Formal Methods
Seminar 2

First Part of the Programming Assignment

Design in Ocaml the Abstract Syntax Tree for a small object-oriented language called CoreJava.

The syntax of CoreJava is described in the following slides.

Syntax of CoreJava

A CoreJava program P consists of a list of class declarations as follows:

P ::= def+

def+ ::= def |def1; def2;...;defn

The program contains at least one class declaration.

The last class declaration must contain a method called “main” which is the starting execution point for the CoreJava program.

Syntax of CoreJava

A class declaration def consists of the following:

def ::= class cn1 extends cn2 { field* # meth*}

where cn1 is the current class name and cn2 is the name of the parent class

The body of the class consists of a list of fields declarations followed by a list of methods declarations. The fields list is separated by "#" from the methods list. The fields list is either empty or field+ ::= field1;field2;...;fieldn. The methods list is either empty or meth+ ::= meth1;meth2;...;methn.

Syntax of CoreJava

A field declaration consists of the following:

field ::= typ fn

where typ is the type of the field and fn is the name of the field

A type typ can be either a primitive type, or a class name or the bottom (the type of null).

typ ::= prim | cn | _|_

prim ::= int | float | bool | void

Syntax of CoreJava

A method declaration consists of the following:

meth ::= typ1 mn((typ pn)*) {e}

where typ1 is the type of the method result, mn is the method name, (typ pn)* is the list of method parameters.

The list of method parameters can be either empty or (typ pn)+ ::= typ1 pn1, typ2 pn2,...,typn pnn

The body of the method is given by the expression e.

Syntax of CoreJava

An expression is defined as follows:

e ::= null (null value)

| kint (a constant value)

|kfloat

| true | false

| () (a value of type void)

| v (a variable name)

| v.f (a field f of an object denoted by v)

Syntax of CoreJava

An expression is defined as follows:

e ::= ...

| v=e (variable assignment)

|v.f=e (object field assignment)

| { (typ v) e1} (local variable declaration such that v has type typ and its scope is the expression e1)

| e1;e2 (a sequence such that e2 is executed only after e1 is executed)

| if v then {e1} else {e2} (conditional where the condition is a variable)

Syntax of CoreJava

An expression is defined as follows:

e ::= ...

| e1 opint e2 (arithmetic expressions for ints where opint ::= +|-|*|/)

|e1 opfloat e2 (arithmetic expressions for floats where opfloat ::= +.|-.|*.|/.)

| e1 && e2 (logical expressions)

| e1 || e2

| !e (negation)

Syntax of CoreJava

An expression is defined as follows:

e ::= ...

| new cn(v*) (creation of an instance object of the class cn. The fields of the new object are initialized by the list of variables v*. The fields are intialized in the order defined in the declaration of class cn)

|v.mn(v*) (method call where v is the method receiver and the list of variables v* are the current arguments initializing the method parameters)

| while v {e} (loop where the condition is a variable)

Syntax of CoreJava

An expression can also be

e ::= ...

| e1 opcmp e2 (relational expressions where opcm ::= <|<=|==|!=|>|>=)

| (cn) v (cast expression)

| v instanceof cn

Discussion on the Ocaml implementation

The following slides show how you can represent the Abstract Syntax Tree of CoreJava in Ocaml.

A program can be represented as a dictionary of classes (association list or you may want to consult Chapter 13 from realworldocaml.org):

```
type progr= (string*classDecl) list

type classDecl =
  (string*string*fldDeclList*mthDeclList)

type fldDeclList = fldDecl list

type fldDecl = typ * string

type mthDeclList = mthDecl list

type mthDecl =(typ*string*fPrmList*blkExp)

type fPrmList = fPrm list

type fPrm = (typ*string)
```

```
type typ = Tprim of tPrim
                | Tclass of string | Tbot
type tPrim = Tint | Tfloat | Tbool| Tvoid

type blkExp = Bvar of typ*string*exp
                    | Bnvar of exp
type val = Vnull | Int of int | Float of float
                | Bool of bool | Vvoid
type exp =  Value of val
                |Var of string | Vfld of string*string
                | ....(see next slide)
```

```
type exp =  ...
          | AsgnV  of string*exp
          | AsgnF of string*string*exp
          | Blk of blkExp
          | Seq of exp*exp
          | If of string*blkExp*blkExp
          | AddInt of exp*exp
          | MulInt of exp*exp
          | .... (please continue yourself the first
  assignment)
```

AST Example

The following Core-Java expression:

if m then {z=z+1} else {z=2}

Is represented in Ocaml as the following AST:

```
Let ifexp1 =(IF  (“m”,
            (Blk (Bnvar (AsgnV (“z”, (AddInt  (Var “z”,(Value (Int
1)))))))),
            (Blk (Bnvar (AsgnV (“z”,(Value (Int 2)) ))))
              )
```

Please represent the following Core-Java examples as ASTs in Ocaml:

Core-Java Example

```
class A extends Object{
int f1;
#
int m1(int a, int b) { (int c)
                        c=a+b;this.f1=this.f1+c;c};
}
```

Core-Java Example

```
class B extends A{
A f2;
#
A m2(A x, A y) {(A z) { (int n)
                        n=x.m1(1,2)-y.m1(2,1);
                        {(bool m)  m=(x.f1-y.f1)>n;
                            if m then {z=new A(m)} else {z=new A(n)}
                        }
                    };this.f2=z;z
}
```

Core-Java Example

```
Class Main extends Object{  #
 Void main(){ (B o1) o1=new B(0,null);
                    { (A o2)  o2=new A(2);
                        { (A o3) o3=new A(3);
                              o2 =o1.m2(o2,o3)
                        }
                    }
               }
}
```