Lecture 13: Type Checking

CSCI 131
Fall, 2008

Kim Bruce

Type checking

- Most statically typed languages also include some dynamic checks.
  - array bounds.
  - Java's instanceof
  - typecase or type casts
- Pascal statically typed, but not strongly typed
  - variant records
- ML, Java strongly typed
- C, C++ not strongly typed

Type Compatibility

- When is x := y legal?

```
Type T = Array [1..10] of Integer;
Var A, B : Array [1..10] of Integer;
C : Array [1..10] of Integer;
D : T;
E : T;
```

- Name Equivalence$_A$ (*Ada)*
- Name Equivalence (*Pascal, Modula-2, Java)*
- Structural Equivalence (*Modula-3, Java arrays only)*

Structural Equivalence

- Can be subtle:

```
T1 = record a : integer; b : real end;
T2 = record c : integer; d : real end;
T3 = record b : real; a : integer end;
```

- Which are the same?

```
T = record info : integer; next : ^T end;
U = record info : integer; next : ^V end;
V = record info : integer; next : ^U end;
```

Type Checking & Inference

- Write explicit rules. Let a, b be expressions
  - if a, b: int, then a+b, a*b, a div b, a mod b: int
  - if a, b: int then a < b, a = b, a > b : bool
  - if a, b: bool then a andalso b, a orelse b: bool
  - ...

Formal Type-Checking Rules

- Can rewrite more formally.
- Expression may involve variables, so type check wrt assignment E of types to variables.
  - E.g., E(x) = int, E(b) = bool, ...

Hypothesis → $\dfrac{E(x) = t}{E \vdash x : t}$ ← *Conclusion*

$$\frac{E \vdash a : \text{int},\ E \vdash b : \text{int}}{E \vdash a+b : \text{int}}$$

Can write formally

Function Application:

$$\frac{E \vdash f: \sigma \rightarrow \tau, \quad E \vdash M : \sigma}{E \vdash f(M) : \tau}$$

Function Definition:

$$\frac{E \cup \{v{:}\sigma\} \vdash \text{Block} : \tau}{E \vdash \text{fun } (v{:}\sigma)\ \text{Block} : \sigma \rightarrow \tau}$$

Can write for all language constructs.
Based on context free grammar.
Can read off type-checking algorithm.

ML Type Inference

1. An identifier should be assigned the same type throughout its scope.
2. In an "if-then-else" expression, the condition must have type boolean and the "then" and "else" portions must have the same type. The type of the expression is the type of the "then" and "else" portions.
3. A user-defined function has type 'a → 'b, where 'a is the type of the function's parameter and 'b is the type of its result.
4. In a function application of the form f x, there must be types T and U such that f has type T → U and x has type T, and the application itself has type U.

Examples of Type Inference

- Use rules to deduce types:

```
val map = fn f => fn l =>
            if l = [] then []
                 else (f (hd l))::(map f (tl l))
```

- map: 'a → 'b *because function*
- f: 'a, `fn l => ...` : 'b, Thus 'b = 'c → 'd
- l: 'c, `if l = [] then ...` : 'd
- `...`