

Lecture 14 — MP7: Compiling MiniJava

- **Translating MiniJava to machine language: MP7**
- **Type-checking**
- **The abstract machine for MP7**
- **Compilation rules**

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 MP7

- 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

- MP 7 has abstract machine language
- Machine has a stack and a heap, as usual.
- 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 contains heap address of an object, which must define, or inherit, a method f. loc1, ..., locn contain integers. 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.

Abstract machine instructions

- Machine has stack and heap and several special registers:
 - PC: current address in code stream
 - Code: code stream for the current method
 - Topofheap: allocation point for next heap item
 - Reg0: special register for returning value from method
- Heap is list of ints. Stack has structure: list 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.

Abstract machine instructions (cont.)

● Instructions:

MOV(tgt,src)

SUB(tgt,src1,src2)

LESS(tgt,src1,src2)

EQUAL(tgt,src1,src2)

CJUMP(loc, iloc_t, iloc_f)

NEWSTRING(tgt,strlit)

GETFLD(tgt,srcfld)

NEWARRAY(tgt,szsrc)

INVOKE(rcvr,m,args)

LOADIMM(tgt,i)

MULT(tgt,src1,src2)

AND(tgt,src1,src2)

JUMP(iloc)

INT2STRING(tgt,src)

CATSTRINGS(tgt,src1,src2)

PUTFLD(tgtfld, loc)

RETURN(src)

ADD(tgt,src1,src2)

DIV(tgt,src1,src2)

OR(tgt,src1,src2)

JUMPIND(src)

BOOL2STRING(tgt,src)

ARRAYREF(tgt,arr,idx)

NEWOBJECT(tgt,cls,sz)

LOADRESULT(tgt)

Specification of abstract machine

- Specification given in rules saying how each instruction changes state.
- State is six items of data: pc, code stream, stack, heap, heaptop, and reg0; write (p,c,s,h,t,r).
- Some rules:

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:

Methods: $M \rightsquigarrow il$

Statements: $S, m \rightsquigarrow il, m'$

Expressions: $e, loc \rightsquigarrow il$

Compilation of methods

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

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

$x=e, m \rightsquigarrow$

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

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

Method calls

$\text{MethodCallT}(e_0, f, [e_1, \dots, e_n])$

MP7

- Due next Thursday morning.
- Compilation rules given for same statements and expressions as in MP6.
- Execution should be the same for all programs, except that short-circuit evaluation is not implemented.
- Extra credit!
 - Compile some statements not included in MP6. Also implement short-circuit evaluation.
 - To be eligible for extra credit, must finish basic assignment by Tuesday morning.