CSCI 742 - Compiler Construction

Lecture 22

Type Checking Implementation

Instructor: Hossein Hojjat

March 21, 2018

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

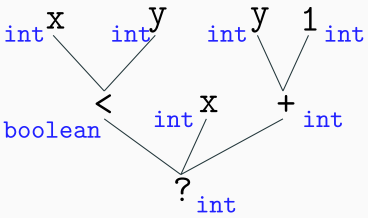
$$\boxed{\frac{\Gamma \vdash e_1 : T_1 \qquad \cdots \qquad \Gamma \vdash e_n : T_n}{\Gamma \vdash e : T}}$$

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

Type Rules with Environment

```
int x;
int y;
(x<y) ? x : (y + 1)
```

(`int x; int y;` — Type Environment Γ)



Type Rules:

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

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

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

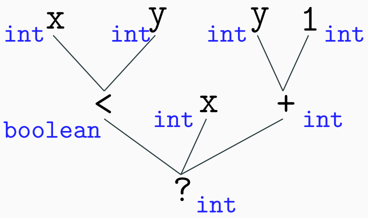
$$\frac{\Gamma \vdash e_1 : \texttt{int} \qquad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash (e_1 + e_2) : \texttt{int}}$$

$$\frac{\Gamma \vdash b : \texttt{boolean} \qquad \Gamma \vdash e_1 : T \qquad \Gamma \vdash e_2 : T}{\Gamma \vdash (b \mathbin{?} e_1 \; : \; e_2) \; : \; T}$$

Type Rules with Environment

```
int x;
int y;
(x<y) ? x : (y + 1)
```

(`int x;` and `int y;`: Type Environment Γ)



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

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;
  }
}
```

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

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 $\Gamma \vdash s : \texttt{void}$ means s is a well-typed statement

$$\frac{\Gamma \vdash b : \texttt{boolean} \qquad \Gamma \vdash s_1 : \texttt{void} \qquad \Gamma \vdash s_2 : \texttt{void}}{\Gamma \vdash (\texttt{if}\,(b)\ \ s_1\ \ s_2)\ :\ \texttt{void}}$$

Type Rule for While Statement

$$\frac{\Gamma \vdash b : \texttt{boolean} \qquad \Gamma \vdash s : \texttt{void}}{\Gamma \vdash \left(\texttt{while}\,\texttt{(}b\texttt{)}\;\; s\right)\; \texttt{:}\; \texttt{void}}$$

Type Rule for Assignment Statement

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

Type Rule for Function Application

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

Type Rule for Function Application

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

- We can treat operators as variables that have function type

$$\begin{aligned} \texttt{+} &: \texttt{int} \times \texttt{int} \to \texttt{int} \\ \texttt{<} &: \texttt{int} \times \texttt{int} \to \texttt{boolean} \\ \texttt{\&\&} &: \texttt{boolean} \times \texttt{boolean} \to \texttt{boolean} \end{aligned}$$

- We can replace many previous rules with application rule:

$$\frac{\Gamma \vdash e_1 : \texttt{boolean} \qquad \Gamma \vdash e_2 : \texttt{boolean} \qquad \Gamma \vdash \texttt{\&\&} : (\texttt{boolean} \times \texttt{boolean}) \to \texttt{boolean}}{\Gamma \vdash e_1 \texttt{\&\&} e_n : \texttt{boolean}}$$

Computing the Environment of a Class

$\Gamma_0 = \{$

```
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;
  }
}
```

$\}$

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

Extending the Environment

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

$\Gamma_0 = \{$

$(\mathtt{value}, \mathtt{int}),$

$(\mathtt{info}, \mathtt{String}),$

$(\mathtt{m}, \mathtt{int} \times \mathtt{int} \rightarrow \mathtt{int}),$

$(\mathtt{n}, \mathtt{int} \rightarrow \mathtt{int}),$

$(\mathtt{p}, \mathtt{int} \rightarrow \mathtt{boolean})$

$\}$

Γ_0

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

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

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

Type Rule for Method Definitions

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

Type Rule for Return

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

- 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 $\{\texttt{ret}:T_r\}$ when we start checking function

$$\frac{\Gamma \oplus \{(a_1, T_1)\ , \cdots ,\ (a_n, T_n), (\texttt{ret}, T_r)\} \vdash e : \texttt{void}}{\Gamma \vdash \left(T_r\ \texttt{fun}\ (T_1\ a_1, \cdots , T_n\ a_n)\ \{e\}\right)\ \ : \texttt{void}}$$

- Look up this entry when we hit return statement

$$\frac{\Gamma \vdash e : T \qquad \texttt{ret} : T \in \Gamma}{\Gamma \vdash \{\texttt{return}\ \ e\})\ \ : \texttt{void}}$$

Overloading of Operators

- `int` + `int` → `int`

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

Not a problem for type checking from leaves to root

- `String` + `String` → `String`

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