Lecture 14 — MP8: Compiling MiniJava

● **Interpretive execution — as in MP6 and 7 — is rarely used in practice because it is inefficient. Instead, programs are translated to an executable form (machine language or bytecode) in one step (compilation), and then executed.**

● **We will compile MiniJava** *only after type-checking*, **because it is a little bit simpler, and more closely follows what a real Java compiler does.**

● **Topics**

  ● **An abstract machine for MP8**
  ● **Compilation rules for MP8**

Type-checking

- **Make sure programs are type-safe**

- **Insert type-conversion functions wherever type-checker says they are needed. E.g. transform** `"abc"+n` **to** `"abc"+converIntToString(n)`, **if** n **is an integer variable. After this, every expression has a single type, not determined at run time (with the exception of classes w/ subclasses).**

- **Calculate location of each field in an object and each variable on run-time stack, plus temporary locations for all expressions.**

Type-checking in MP8

- **Check types and add locations, constructing new AST of type `programT`:**

```
type programT = ProgramT of (class_declT list)

and class_declT = ClassT of id * id * ((var_kind * var_decl) list)
        * (method_declT list) * int (* number of fields *)

and method_declT = MethodT of exp_type * id * (var_decl list)
        * (var_decl list) * (statementT list) * annExpT * int (* size of stack frame *)

and statementT = BlockT of (statementT list)
    | IfT of annExpT * statementT * statementT
    | AssignVarT of id * annExpT * int
    | AssignFieldT of id * annExpT * int

and annExpT = expT * exp_type * int

and expT = OperationT of annExpT * binary_operation * annExpT | IntegerT of int
    | TrueT | FalseT | MethodCallT of annExpT * id * (annExpT list) | ThisT | NewIdT of id
    | VarT of id | FieldRef of int | NewIdAlloc of id * int | NotT of annExpT | NullT
    | StringT of string | CvtIntToStringT of annExpT | CvtBoolToStringT of annExpT
```

Example

- **Frame consists of four integer variables (x, y, z, and w (locations 1, 2, 3, 4); objects include field s, of type string, at offset 5. Write statement T for:**

```
z = x + 3
```

```
s = s + 2
```

Abstract machine

- **Abstract machine for MP8 has stack and heap, as usual. Stack frame contains integers, which may either be actual integers, boolean values (0 for false, 1 for true), or heap addresses; heap contains strings and objects.**

- **Sample instructions (locations are offsets in stack frame):**
  - **MOV loc1,loc2 — move value from loc2 in current frame to loc1**
  - **ADD loc1,loc2,loc3 — add value in loc2 and value in loc3, and put in location loc1**
  - **INT2STRING loc1,loc2 — value in loc2 is an int; convert it to a string, put that string in the heap, and store the address in loc1**
  - **INVOKE loc0,f,[loc1,...,locn] — loc0...locn are addresses in stack frame. loc0 contains heap address of an object, which must define, or inherit, a method f. Allocate a new stack frame (of correct size for f), fill locations 0, 1, ..., n with contents of loc0, loc1, ..., locn. Push it on stack, along with current pc. Jump to beginning of code for f.**

Example of abstract machine code

```
class Main {
    public boolean main (int n) {
        return this.isOdd(n);
    }

    public boolean isOdd (int n) {
        boolean b;
        if (n == 0)
            b = false;
        else
            b = this.isEven(n - 1);
        return b;
    }

    public boolean isEven (int n) {
        boolean b;
        if (n == 0)
            b = true;
        else
            b = this.isOdd(n - 1);
        return b;
    }
```

Example of abstract machine (cont.)

```
class Main
   main Main
   isOdd Main
   isEven Main
method main in Main (3)
0:    INVOKE      0,isOdd,1
      LOADRESULT 2
      RETURN      2

method isOdd in Main (6)
0:    LOADIMM     3,0
      EQUAL       4,1,3
      CJUMP       4,3,6
3:    LOADIMM     3,0
      MOV         2,3
      JUMP        11
6:    LOADIMM     3,1
      SUB         4,1,3
      INVOKE      0,isEven,4
      LOADRESULT 5
      MOV         2,5
11:   RETURN      2
```

Abstract machine instructions

- Machine has stack and heap and several special registers:

  - Code: machine code for the current method
  - PC: current address in machine code
  - Topofheap: allocation point for next heap item
  - Reg0: special register for returning value from method

- Stack is a stack of triples, (env,pc,code), where env is an array of integers giving the values of args, local variables, temporary values; pc and code are pc and code from *calling* function (to allow for return from this call).

- Heap is list of strings and objects. Each object is a pair containing a class name and a list of integers.

- *Note that values are not tagged.*

Abstract machine instructions (cont.)

- **Instructions:**

MOV(tgt,src)	**LOADIMM(tgt,i)**	**ADD(tgt,src1,src2)**
SUB(tgt,src1,src2)	**MULT(tgt,src1,src2)**	**DIV(tgt,src1,src2)**
LESS(tgt,src1,src2)	**AND(tgt,src1,src2)**	**OR(tgt,src1,src2)**
EQUAL(tgt,src1,src2)	**JUMP(iloc)**	**JUMPIND(src)**
CJUMP(loc, iloc_t, iloc_f)	**INT2STRING(tgt,src)**	**BOOL2STRING(tgt,src)**
NEWSTRING(tgt,strlit)	**CATSTRINGS(tgt,src1,src2)**	**ARRAYREF(tgt,arr,idx)**
GETFLD(tgt,srcfld)	**PUTFLD(tgtfld, loc)**	**NEWOBJECT(tgt,cls,sz)**
NEWARRAY(tgt,szsrc)	**RETURN(src)**	**LOADRESULT(tgt)**
INVOKE(rcvr,m,args)		

Specification of abstract machine

- **Specification is given in rules saying how each instruction changes state.**

- **State is six items of data: pc, machine code of currently-executing method, stack, heap, heaptop, and reg0. Write (p,c,s,h,t,r).**

- **Some rules (when "s" occurs on the right side of a rule, it refers to the environment of the top stack frame):**

MOV tgt,src	(p, c, s, h, t, r)	$\mapsto$	(p+1, c, s[s(src)/tgt], h, t, r)
LOADIMM tgt,i	(p, c, s, h, t, r)	$\mapsto$	(p+1, c, s[i/tgt], h, t, r)
ADD tgt,src1,src2	(p, c, s, h, t, r)	$\mapsto$	(p+1, c, s[s(src1)+s(src2)/tgt], h, t, r)
INT2STRING tgt,src	(p, c, s, h, t, r)	$\mapsto$	(p+1, c, s[int2str(s(src))/tgt], h, t, r)
CATSTRINGS tgt,src1,src2	(p, c, s, h, t, r)	$\mapsto$	(p+1, c, s[t/tgt], h[str1+str2/t], t+1, r)

where h(s(src1)) = str1 and h(s(src2)) = str2

NEWOBJECT tgt,C,i	(p, c, s, h, t, r)	$\mapsto$	(p+1, c, s[t/tgt], h[obj/t], t+1, r)

where obj = Obj(C,[0,0,...,0]) (i times)

PUTFLD i,src	(p, c, s, h, t, r)	$\mapsto$	(p+1, c, s, h[Obj(C,flds[s(src)/i])/s(0)], t, r)

where h(s(0)) = Obj(C,flds)

Abstract machine exercises

Suppose the code sequence C has the following instructions at locations 10–12:

```
LOADIMM  6,3
ADD      5,1,6
MOV      3,5
```

If the current frame has values: [3,7,2,4,9,21,13,15], give the state after each instruction:

$$(10, C, [3,7,2,4,9,21,13,15], h, t, r)$$

LOADIMM 6,3

ADD 5,1,6

MOV 3,5

Compilation

- **As usual, can compile by recursive traversal of AST.**

- **Will specify compilation with SOS-like rules. "Compilation judgments" have these forms:**

$$\textbf{Methods:} \quad M \rightsquigarrow il$$

$$\textbf{Statements:} \quad S, m \rightsquigarrow il, m'$$

$$\textbf{Expressions:} \quad e, loc \rightsquigarrow il$$

Compilation of methods

MethodT(typ,f,args,vars,sl,ret,sz) $\leadsto$

Compilation of expressions

IntegerT i, loc $\rightsquigarrow$

StringT s, loc $\rightsquigarrow$

TrueT, loc $\rightsquigarrow$

VarT id, loc $\rightsquigarrow$

NotT e, loc $\rightsquigarrow$

Compilation of expressions (cont.)

OperationT($e1$,Plus,$e2$), loc $\rightsquigarrow$

CvtIntToString e, loc $\rightsquigarrow$

NewIdAlloc(c,sz), loc $\rightsquigarrow$

Compilation of statements

$x=e$, m $\rightsquigarrow$

$\{ S_1, \ldots, S_n \}$, m $\rightsquigarrow$

if (e) S_1 else S_2, m $\rightsquigarrow$

Method calls

MethodCallT(e0,f,[e1,...,en])

MP8

- **Due next Thursday morning.**

- **Compilation rules given for same statements and expressions as in MP7.**

- **Execution should be the same for all type-correct programs, except that short-circuit evaluation is not implemented.**

Wrap-up

- **Today we discussed:**

  - **Compilation of MiniJava**
  - **Definition of an abstract machine**
  - **Compilation rules (in SOS style)**

- **We discussed it because:**

  - **This information will allow you to complete MP8.**

- **What to do now:**

  - **MP8**
  - *The write-up for MP8 is extremely complex, so we urge you to start early.*