Types and Inheritances

66.648 Compiler Design Lecture (03/04//98)

Computer Science

Rensselaer Polytechnic

Lecture Outline

- Unification
- Examples
- Inheritances
- Administration

Unification Algorithm

```
Boolean unify(node m, node n)
 { s= find(m); t = find(n); /* find the root node */
   if (s==t) return true; else if (s and t represent the
    same basic type)  return true;
   else if s is an op-node with children s1 and s2
    and t is an op_node with children t1 and t2
 { union(s,t); return unify(s1,t1) and unify(s2,32) }
else if s or t represents a variable { union(s,t);
return true;} else return false; }
```

Example

(a1->a2) x list(a3) -> list(a2)

(a3->a4)x list(a2) -> a5

```
fun length(lptr) = if null(lptr) then 0 else
              length(tl(lptr))+1 end;
```

list(a) -> integer

Examples

Fun temp(a,b) = a(b);

Fun temp1 a b = a(b);

Fun temp2 a b = temp1 b a;

Fun x1(a,b) = a;

Fun X2(a,b) =b;

Fun X3(a,b) =X1(a,X2(a,b));

Fun map(f,lptr) = if null(lptr) then nil else

cons(f(hd(l)) , map(f, tl(l)));

Inheritance

Inheritance extends abstract data types by allowing for super type / subtype relationships. Inheritance supports incremental code reuse by allowing one to define a suntype with incremental changes to the procedural and data abstractions in the base type.

Example:

```
public class animal {

}
```

Inheritance-Contd

```
Public class brontasaurus extends dinosaur {
int neck_length;
   private int compute_neck_length { ..}
   public void print_neck_length() {..}
}
```

The class brontasaurus is a subtype of class dinosaur. An instance of brontasaurus contains all the fields of dinosaur. All public and private functions are available to brontasarus.

Inheritance - Contd

Single Inheritance: When the type system restricts each type to have at most one super type.

Multiple Inheritance: When the type system permits a type to have more than one super type.

Inheritance Hierarchy: The type/supertype relationship defined by extends construct is called an inheritance hierarchy. For a type system with single inheritance, the inheritance hierarchy can be modeled as a forest trees.

Access Rules

1. Scope visibility: variables and fields of type class T1 can be declared anywhere in a program that a declaration is permitted and T1's definition is visible.

Note: If the language permits two classes T1 and T2 to contain functions with the same name when class T1 is an ancestor of T2, then name resolution follows standard scope rules by treating T2 is a scope contained with T1.

2. Data access: public data-fields of class T1 are accessible.

Access - Contd

3. Access to private procedures: private procedures of an class T1 can only be invoked by functions in class T1 or in T2 that is a descendant of T1 in the inheritance hierarchy.
4. Access to public functions: public functions of T1 can be invoked by any function that can declared as an instance of T1.
5. Automatic Type Conversion:an expression of type class T2 is coerced into an ancestor of class T1 when required but not vice-versa.

Aggregation

A class T2 is an aggregation of class T1 if T2 contains one or more fields of type T1. Unlike in inheritance class T2 cannot access private functions in T1.

```
Public class Jurassic_Park  {
 dinosaur animals[];
}
```

We cannot invoke private functions from Jurassic Park.

Multiple Inheritance

Some languages permit multiple inheritance I.e., allow a class to be an extension of mutiple classes. This leads to more complicated semantics for sub typing.

```
Public class flying extends dinosaur, fly {
}
```

Multiple Inheritance

The inheritance hierarchy becomes a directed acyclic graph in the case of multiple inheritance. If a function is defined both classes, then the leftmost parent leading to the definition of the super type is taken.

Java Program

```
Public class test1 {
public static void main(String argsv[]) {
int I;
I = 234;
I = 5000;
}
}
```

Byte Code

Class File Format

Page 84 of the Virtual Machine Book

```
ClassFile {
u4 magic;
u2 minorversion;
u2 majorversion;
u2 constant_pool_count;
cp_info constant_pool[constant_pool_count-1];
u2 access_flags;
```

Class File Format- Contd

U2 this_class;

u2 super_class;

u2 interfaces_count;

u2 interfaces[interfaces_count];

u2 fields_count;

field_info fields_count[fields_count];

u2 methods_count;

methods_info methods[methods_count];

Class File Format- Contd

U2 attributes_count;

attribute_info attributes[attributes_count];

}

Example:

Type Signatures

B byte

C Char   Field Descriptor

D Double

F Float

I Int

J Long

L<classname> an instance of a class

S Short

Type Signatures-Contd

Z Boolean

[one array of one dimension

Example: double d[][];

[[D

Method Descriptor:

(Parameter Descriptor *) Return Descriptor

Eg: Object Mymethod(int I, doble d)
(ID)Ljava/lang/Object

Constant Pool

Cp_info {

u1 tag;

u1 info[];}

Tags are 1 unicode, 3 integer, 4 float, 5 long, 6 double, 7 class, 8 String, 9 Fieldref, 10 Methodref, 11 InterfaceMethodref and 12 NameandType.

Comments and Feedback

Project 2 is out. Please start working. PLEASE do not wait for the due date to come.

We have finished chapter 6 and looked at the relavant portion of Java.. Please keep studying this material. It may look difficult.