



# CSCI 742 - Compiler Construction

---

Lecture 24

Type Checking Implementation

Instructor: Hossein Hojjat

March 27, 2017

## Recap: Type Judgments and Type Rules

$$\boxed{\Gamma \vdash e : T}$$

If the (free) variables of  $e$  have types given by the type environment  $\Gamma$ , then  $e$  (correctly) type checks and has type  $T$

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

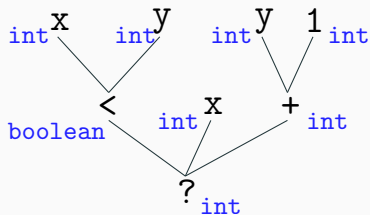
If  $e_1$  type checks in  $\Gamma$  and has type  $T_1$   
and ...

and  $e_n$  type checks in  $\Gamma$  and has type  $T_n$

then  $e$  type checks in  $\Gamma$  and has type  $T$

# Type Rules with Environment

`int x;`  
`int y;`  
`(x < y) ? x : (y + 1)` } Type Environment  $\Gamma$



Type Rules:

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

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

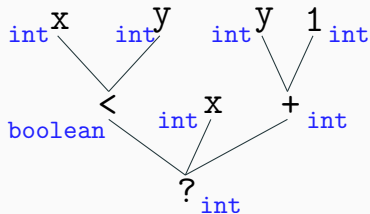
$$\frac{\Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : int}{\Gamma \vdash (e_1 < e_2) : boolean}$$

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

$$\frac{\Gamma \vdash b : boolean \quad \Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : T}{\Gamma \vdash (b ? e_1 : e_2) : T}$$

# Type Rules with Environment

`int x;`  
`int y;`  
`(x < y) ? x : (y + 1)` } Type Environment  $\Gamma$



$$\frac{\frac{(x : \text{int}) \in \Gamma}{\Gamma \vdash x : \text{int}} \quad \frac{(y : \text{int}) \in \Gamma}{\Gamma \vdash y : \text{int}}}{\Gamma \vdash (x < y) : \text{boolean}} \quad \frac{(x : \text{int}) \in \Gamma}{\Gamma \vdash x : \text{int}} \quad \frac{(y : \text{int}) \in \Gamma}{\Gamma \vdash y : \text{int}} \quad \frac{}{\Gamma \vdash \text{IntConst}(1) : \text{int}}}{\Gamma \vdash (x < y) ? x : (y + 1) : \text{int}}$$

# Type Checking in Practice

```
class Expression extends AST {  
  // ...  
  Type typeCheck(Environment gamma);  
}
```

- $\Gamma \vdash e : t$
- $t = e.typeCheck(gamma)$
- In the type environment  $\gamma$ , the expression  $e$  type checks with the type  $t$

# Type Checking in Practice

```
class ConditionalOperator extends Expression {
  Expression cond;
  Expression e1;
  Expression e2;
  // ...
  Type typeCheck(Environment gamma) {
    Type t = cond.typeCheck(gamma); // premise 1
    if (!t.equals(boolType))
      throw new TypeError("condition must be a boolean");
    Type t1 = e1.typeCheck(gamma); // premise 2
    Type t2 = e2.typeCheck(gamma); // premise 3
    if (!t1.equals(t2))
      throw new TypeError("type mismatch in conditional operator");
    else
      return t1;
  }
}
```

$$\frac{\Gamma \vdash b : \text{boolean} \quad \Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : T}{\Gamma \vdash (b ? e_1 : e_2) : T}$$

# Type Judgments for Statements

- Statements don't return any interesting value:  
we can think of them as computing a value of type `void`
- Typing judgment  $\Gamma \vdash s : \text{void}$  means  $s$  is a well-typed statement

$$\frac{\Gamma \vdash b : \text{boolean} \quad \Gamma \vdash s_1 : \text{void} \quad \Gamma \vdash s_2 : \text{void}}{\Gamma \vdash (\text{if } (b) \ s_1 \ s_2) : \text{void}}$$

## Type Rule for While Statement

$$\frac{\Gamma \vdash b : \text{boolean} \quad \Gamma \vdash s : \text{void}}{\Gamma \vdash (\text{while}(b) \ s) : \text{void}}$$



## Type Rule for Assignment Statement

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

## 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

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

- 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:

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

## Computing the Environment of a Class

```
class World {  
  int value; _____ (value, int),  
  String info; _____ (info, String),  
  int m(int x , int y) { _____ (m, int × int → int),  
    return x + y - 1;  
  }  
  int n(int x) { _____ (n, int → int),  
    if (info == "") return m(x + 1, 0);  
    else return 1;  
  }  
  boolean p(int r) { _____ (p, int → boolean)  
    int k = r + 2;  
    return m(k, n(value)) > 1;  
  }  
}
```

$\Gamma_0 = \{$

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

## Extending the Environment

```
class World {  
  int value; _____ (value, int),  
  String info; _____ (info, String),  
  int m(int x , int y) { _____ (m, int × int → int),  
    return x + y - 1;  
  }  
  int n(int x) { _____ (n, int → int),  
    if (info == "") return m(x + 1, 0);  
    else return 1;  
  }  
  boolean p(int r) { _____ Γ0 (p, int → boolean)  
    int k = r + 2; _____ Γ1 = Γ0 ⊕ {(r, int)}  
    return m(k, n(value)) > 1; _____ Γ2 = Γ1 ⊕ {(k, int)}  
  }  
}
```

- $\Gamma_2 = \Gamma_0 \oplus \{(r, \text{int}), (k, \text{int})\} = \Gamma_0 \cup \{(r, \text{int}), (k, \text{int})\}$

## Type Rule for Method Definitions

$$\frac{\Gamma \oplus \{(a_1, T_1), \dots, (a_n, T_n)\} \vdash e : T_r}{\Gamma \vdash \left( T_r \text{ fun } (T_1 a_1, \dots, T_n a_n) \{ \text{return } e \} \right) : \text{void}}$$

## Type Rule for Return

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

- Return statement produces no value for the current containing environment
- We can use type `void`
- How to make sure  $T$  is the return type of the current function?

## Type Rule for Return

- We record the expected return type of function in a special name
- Add special entry  $\{\text{ret} : T_r\}$  when we start checking function

$$\frac{\Gamma \oplus \{(a_1, T_1), \dots, (a_n, T_n), (\text{ret}, T_r)\} \vdash e : \text{void}}{\Gamma \vdash (T_r \text{ fun } (T_1 a_1, \dots, T_n a_n) \{e\}) : \text{void}}$$

- Look up this entry when we hit return statement

$$\frac{\Gamma \vdash e : T \quad \text{ret} : T \in \Gamma}{\Gamma \vdash \{\text{return } e\} : \text{void}}$$



# Overloading of Operators

- $\text{int} + \text{int} \rightarrow \text{int}$

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

Not a problem for type checking from leaves to root

- $\text{String} + \text{String} \rightarrow \text{String}$

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