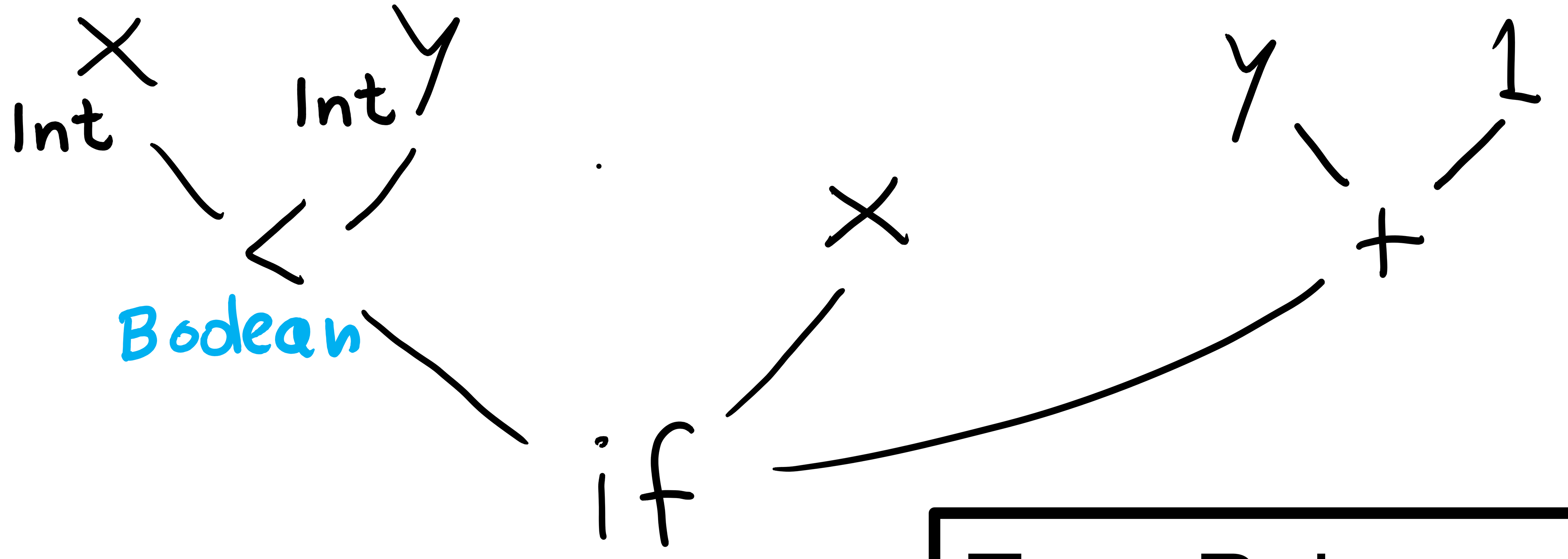
if the (free) variables of e have types given by the type environment gamma, then e (correctly) type checks and has type T

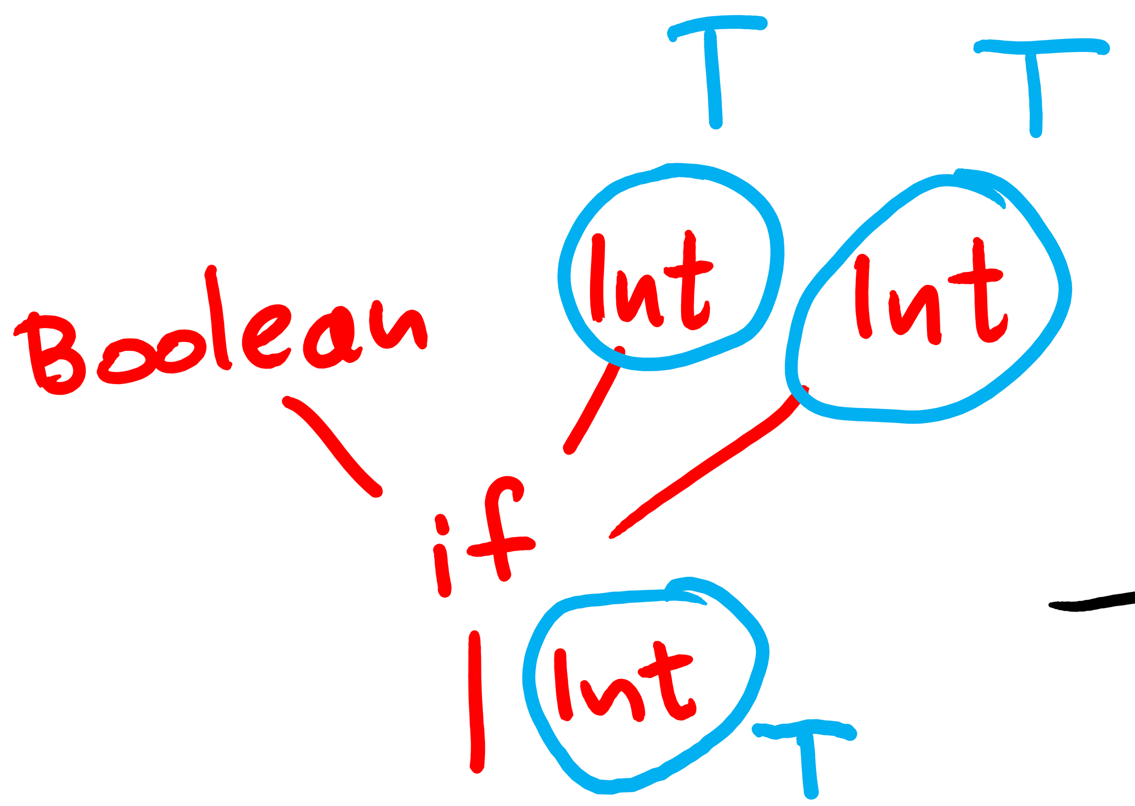
type rule
$$\frac{\Gamma \vdash e_1 : T_1 \quad \cdots \quad \Gamma \vdash e_n : T_n}{\Gamma \vdash e : T}$$

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 Rules as Local Tree Constraints

x : Int
y : Int





Type Rules

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

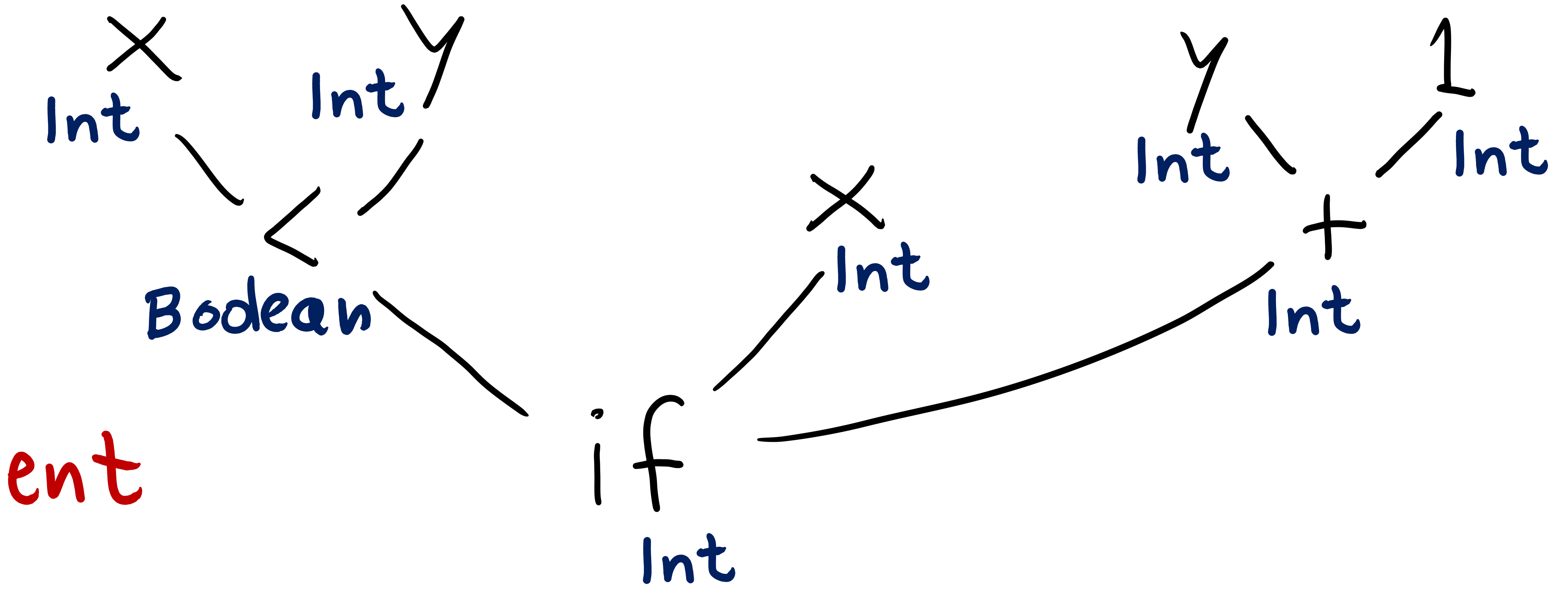
for every type T, if
b has type Boolean, and ...
then

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

Type Rules with Environment

x : Int
y : Int

type environment Γ



Type Rules

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

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

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

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

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

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

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))
   }
```

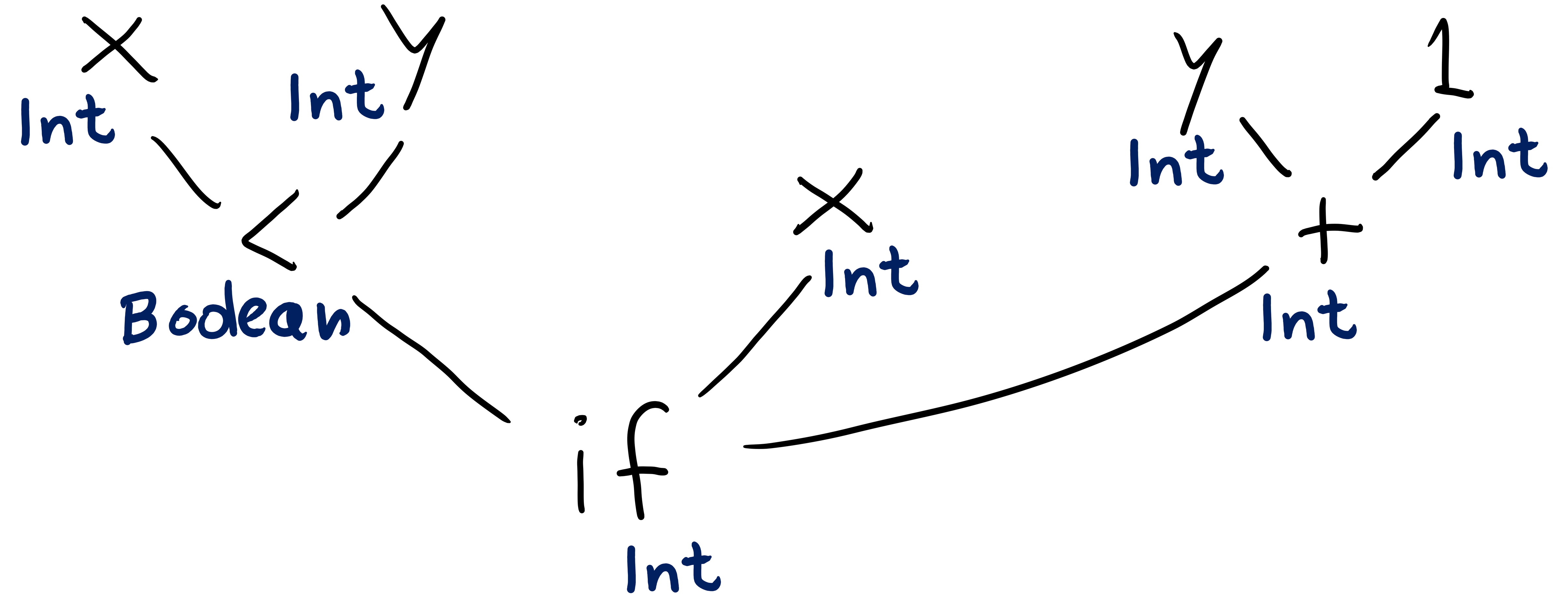
Type Checker Implementation Sketch

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

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

Type Rule for Function Application

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

Type Rule for Function Application [Cont.]

We can treat operators as variables that have function type

$+ : Int \times Int \to Int$

$< : Int \times Int \to Boolean$

$\&\& : Boolean \times Boolean \to Boolean$

We can replace many previous rules with application rule:

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

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

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))
  }
}
```

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

We can type check each function m,n,p in this global environment

Extending the Environment

```
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
    k = r + 2
    m(k, n(k))
  }
}
```

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

$\leftarrow \Gamma_0$

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

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

Type 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

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

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

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

Blocks with Declarations in the Middle

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

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

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

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

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

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

Type Checking Expression in a Body

```
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
     k =  r + 2
     m(k, n(k))
  }
}
```

Γ_0

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

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

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

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

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 }
  }
}
```

$\Gamma_0 = \{$
$(z, \text{Boolean}),$
$(u, \text{Int}),$
$(f, \text{Boolean} \rightarrow \text{Int})\ \}$

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

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

Exercise:

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

Solution

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