Virtual Machines
Lecture 6 – CoreJava Operational Semantics and Second Assignment

Overview

1. Discussion about the First Assignment – CoreJava AST representation and a small revision

2. Second Assignment – CoreJava Operational Semantics

First Assignment Discussion

–

CoreJava Abstract Syntax Tree

Syntax of CoreJava—small revision

An expression can also be

e ::= ...

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

| (cn) v (cast expression)

| v instanceof cn

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 the first
  assignment)
```

Second Assignment

–

CoreJava Operational Semantics

Second Assignment – 25% of the final grade

Please implement in Ocaml an Interpreter for CoreJava language according to the CoreJava operational semantics.

The operational semantics of CoreJava is described in the following slides

CoreJava Operational Semantics

The operational semantics is defined as a small-step rewriting relation from machine states to machine states.

A machine state (configuration) consists of the following tuple:

<H,V,e> where

- V is the current variable environment (stack)
- H is the current heap
- e is the current program

CoreJava Operational Semantics

The evaluation judgement of the operational semantics (small-step rewriting relation) is of the form:

<H,V,e> --> <H',V',e'>

where: - H is the heap before the evaluation

- H' is the heap after the evaluation

- V is the variable environment before the evaluation

- V' is the variable environment after the evaluation

CoreJava Operational Semantics

The evaluation rules are applied until we get a value such that <H,V, value>.

In CoreJava a value can be:

val = Enull | Int of int | Float of float

| Bool of bool | Evoid

In addition during the evaluation we can get a location (a heap memory address) as value.

Therefore we extend exp as follows:

val = ... | Loc of int

CoreJava Operational Semantics

The variable environment **V** is a partial function from variable name to a pair. The pair consists of the declared type of the variable and the assigned value.

It can be represented as an association list

(string*typVal) list

type typVal = typ*val

But this association list is organized as a stack, we push in front of the list and we pop from the front of the list

CoreJava Operational Semantics

To avoid variable name duplication, we assume that the local variables of the blocks and the arguments of the functions are uniquely renamed in a preprocessing phase.

For simplicity CoreJava programmer can avoid the variable name duplication.

CoreJava Operational Semantics

The heap H is a partial function from locations to object values. An object value consists of the class name and a field environment. The field environment is a partial function from field name to a pair consisting of the field declared type and the assigned value .

They can be represented as follows:

H is (int *objVal) list

type objVal= string*fldEnv

type fldEnv = (string*typVal) list

CoreJava Operational Semantics

We require an intermediate expressions for the small-step dynamic semantics to follow through. The syntax of intermediate expressions is thus extended from the original expression syntax as follows:

exp = ... | ret(v,e)

CoreJava Operational Semantics

The expression **ret(v, e)** is used to capture the result of evaluating a local block, or the result of a method invocation. The variable associated with ret denotes either a block local variable or a method receiver or a method parameter. This variable is popped from the variable environment at the end of the block's evaluation. In the case of a method invocation there are multiple nested rets which pop off from the variable environment the receiver and the method parameters at the end of the method's evaluation.

CoreJava Operational Semantics

- In the following we present the evaluation rules of all CoreJava.
- The presentation is not so formal as in the literature
- The rules have the following form

conditions to be met (IF conds to be met)

---------- <==> THEN

<H,V,e> --> <H',V',e'> <H,V,e> --> <H',V',e'>

CoreJava Operational Semantics

$$\frac{\text{is\_value}(e)}{\langle H, V, e\rangle \rightarrow \text{stop}}$$

- this rule ensures the termination of the evaluation
- is_value checks if the current exp is a val

CoreJava Operational Semantics

(var is defined in V) and

(val=getVal var V)

<H,V, var> --> <H,V, val>

- getVal function extracts the value associated to the variable v inside the variable environment

CoreJava Operational Semantics

(var is NOT defined in V)

<H,V, var> --> error

CoreJava Operational Semantics

(var is defined in V) and (loc= getVal var V) and

(loc is a location) and (loc is defined in H) and

(fldE = getFieldE loc H) and

(fld is defined in fldE)

and (val = getVal fld fldE)

<H,V, var.fld> --> <H,V, val>

GetFieldE extracts the field environment of an object value from the location loc of the heap H

CoreJava Operational Semantics

(var is defined in V) and (loc= getVal v V) and
(loc is a location) and (loc is defined in H) and
(fldE = getFieldE loc H) and
(fld is NOT defined in fldE)

<H,V, var.fld> --> Error

CoreJava Operational Semantics

(var is defined in V) and (loc= getVal v V) and

(loc is a location) and (loc is NOT defined in H)

<H,V, var.fld> --> Error

CoreJava Operational Semantics

(var is defined in V) and (loc= getVal v V) and (loc is NOT a location)

<H,V, var.fld> --> Error

CoreJava Operational Semantics

(var is NOT defined in V)

<H,V, var.fld> --> Error

CoreJava Operational Semantics

$$\frac{\langle H, V, e\rangle \rightarrow \langle H', V', e'\rangle}{\langle H, V, var = e\rangle \rightarrow \langle H', V', var = e'\rangle}$$

CoreJava Operational Semantics

(var is defined in V) and (typ_var = getType var V)

and (typ_val = getType val) and

(is_subtype typ_val typ_var) and (update V var val)

<H,V, var=val> --> <H,V',()>

- update function updates the value associated to var inside the variable envirohment
- getType extracts the type of a value or a variable
- is_subtype t1 t2 checks if t1 is a subtype of t2

CoreJava Operational Semantics

(var is defined in V) and (typ_var = getType var V)

and (typ_val = getType val) and

(NOT is_subtype typ_val typ_var)

<H,V, var=val> --> Errror

CoreJava Operational Semantics

(var is NOT defined in V)

<H,V, var=val> --> Error

CoreJava Operational Semantics

is_subtype t1 t2 = match (t1,t2) with

|(Tprim ta, Tprim tb) -> if ta=tb then true else false

| (Tbot, Tbot) -> true

| (Tbot, Tclass cn) -> true

|(Tclass cn1,Tclass cn2) -> check if cn1=cn2 or cn1 is derived from cn2

| (_,_) -> false

CoreJava Operational Semantics

$$\frac{\langle H, V, e \rangle \rightarrow \langle H', V', e' \rangle}{\langle H, V, var.f = e \rangle \rightarrow \langle H', V', var.f = e' \rangle}$$

CoreJava Operational Semantics

(var is defined in V) and (loc= getVal var V) and
(loc is a location) and (loc is defined in H) and
(fldE = getFieldE loc H) and (f is defined in fldE)
and (typ_val = getType val) and
(typ_fld = getType f (getClassName loc H)) and
(is_subtype typ_val typ_fld) and(update fldE f val)

<H,V, var.f=val> --> <H',V,()>

CoreJava Operational Semantics

(var is defined in V) and (loc= getVal v V) and
(loc is a location) and (loc is defined in H) and
(fldE = getFieldE loc H) and (f is defined in fldE)
and (typ_val = getType val) and
(typ_fld = getType f (getClassName loc H)) and
(NOT is_subtype typ_val typ_fld)

<H,V, var.f=val> --> Error

CoreJava Operational Semantics

(var is defined in V) and (loc= getVal v V) and
(loc is a location) and (loc is defined in H) and
(fldE = getFieldE loc H) and
(f is NOT defined in fldE)

<H,V, var.f=val> --> Error

CoreJava Operational Semantics

(var is defined in V) and (loc= getVal v V) and (loc is a location) and (loc is NOT defined in H)

<H,V, var.f=val> --> Error

CoreJava Operational Semantics

(var is defined in V) and (loc= getVal v V) and (loc is NOT a location)

<H,V, var.f=val> --> Error

CoreJava Operational Semantics

(var is NOT defined in V)

<H,V, var.f=val> --> Error

CoreJava Operational Semantics

<H,V, {e}> --> <H,V,e>

CoreJava Operational Semantics

$$\frac{V' = (v, (typ, init(typ))) :: V}{\langle H, V, \{(typ\ v)\ e\} \rangle \rightarrow \langle H, V', ret(v, e) \rangle}$$

init(typ) returns the default value corresponding to the given type

CoreJava Operational Semantics

$$\frac{\langle H, V, e\rangle \rightarrow \langle H', V', e'\rangle}{\langle H, V, ret(v,e)\rangle \rightarrow \langle H', V', ret(v,e')\rangle}$$

CoreJava Operational Semantics

$$\frac{V = (v,\_)::V' \quad \text{and} \quad is\_value(val)}{\langle H, V, ret(v,val)\rangle \rightarrow \langle H, V', val\rangle}$$

CoreJava Operational Semantics

$$\frac{\langle H, V, e1 \rangle \rightarrow \langle H', V', e1' \rangle}{\langle H, V, e1; e2 \rangle \rightarrow \langle H', V', e1'; e2 \rangle}$$

CoreJava Operational Semantics

$$\frac{\text{is\_value}(e1)}{\langle H,V,\ e1;e2\rangle \rightarrow \langle H,V,e2\rangle}$$

CoreJava Operational Semantics

<H,V, Error;e2> --> Error

CoreJava Operational Semantics

$$\frac{\langle H,V,v\rangle \rightarrow \langle H,V,\text{Value (Bool true)}\rangle}{\langle H,V,\ \text{if } v \text{ then } \{e1\} \text{ else } \{e2\}\rangle \rightarrow \langle H,V,e1\rangle}$$

CoreJava Operational Semantics

<H,V,v> --> <H,V,Value (Bool false)>

<H,V, if v then {e1} else {e2}> --> <H,V,e2>

CoreJava Operational Semantics

$$\frac{\langle H,V,v\rangle \rightarrow \text{error}}{\langle H,V,\ \text{if } v \text{ then } \{e1\} \text{ else } \{e2\}\rangle \rightarrow \text{error}}$$

CoreJava Operational Semantics

$$\frac{\langle H, V, e1\rangle \rightarrow \langle H', V', e1'\rangle}{\langle H, V, e1 \ opint \ e2\rangle \rightarrow \langle H', V', e1' opint \ e2\rangle}$$

CoreJava Operational Semantics

<H,V, e1> --> Error

<H,V, e1 opint e2> --> Error

CoreJava Operational Semantics

is_value(e1) and (Tint == getType e1) and

<H,V,e2> -> <H',V',e2'>

<H,V, e1 opint e2> --> <H',V',e1 opint e2'>

CoreJava Operational Semantics

is_value(e1) and (Tint != getType e1)

<H,V, e1 opint e2> --> Error

CoreJava Operational Semantics

is_value(e1) and (Tint == getType e1) and
is_value(e2) and (Tint == getType e2)

<H,V, e1 opint e2> --> <H,V,Int (e1 opint e2)>

CoreJava Operational Semantics

is_value(e2) and (Tint != getType e2)

<H,V, e1 opint e2> --> Error

CoreJava Operational Semantics

$$\frac{\langle H, V, e1\rangle \rightarrow \langle H', V', e1'\rangle}{\langle H, V, e1 \ opfloat \ e2\rangle \rightarrow \langle H', V', e1' opfloat \ e2\rangle}$$

CoreJava Operational Semantics

<H,V, e1> --> Error

<H,V, e1 opfloat e2> --> Error

CoreJava Operational Semantics

is_value(e1) and (Tfloat == getType e1) and

<H,V,e2> -> <H',V',e2'>

<H,V, e1 opfloat e2> --> <H',V',e1 opfloat e2'>

CoreJava Operational Semantics

is_value(e1) and (Tfloat != getType e1)

<H,V, e1 opfloat e2> --> Error

CoreJava Operational Semantics

is_value(e2) and (Tfloat == getType e2)

<H,V, e1 opfloat e2> --> <H,V,Float (e1 opfloat e2)>

CoreJava Operational Semantics

is_value(e2) and (Tfloat != getType e2)

<H,V, e1 opfloat e2> --> Error

CoreJava Operational Semantics

$$\frac{\langle H, V, e1\rangle \rightarrow \langle H', V', e1'\rangle}{\langle H, V, e1 \;\&\&\; e2\rangle \rightarrow \langle H', V', e1' \;\&\&\; e2\rangle}$$

CoreJava Operational Semantics

$$\frac{\langle H, V, e_1 \rangle \rightarrow \text{Error}}{\langle H, V, e_1 \;\&\&\; e_2 \rangle \rightarrow \text{Error}}$$

CoreJava Operational Semantics

(e1 == Value Bool true)

<H,V, e1 && e2> --> <H,V,e2>

CoreJava Operational Semantics

$$\frac{(e1 == \text{Value Bool false})}{\langle H,V,\ e1\ \&\&\ e2\rangle \rightarrow \langle H,V,\ \text{Value Bool false}\rangle}$$

CoreJava Operational Semantics

$$\frac{\text{is\_value}(e1) \text{ and } (T_{bool} \neq \text{getType } e1)}{\langle H, V, e1 \,\&\&\, e2 \rangle \rightarrow \text{Error}}$$

CoreJava Operational Semantics

$$\frac{\langle H, V, e1\rangle \rightarrow \langle H', V', e1'\rangle}{\langle H, V, e1 \parallel e2\rangle \rightarrow \langle H', V', e1' \parallel e2\rangle}$$

CoreJava Operational Semantics

$$\frac{\langle H, V, e_1 \rangle \rightarrow \text{Error}}{\langle H, V, e_1 \,||\, e_2 \rangle \rightarrow \text{Error}}$$

CoreJava Operational Semantics

(e1 == Value Bool false)

<H,V, e1 || e2> --> <H,V,e2>

CoreJava Operational Semantics

$$\frac{(e1 == \text{Value Bool true})}{\langle H,V,\ e1 \mid\mid e2\rangle \dashrightarrow \langle H,V,\ \text{Value Bool true}\rangle}$$

CoreJava Operational Semantics

$$\frac{\text{is\_value}(e1) \text{ and } (T_{bool} \neq \text{getType } e1)}{\langle H, V, e1 \,||\, e2 \rangle \rightarrow \text{Error}}$$

CoreJava Operational Semantics

$$\frac{\langle H,V,e\rangle \rightarrow \langle H',V',e'\rangle}{\langle H,V,\ !e\rangle \rightarrow \langle H',V',e'\rangle}$$

CoreJava Operational Semantics

$$\frac{(\text{is\_value } e) \text{ and } (T_{bool} \neq \text{getType } e_1)}{\langle H, V, !e \rangle \rightarrow \text{Error}}$$

CoreJava Operational Semantics

<H,V, !(Value Bool v) > --> <H,V,Value Bool (not v) >

CoreJava Operational Semantics

$$\frac{\langle H,V,e1\rangle \rightarrow \langle H',V',e1'\rangle}{\langle H,V,\ e1\ \mathrm{opcmp}\ e2\rangle \rightarrow \langle H',V',e1'\ \mathrm{opcmp}\ e2\rangle}$$

CoreJava Operational Semantics

$$\frac{\langle H,V,e1\rangle \rightarrow \text{Error}}{\langle H,V,\ e1\ \text{opcmp}\ e2\rangle \rightarrow \text{Error}}$$

CoreJava Operational Semantics

$$\frac{\text{is\_value}(e1) \text{ and } (typ = \text{getType } e1) \text{ and } (typ == Tint \text{ or } typ = Tfloat) \text{ and } \langle H,V,e2\rangle \rightarrow \langle H',V',e2'\rangle}{\langle H,V, e1 \text{ opcmp } e2\rangle \rightarrow \langle H',V',e1 \text{ opcmp } e2'\rangle}$$

CoreJava Operational Semantics

is_value(e1) and (typ=getType e1)
and NOT (typ==Tint or typ=Tfloat) and

<H,V, e1 opcmp e2> --> Error

CoreJava Operational Semantics

is_value(e1) and (typ1=getType e1) and

is_value(e2) and (typ2=getType e1) and

(typ1==typ2) and (typ1==Tint or typ1=Tfloat)

<H,V, e1 opcmp e2> --> <H,V, Value Bool (e1 opcmp e2)>

CoreJava Operational Semantics

is_value(e1) and (typ1=getType e1) and
is_value(e2) and (typ2=getType e1) and
NOT (typ1==typ2)

<H,V, e1 opcmp e2> --> Error

CoreJava Operational Semantics

(v1..vn are defined in V) and (cn defined in Prg)

getFieldList cn = [(t1,f1);...;(tn,fn)] and

getValList V [v1;...;vn] = [val1;...;valn] and

getTypeList [val1;...;valn] = [tv1;...;tvn] and

(is_subtype tv1 t1) and ... and (is_subtype tvn tn)

and fEnv =[(f1,(t1,val1));...;(fn,(tn,valn))] and

(l is not a location in H) and H'= (l,(cn,fEnv))::H

<H,V, new cn(v1,...,vn)> --> <H',V, Loc l>

CoreJava Operational Semantics

(v1..vn are defined in V) and (cn defined in Prg)

getFieldList cn = [(t1,f1);...;(tn,fn)] and

getValList V [v1;..;vn] = [val1;...;valn] and

getTypeList [val1;...;valn] = [tv1;...;tvn] and

Exists i such that NOT (is_subtype tvi ti)

<H,V, new cn(v1,...,vn)> --> Error

CoreJava Operational Semantics

getFieldList cn = [(t1,f1);...;(tm,fm)] and

Not (m==n)

<H,V, new cn(v1,...,vn)> --> Error

CoreJava Operational Semantics

NOT (v1..vn are defined in V)

<H,V, new cn(v1,...,vn)> --> Error

CoreJava Operational Semantics

NOT (cn defined in Prg)

<H,V, new cn(v1,...,vn)> --> Error

CoreJava Operational Semantics

$$\frac{\text{Value Bool true} = \text{getVal c V}}{\text{<H,V, while c \{e\}> --> <H,V,e;while c \{e\}>}}$$

CoreJava Operational Semantics

$$\frac{\text{Value Bool false} = \text{getVal c V}}{\text{<H,V, while c \{e\}> --> <H,V,()>}}$$

CoreJava Operational Semantics

NOT (Value Bool false = getVal c) and

NOT (Value Value BBool true = getVal c)

<H,V, while c {e}> --> Error

(v0,v1,...,vn are defined in V) and

(loc = getVal v0 V) and (loc is a location in H) and

(cn =getClassName loc H) and

(tr mn(t1 a1,...,tn an) {e} is defined in class cn) and

(getValList [v1;...;vn] V =[val1;...;valn]) and

(getTypeList [val1;...;valn] =[t1';...;tn'])and

(is_subtype t1' t1) and ... and (is_subtype tn' tn) and

(generate fresh variables x0,x1,...xn) and

V'=(x0,(cn,loc))::(x1,(t1,val1))::...::(xn,(tn,valn))::V and

e1= (Subst this x0 (Subst a1 x1 (...(Subst an xn e)...)) and

e' = ret(x0, (ret x1, ... (ret xn e1)...)

<H,V, v0.mn(v1,...vn)> --> <H,V', e'>

(v0,v1,...,vn are defined in V) and

(loc = getVal v0 V) and (loc is a location in H) and

(cn =getClassName loc H) and

(tr mn(t1 a1,...,tn an) {e} is defined in class cn) and

(getValList [v1;...;vn] V =[val1;...;valn]) and

(getTypeList [val1;...;valn] =[t1';...;tn'])and

exists i such that NOT (is_subtype ti' ti)

<H,V, v0.mn(v1,...vn)> --> Error

(v0,v1,...,vn are defined in V) and

(loc = getVal v0 V) and (loc is a location in H) and

(cn =getClassName loc H) and

NOT (tr mn(t1 a1,...,tn an) {e} is defined in class cn)

<H,V, v0.mn(v1,...vn)> --> Error

(v0,v1,...,vn are defined in V) and

NOT ((loc = getVal v0 V) and (loc is a location in H))

<H,V, v0.mn(v1,...vn)> --> Error

NOT (v0,v1,...,vn are defined in V) and

<H,V, v0.mn(v1,...vn)> --> Error

CoreJava Operational Semantics

(cn is defined in the program) and (v is defined in V)
and (loc =getValue v V) and (loc is defined in H)

and (cn1 =getClassName loc H) and

and (is_subtype cn1 cn)

<H,V, (cn)v> --> <H,V,v>

CoreJava Operational Semantics

(cn is defined in the program) and (v is defined in V)
and (loc =getValue v V) and (loc is defined in H)

and (cn1 =getClassName loc H) and

and NOT (is_subtype cn1 cn)

<H,V, (cn)v> --> Error

CoreJava Operational Semantics

(cn is defined in the program) and (v is defined in V) and

NOT((loc =getValue v V) and (loc is defined in H))

<H,V, (cn)v> --> Error

CoreJava Operational Semantics

(cn is defined in the program) and

NOT (v is defined in V)

<H,V, (cn)v> --> Error

CoreJava Operational Semantics

NOT(cn is defined in the program)

<H,V, (cn)v> --> Error

CoreJava Operational Semantics

(cn is defined in the program) and (v is defined in V)
and (loc =getValue v V) and (loc is defined in H)

and (cn1 =getClassName loc H) and

and val= Value Bool (is_subtype cn1 cn)

<H,V, v instanceof cn> --> <H,V,val>

CoreJava Operational Semantics

(cn is defined in the program) and (v is defined in V) and

NOT((loc =getValue v V) and (loc is defined in H))

<H,V, v instanceof cn> --> Error

CoreJava Operational Semantics

(cn is defined in the program) and

NOT (v is defined in V)

<H,V, v instanceof cn> --> Error

CoreJava Operational Semantics

NOT (cn is defined in the program)

<H,V, v instanceof cn> --> Error

CoreJava Operational Semantics

(Prg = [clsD1;...;clsDn]) and

(clsDn=class Main { # void main() {e}})

<H,V, Prg> --> <H,V,e>

- the execution of a CoreJava program starts with the execution of a method main of a class Main.
- we assume that the class Main is the last declared class in the program and it contains only the method main and no other methods or other fields

CoreJava Operational Semantics

(Prg = [clsD1;...;clsDn]) and

NOT (clsDn=class Main { # void main() {e}})

<H,V, Prg> --> Error