Assignment - 5

Name:- Sumit Kumar Yadav   Roll No.: 18CS30042

Ans 1:- Type:- It is a collection of computational entities that share some common property.

Type System: It is a tractable syntactic method for proving the absence of certain program behaviours by classifying phrases according to the kind of values they compute.

Advantages : (user)
→ naming and organizing concepts
→ making sure that bit sequence in computer memory are interpreted consistently.
→ providing information to the compiler about data manipulated by the program

Advantage of types:-
1. Program organization and documentation
→ separate types for separate concepts
• Represent concepts from problem domain
→ Document intended use of declared identifiers
• Types can be checked, unlike program comments.

2. Identify and prevent errors
→ Compile time or run time checking can prevent meaningless computation such as 3 + true - "Bill"

3. Support Optimization
→ short integers require fewer bits
→ Access components of structures by known effect.

Sumit (18CS30042)

Ans 2:- Type ~~Intext~~ Inference :- It is the process of determining the types of expressions based on the known types of some symbols that appear in them.

eg:-

$f\,x = 2+x$

$f :: Int \to Int$

⇒ + has type : $Int \to Int \to Int$

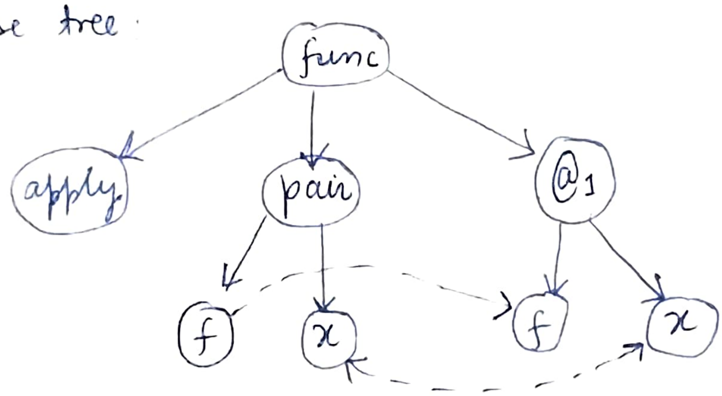
⇒ 2 has type : Int

⇒ since we are applying + to x, we need $x :: Int$

∴ $f\,x = 2+x$ has type: $Int \to Int$

Now, apply $(f, x) = f\,x$

parse tree :



→ assigning type variable to the nodes as follows :

apply :: t_0

pair :: t_3

f :: t_1

x :: t_2

$@_1$ :: t_6

→ Add constraints

t_1 = t_2 → t_6

t_0 = t_3 → t_6

t_3 = (t_1 , t_2)

Sumit (18CS30042)

→ Solving constraints :

$t\_0 = t\_3 \rightarrow t\_6$

$t\_1 = t\_2 \rightarrow t\_6$

$t\_3 = (t\_1 , t\_2)$

Now, replace t_3, we get

$t\_0 = (t\_1, t\_2) \rightarrow t\_6$

$t\_1 = t\_2 \rightarrow t\_6$

$t\_3 = (t\_1, t\_2)$

& replace t_1 : we get

$t\_0 = (t\_2 \rightarrow t\_6 , t\_2) \rightarrow t\_6$

$t\_1 = t\_2 \rightarrow t\_6$

$t\_3 = (t\_1 , t\_2)$

replace t_2 with t & t_6 with t1 :

$\therefore$ $t\_0 = (t \rightarrow t1 , t) \rightarrow t1$

$t\_1 = t \rightarrow t1$

$t\_3 = (t \rightarrow t1 , t)$

$\therefore$ apply :: $(t \rightarrow t1 , t) \rightarrow t1$.

Ans3:- Type Inference Algorithm:-

(i) Assign a type to the expression and each subexpression. for any compound expression or variable, use a type variable. for known operation or constraints such as + or 3, use the type that is known for this symbol.

(ii) Generate a set of constraints on types using the parse tree of the expression. These ~~expression~~ constraints reflect the fact that if a function is applied to an argument, for ex. then the type of the argument must be equal ~~to~~ the type of the domain of the function.

(iii) Solve these constraints by means of unification which is a substitution based algorithm for solving system of equations.

→ frame & solving type constraints: ~~by means of unificati~~

→ let ~~us~~ we write the type of ~~a~~ $m \times n$ matrix as $M \to N$. Now the rules of matrix algebra can be expressed as typing rules:

→ multiplication: ~~$\varepsilon \to A$:~~

$$\frac{\varepsilon \vdash A : s \to t \, , \; \varepsilon \vdash B : t \to u}{\varepsilon \vdash AB : s \to u}$$

→ Addition:

$$\frac{\varepsilon \vdash A : s \to t \, , \; \varepsilon \vdash B : s \to t}{\varepsilon \vdash A+B : s \to t}$$

→ Squaring:

$$\frac{\varepsilon \vdash A : s \to s}{\varepsilon \vdash A^2 : s \to s}$$

eg:- type of $(AB+CD)^2$

if $A : s \to t$, $B : u \to v$, $C : w \to R$, $D : y \to z$

soln:-

→ assign type variables.

$AB : a \to b$

$CD : c \to d$

$AB + CD : e \to f$

$(AB+CD)^2 : g \to h$

Sumit (18CS30042)

→ apply typing rule, we get

$t = u,\ a = s,\ b = v$ for AB

$x = y,\ c = w,\ d = z$ for CD

$a = c = e,\ b = d = f$ for AB+CD

$e = f = g = h$ for $(AB+CD)^2$

→ Solving the constraints

- $a = b = c = d = e = f = g = h = s = v = w = z$
- $t = u$
- $x = y$

→ hence,

$$\left.\begin{array}{l} A: a \to t \\ B: t \to a \\ C: a \to x \\ D: x \to a \end{array}\right\} \text{most general Solution}$$

Ans 4 :- Overloading is the method in which two or more implementation with different types are referred to as the same name.

eg: Overload resolution.

```
→ void f(char);          —(1)
  void f(int) = delete;  —(2)
  void f();              —(3)
  void f(int &);         —(4)

f(4);   // (1), (2) are viable (even though (2) is deleted)
        // (3) is not viable because the arguments lists doesn't match
        // (4) is not viable because we cannot bind a temporary to a non-constant l value reference.
```

Sumit (18CS30042)

→ pick the best viable candidate among viable functions.

eg:-
```
void f(int);   // (1)
void f(char);  // (2)
```
∴ f(4); // call (1) better conversion sequence

Now, example of overload resolution:

```
int g(double);              //F1
void f();                   // F2
void f(int);                // F3
double h(void);             // F4
int g(char, int);           // F5
void f(double, double=3.4); // F6
void h(int, double);        // F7
void f(char, char *);       // F8
```

then the call site to resolve is f(5.6);

Now, resolution:

→ candidate functions (by name) : F2, F3, F6, F8

→ viable functions (by # of params): F3, F6

→ best viable function (by type double - exact match

∴ F6

Ans 5:- Parametric polymorphism:- In programming languages and type theory, parametric polymorphism is a way to make a language more expressive, while still maintaining full static type-satisfy. A function

Sumit (18CS30042)

or a data type can be written generically so that it can handle values identically without depending on type.

parametric polymorphism may be implicit or explicit:

→ In explicit parametric polymorphism, the program ~~text~~ text contains type variables that determine the way that a function or other value may be treated polymorphically ie, C++

→ Haskell polymorphism is called implicit because programs that declares and use polymorphic functions don't need to contain types → the type inference algorithm compute when a function is polymorphic and compute the instantiation of type variable as needed.

<u>r-value Reference</u> :- r-value refers to the data value that is stored at some address in memory. 'rvalue' reference extend the lifespan of the temporary objects to which they are assigned. non-const "r-value" reference allow you to modify the "r-value".

eg - in C++,

```
→ int a;
  int &< a-ref2 = a;      // an r-value reference

→ template <class T>
  typename remove-reference <T> :: type && 
  move (T && a) {
      return a;
  }
```

It accepts an lvalue or rvalue and return it. an rvalue with triggering a copy.