Types and Type Systems

What is a type?

1) The set of all legal values for the type. (denotational definition)

A string is {“”, “a”, “b”, “c”,… “aa”, “ab”, .. “a?>\_”, }

2) Either a primitive type of the language or a compound type built up of other types.

A string is a sequence of char values.

3) All of the operations that we can do on values of the type.

A string is defined by: +, length(), charAt(index), substring(index1, index2), …

We can define the type system for a programming language by:

1) type equivalence

2) type compatibility

3) type inference

Type equivalence: When are two types considered to be the same. We can use values of each type interchangeably.

a) structural equivalence: two types are the same if they have the same structure. (Made up of the same pieces combined in the same way.)

ex: array[0...9] of integer

array[1...10] of integer

Are these structurally equivalent?

Ex: type rgb\_color = struct { r: 8-bit int; g: 8-bit int, b: 8-bit int }

type hsv\_color = struct { h: 8-bit int; s: 8-bit int, v: 8-bit int }

Are these structurally equivalent?

b) name equivalence

strict name equivalence

loose name equivalence

Are aliases of type considered to be equivalent?

typedef fahrenheit int;

typedef celsius int;

type fahrenheit = int;

Are fahrenheit, celsius, and int equivalent to each other?

Strict name equivalence: no

Loose name equivalence: yes

In name equivalence, two types must have the same name to be considered equivalent.

Ex: array [0..9] of int → has no name

type A = array [0..9] of int

A a;

A b;

a = b;

2) Type compatibility: We can use type B where type A is expected when B is compatible with type A.

type coercion: implicit type casting.

We will coerce the value of type B into type A so that we can use type B where A is expected.

Ex: double x = 10

int is compatible with double, the int value is coerced into double.

two kinds of type casts and type coercions: where the data of the value must be changed, and where the data is not changed (nonconverting type cast).

Ex: In Java, we define when a type is “wider” or “narrower”, narrower types are compatible with wider types, and Java uses type coercion to covert the narrower into the wider.

Type inference: Given an expression of the language, can we infer the type of the expression.

Ex: p AND q => structure that contains p and q

p OR q => union that can be type p or type q

p → q => a function that takes type p as input and produces type q

To infer a type, we can do discrete math type proof:

double f(int x) { type of f is int → double

int g(char c) { type of g is char → int

f(g(‘a’)) is the same as doing (char → int) (int → double) ==> char → double “hypothetical syllogism”

strict typing (strong typing): the language prohibits use of values that violate the type system

static typing: type rules are enforced by the compiler.