Types

CS 329 Lecture 3

Aug 1, 2007

Rushikesh K. Joshi

Types and Values

Types can be considered as sets

The members of the set that represents a type represent all possible values of the type

Value Assignment

```
T var;
var = v1;
var = v2;
```

A variable of type T can be assigned a value that is a member of the set defining type T

Assignment statement can be used to change the assignment of a value to a variable of given type; only that the value should be from the set defining the type.

An example

bool = {true, false}

b1 = true;
b2 = false;
...
b1=b2

The type is *bool*

b1, b2 are the only possible values of this type

Cardinality of a type

The count of all possible discrete values

bool = {true, false}
#bool = 2

Week = {Mon,Tue,Wed,Thu,Fri,Sat,Sun}
#Week = 7

Primitive Types

Sets of discrete values

To specify a type, simply enumerate all its values
e.g. int = {-MAX,..,0...,+Max}

e.g. bool, int, float, char, short int, unsigned int, enumerated data types etc.

Language definitions provide some standard primitive types from which composite types such as structures, functions, lists can be constructed

Composite Types

These are constructible from other types

e.g. struct xyz {
    int i;
    char c;
}

A structure or a record is thus a composite formed by taking a cross product of multiple types

Composite Types: Product types

```
Record R1 {
    T1 v1;
    T2 v2;
}
```

R1 = T1 X T2

#R1 = #T1 x #T2

Example: if T1=T2=bool, #R1 = 4.

R1 = { (t,t), (t,f), (f,t), (f,f) }

The cardinality again represents the count of all possible values of the given type.

Composite Types: Function types

A function *T2 f (T1)* is a mapping from set T1 to set T2. i.e. f computes a value of type T2 given a value of type T1 as input parameter.

f: T1 --> T2

If T1 is boolean, and T2 is also boolean, we have
f={{(t->t),(f->t) },{(t->t) (f->f)},{(t->f),(f->t)},{(t->f),(f->f)}}

how many different function bodies can you write against a function signature T2 f(T1)?

ans: $\#f = (\#T2)^{(\#T1)}$

Cardinality and values of a function type

The elements of the set corresponding to a function type are all possible mappings for a given function signature.

A function body is merely one of the many possible values for the function type.

cardinality of a function type is the number of discrete function bodies (i.e. mappings) for the function type.

Thus we can represent a function body as a value of a function type, or in other words, a program is a value and its specification, a type.

Composite types: Array types

int A[10]

It can be modeled as a function that maps integers from range 1..10 to int

so type of array A is T1-->int, where T1={1..10}

default initializer is the default mapping.
Cardinality of an array type represents the number of possible valuations of the array
e.g. 1111111111 is one of the many possible valuations.
Any other mapping can be used as a value of A, if the mapping is a valid value of the type that defines A.

A function type represents an array more naturally than a product type since we have the associated operation of indexing. Record elements are accessed by their names, whereas array elements are accessed by their indices.

Type Errors

Consider Type
int A[10]
In 'C', if you access element A[10], it constitutes an error. Since the type is undefined on index=10, such an access is called type error, or type violation.

Depending on the design of the programming language, a type error may get detected at compile time, or at runtime, or go undetected by the language's runtime environment, and may eventually get trapped inside the operating system such as through a segmentation fault.