POPL Assignment IV

Ilish Agarwal
18CS30021

Q1) In general, a <u>type</u> is a collection of computational entities that share some common properties.

For eg, INT, BOOL, BOOL→INT, etc

A <u>type system</u> is a tracable syntactic method for proving the absence of certain program behaviours. This is done by classifying phrases according to the kind of values (types) they compute.

Some advantages of using a type system are:

(i) Identify and prevent errors
→ Compile-time or run-time checking can prevent meaningless computation.

(ii) Abstraction
→ enforces disciplined programming

(iii) Language safety
→ Protects abstractions. Increases

Itish Agarwal
18CS30021

portability

(iv) Increases efficiency

→ Distinguishes between integer arithmetic and real-valued arithmetic.

Q2) Type inference is the process of determining the type of expressions based on the known type of symbols that appear in them. It uses type-variables as place-holders for types that are not known.

- Type inference by hand weaving:

Given $fx = 5 * x$

* has type Int → Int → Int

and,

5 has type Int

To find: type of f

Since we are applying * to x, we need x : Int

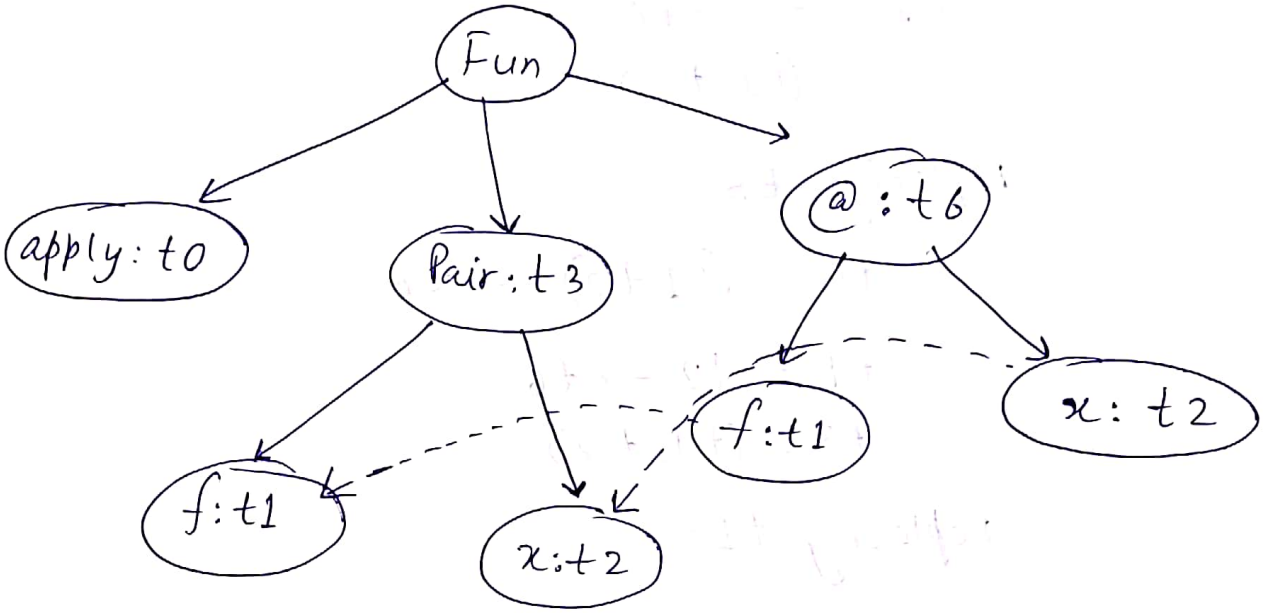
Hence the function:

$fx = 5 * x$ has type Int → Int.

PTO

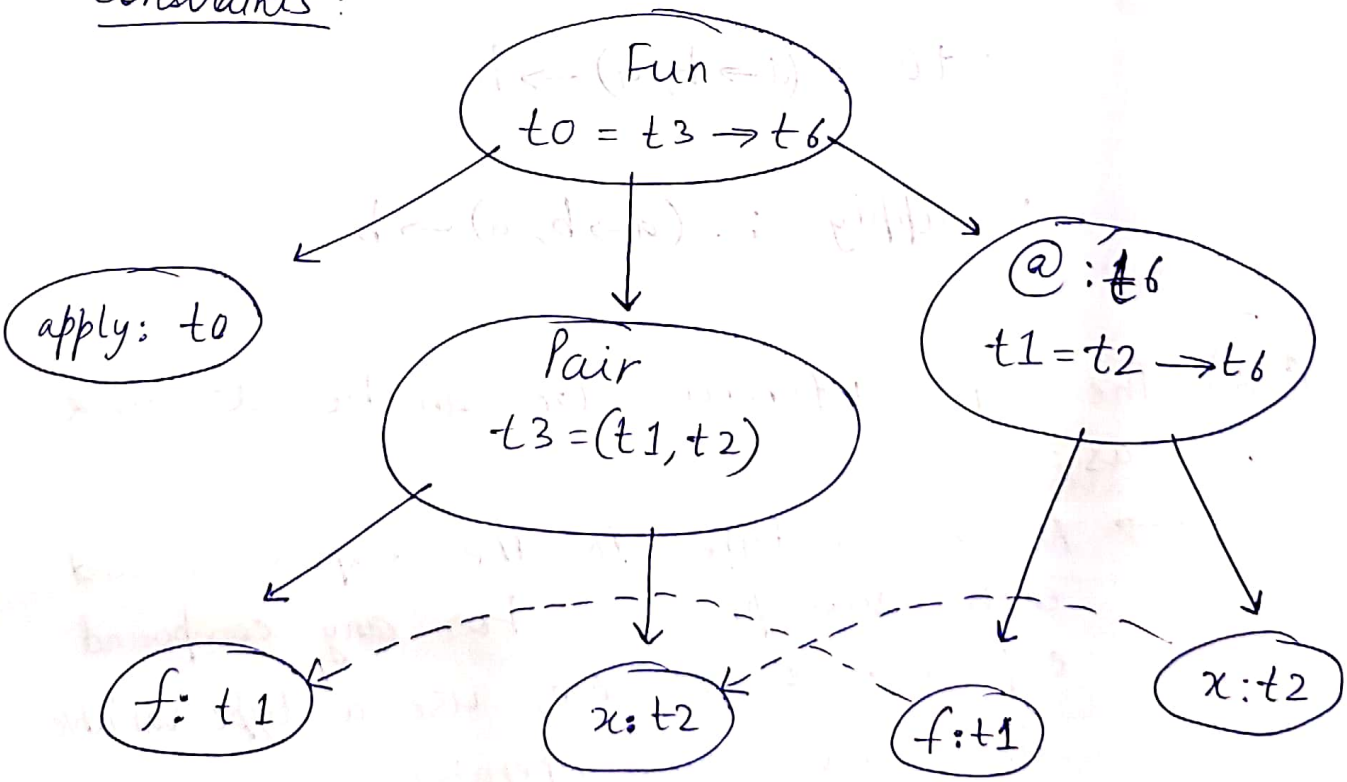
[Itish Agarwal 18CS30021]

• Type inference algorithm for apply (f, x) = fx



Parse tree with type constraints

Constraints :



Itish Agarwal
18CS30021

Now, simplifying constraints

$t0 = t3 \to t6$

$t1 = t2 \to t6$

$t3 = (t1, t2)$

replacing t3:

$\therefore t0 = (t1, t2) \to t6$

$t1 = t2 \to t6$

$t3 = (t1, t2)$

replacing t1:

$\therefore t0 = (t2 \to t6, t2) \to t6$

replace t2 with a, t6 with b:

$\therefore t0 = (a \to b, a) \to b$

$\therefore$ apply $:: (a \to b, a) \to b$

03) The type inference algo can be described as:

→ Assign a type to the expression and each subexpression. For any compound expression or variable, use a type variable. There are some operations and constants whose types are known. Use those types for them.

→ Now, use the parse tree to generate constraints. For eg, if a function is applied to an argument, then the constraint says that the argument must be equal to the domain of the function.

→ Use unification to solve these constraints. Unification is a substitute based algorithm for solving system of equations.

To solve: $(A + B) * C$
where A, B, C are matrices. $A: s \to t$,
$B: u \to v$,
$C: y \to z$

Take, $f(A, B, C) = (A + B) * C$

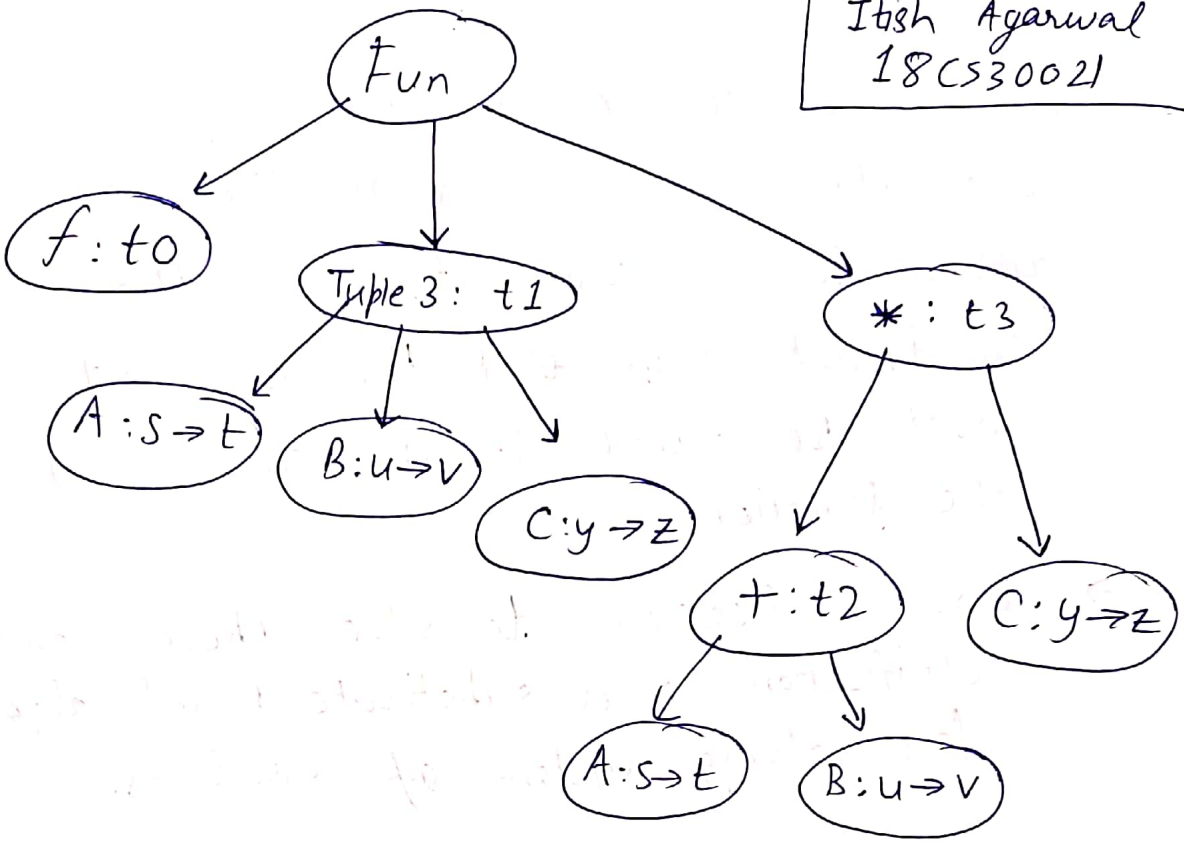
Here,

- $\to$: denotes the 'x' used for matrix dimension
- $\Rightarrow$: denotes application

Itish Agarwal
18CS30021

PTO

Itish Agarwal
18CS30021



Constraints required to find type of $(A+B)*C$:

$t1 = (s \to t, u \to v, y \to z)$

$t2 = (a \to b) \Rightarrow (a \to b) \Rightarrow (a \to b)$

$t3 = (c \to d) \Rightarrow (d \to e) \Rightarrow (c \to e)$

$\therefore$

$s = a = u$ (since $a \to b = s \to t$)

$t = b = v$

Hence,

$a = c = s = u$ (since $a \to b : c \to d$)

$b = t = v = d = y$ ($y \to z = d \to e$)

$e = z$

Itish Agarwal
18CS30021

Hence

A : a → b
B : a → b
C : b → e

[This is most general typing. Any values of a, b, c makes (A + B) * C well formed]

Q 4) Overloading is the method in which two or more implementations with different types are referred to by the same name.

Assume that following are declared:

```
void f(char);          (1)
void f(int);           (2)
void f();              (3)
void f(int &);         (4)
template<class T> void f(T);   (5)
void f(int, char);     (6)
```

Suppose we perform f(T);

Steps

1) Find candidate functions via name search

```
void f(char);          (1)
void f(int);           (2)
void f();              (3)
void f(int &);         (4)
template <class T> void f(T);   (5)
void f(int, char);     (6)
```

Itish Agarwal
18CS30021

2) Remove unviable functions:

```
void f(char);          (1)
void f(int);           (2)
template <class T> void f(T);   (5)
```

3) Pick best viable candidate via implicit conversion sequence:

```
void f(int);           (2)
template <class T> void f(T);   (5)
```

No distinction in step 1), 3).

4) Pick more specialized function

void f(int); (2) is more specialized. Hence void f(int); will be used. This is an example of overload resolution.

Q5.) → In programming languages and type theory, Parametric Polymorphism is a way to make a language more expressive, while still maintaining full static type-safety. A function or a data-type can be written generically so that it can handle values identically without depending on type.

Two types of parametric polymorphism:

(i) implicit parametric polymorphism.

Here the programs that use it do not need to contain types, it is inferred from the type inference algo.

(ii) explicit parametric polymorphism:

The program text contains type variables that determine the way a function/ value is treated.

→ An rvalue is an expression that can only appear on the right hand side of an assignment.

```
int a, b, c;
a = b = c = 3;
c = a * b
```

Here, 'a * b' is an rvalue. That is, it cannot appear in left side of an expression.

→ In C++ semantics, an rvalue is an expression that is not lvalue.

Itish Agarwal
18CS30021

Itish Agarwal
18CS30021

→ If X is any type, then X&& is called an rvalue reference to X. The ordinary reference X& is called lvalue reference.

An rvalue reference behaves like an lvalue reference with certain exceptions.

For example:

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

void temp(int& a) {
    cout << "Lvalue reference\n";
}

void temp(int&& a) {
    cout << "Rvalue reference\n";
}

int temp3() {
    return 20;
}

int main() {
    int a = 55;
    temp(a);
    temp(temp3);
}
```

output: Lvalue reference
Rvalue reference

Itish Agarwal
18CS30021