
The Backend (continued)

David Notkin
Autumn 2008

MiniJava's IL (repeat)

- Want intermediate language to have simple, explicit operations
- Use simple declaration primitives
 - global functions, global variables
 - no classes, no implicit method lookup, no nesting
- Use simple data types
 - ints, doubles, explicit pointers, records, arrays
 - no class types, no implicit class fields
 - arrays are naked sequences; no implicit length or bounds checks
- Use explicit gotos instead of control structures
- Make all implicit checks explicit (e.g. array bounds checks)
- Implement method lookup via explicit data structures and code

CSE401 Au08

2

Data Type Representation (1)

- What IL type to use for each source type?
 - what operations are we going to need on them?
- `int`, `boolean`, `double`?

CSE401 Au08

3

Data Type Representations (2)

- What IL type to use for each source type?
 - what operations are we going to need on them?
- `class B` {


```
    int i;
    D j;
  }
```
- Instance of `Class B`?

CSE401 Au08

4

Inheritance

- How to lay out subclasses
 - Subclass inherits from superclass
 - Subclass can be assigned to a variable of superclass type implying subclass layout must "match" superclass layout
- `class B` {


```
    int i;
    D j;
  }
```
- `class C extends B` {


```
    int x;
    F y;
  }
```
- instance of `class C`

CSE401 Au08

5

Methods

- How to translate a method?
- Use a function
 - name is "mangled": name of class + name of method
 - make `this` an explicit argument
- Example
 - `class B` { ...


```
    int m(int i, double d) { ...
      body ... }
    }
```
- B's method `m` translates to


```
int B_m(*{...B...} this, int i, double d)
{ ... translation of body ... }
```

CSE401 Au08

6

Implementing Method Invocation

```

class B { ...
  int m(...) { ... }
  E n(...) { ... }
}
class C extends B { ...
  int m(...) { ... } // override
  F p(...) { ... }
}
B b1=new(B)
C c2=new(C)
B b2=c2
b1.m(...)
b1.n(...)
c2.m(...)
c2.n(...)
c2.p(...)
b2.m(...)
b2.n(...)

```

CSE401 Au08

7

Methods via Function Pointers in Instances

- Store code pointer for each new method in each instance
 - Reuse member for overriding methods
- Initialize with right method for that name for that object
- Do "instance var lookup" to get code pointer to invoke

```

class B { int i;
  int m(...) { ... }
  E n(...) { ... }
}
class C extends B { int j;
  int m(...) { ... } // override
  F p(...) { ... }
}

```

- Instance of class B:
 - * (int i, *code m, *code n)
- Instance of class C:
 - * (int i, *code m, *code n, int j *code p)

CSE401 Au08

8

Manipulating Method Function Ptrs

• Example

```

B b1 = new B();
C c2 = new C();
B b2 = c2;
b1.m(3, 4.5);
b2.m(3, 4.5);

```

• Translation:

```

*.. b1 = alloc {...B...}
b1->i = 0; b1->m = &B_m; b1->n = &B_n;
*.. c2 = alloc {...C...};
c2->i = 0; c2->m = &C_m; c2->n = &B_n;
c2->j = 0; c2->p = &C_p;
*.. b2 = c2
*(b1->m) (b1, 3, 4.5);
*(b2->m) (b2, 3, 4.5);

```

CSE401 Au08

9

Shared Method Function Pointers

- All direct instances of a class store the same method function pointer values
- So, can factor out common values into a single record shared by all -- often called a virtual function table, or vtbl
 - smaller objects, faster object creation
 - slower method invocations

- B's virtual function table (a global initialized variable):
 - {*code m, *code n} B_vtbl = {m=&B_m, n=&B_n};

• Example:

```

B b1 = new B();
b1.m(3, 4.5);

```

• Translation

```

*.. b1 = alloc(int i, {...B_vtbl...} vtbl);
b1->i=0; b1->vtbl = &B_vtbl;
*(b1->vtbl->m) (b1, 3, 4.5);

```

CSE401 Au08

10

Method Inheritance

- A subclass inherits all the methods of its superclasses: its method record includes all fields of its superclass
- Virtual function tables of subclass extends that of superclass with new methods, replacing overridden methods

```

class B { int i;
  int m(...) { ... }
  E n(...) { ... }
}
class C extends B { int j;
  int m(...) { ... } // override
  F p(...) { ... }
}

```

- {*code m, *code n} B_vtbl = {m=&B_m, n=&B_n};
- {*code m, *code n, *code p} C_vtbl = {m=&C_m, n=&B_n, p=&C_p};

CSE401 Au08

11

Example

• Example

```

B b1 = new B();
C c2 = new C();
B b2 = c2;
b1.m(3, 4.5);
b2.m(3, 4.5);

```

• Translation

```

*.. b1 = alloc (int i, {...B_vtbl...} vtbl);
b1->i = 0; b1->vtbl = &B_vtbl;
*.. c2 = alloc (int i, {...C_vtbl...} vtbl,int j);
c2->i = 0; c2->vtbl = &C_vtbl; c2->j = 0;
*.. b2 = c2
*(b1->vtbl->m) (b1, 3, 4.5);
*(b2->vtbl->m) (b2, 3, 4.5);

```

CSE401 Au08

12

Main ICG Operations

```
ILProgram Program.lower();
```

- translate the whole program into an ILProgram

```
void ClassDecl.lower(ILProgram);
```

- translate method decls
- declare the class's method record (vtbl)

```
void MethodDecl.lower(ILProgram, ClassSymbolTable);
```

- translate into IL fun decl, add to IL program

```
void Stmt.lower(ILFunDecl);
```

- translate into IL statement(s), add to IL fun decl

```
ILExpr Expr.evaluate(ILFunDecl);
```

- translate into IL expr, return it

```
ILType Type.lower();
```

```
ILType ResolvedType.lower();
```

- return corresponding IL type

CSE401 Au08

13

An Example ICG Operation

```
class IntLiteralExpr extends Expr {
    int value;
    ILExpr lower(ILFunDecl fun) {
        return new ILIntConstantExpr(value);
    }
}
```

CSE401 Au08

14

An Example ICG Operation

```
class AddExpr extends Expr {
    Expr arg1;
    Expr arg2;
    ILExpr lower(ILFunDecl fun) {
        ILExpr arg1_expr = arg1.lower(fun);
        ILExpr arg2_expr = arg2.lower(fun);
        return new ILIntAddExpr(arg1_expr,
                                arg2_expr);
    }
}
```

CSE401 Au08

15

Example Overloaded ICG Operation

```
class EqualExpr extends Expr {
    Expr arg1;
    Expr arg2;
    ILExpr lower(ILFunDecl fun) {
        ILExpr arg1_expr = arg1.lower(fun);
        ILExpr arg2_expr = arg2.lower(fun);
        if (arg1.getResultType().isIntType() &&
            arg2.getResultType().isIntType()) {
            return new ILIntEqualExpr(arg1_expr, arg2_expr);
        } else if (arg1.getResultType().isBoolType() &&
                    arg2.getResultType().isBoolType()) {
            return new ILBoolEqualExpr(arg1_expr, arg2_expr);
        } else {
            throw new InternalCompilerError(...);
        }
    }
}
```

CSE401 Au08

16

An Example ICG Operation

```
class VarDeclStmt extends Stmt {
    String name;
    Type type;
    void lower(ILFunDecl fun) {
        fun.declareLocal(type.lower(), name);
    }
}
```

declareLocal declares a new local variable in the IL function

CSE401 Au08

17

ICG of Variable References

```
class VarExpr extends Expr {
    String name;
    VarInterface var_iface; //set during typechecking
    ILExpr lower(ILFunDecl fun) {
        return var_iface.generateRead(fun);
    }
}

class AssignStmt extends Stmt {
    String lhs;
    Expr rhs;
    VarInterface lhs_iface; //set during typechecking
    void lower(ILFunDecl fun) {
        ILExpr rhs_expr = rhs.lower(fun);
        lhs_iface.generateAssignment(rhs_expr, fun);
    }
}
```

generateRead/generateAssignment gen IL code to read/assign the variable
code depends on the kind of variable (local vs. instance)

CSE401 Au08

18

ICG of Instance Variable References

```
class InstanceVarInterface extends VarInterface {
    ClassSymbolTable class_st;
    ILEExpr generateRead(ILFunDecl fun) {
        ILEExpr rcvr_expr =
            new ILVarExpr(fun.lookupVar("this"));
        ILType class_type =
            ILType.classILType(class_st);
        ILRecordMember var_member =
            class_type.getRecordMember(name);
        return new ILFieldAccessExpr(rcvr_expr,
            class_type,
            var_member);
    }
}
```

CSE401 Au08

19

ICG of Instance Variable Reference

```
void generateAssignment(ILEExpr rhs_expr,
    ILFunDecl fun) {
    ILEExpr rcvr_expr =
        new ILVarExpr(fun.lookupVar("this"));
    ILType class_type =
        ILType.classILType(class_st);
    ILRecordMember var_member =
        class_type.getRecordMember(name);
    ILEAssignableExpr lhs =
        new ILFieldAccessExpr(rcvr_expr,
            class_type,
            var_member);
    fun.addStmt(new ILAssignStmt(lhs, rhs_expr));
}
}
```

CSE401 Au08

20

ICG of if Statements

What IL code to generate for an if statement?
if (testExpr) thenStmt else elseStmt

CSE401 Au08

21

if

```
class IfStmt extends Stmt {
    Expr test;
    Stmt then_stmt;
    Stmt else_stmt;
    void lower(ILFunDecl fun) {
        ILEExpr test_expr = test.lower(fun);
        ILLabel false_label = fun.newLabel();
        fun.addStmt(
            new ILCondBranchFalseStmt(test_expr, false_label));
        then_stmt.lower(fun);
        ILLabel done_label = fun.newLabel();
        fun.addStmt(new ILGotoStmt(done_label));
        fun.addStmt(new ILLabelStmt(false_label));
        else_stmt.lower(fun);
        fun.addStmt(new ILLabelStmt(done_label));
    }
}
```

CSE401 Au08

22

ICG of Print Statements

- What IL code to generate for a print statement?
 - `System.out.println(expr);`
- No IL operations exist that do printing (or any kind of I/O)
 - oops

CSE401 Au08

23

Runtime Libraries

- Can provide some functionality of compiled program in
 - external runtime libraries
 - libraries written in any language, compiled separately
 - libraries can contain functions, data declarations
- Compiled code includes calls to functions & references to data declared libraries
- Final application links together compiled code and runtime libraries
- Often can implement functionality either through compiled code or through calls to library functions
 - tradeoffs?

CSE401 Au08

24

ICG of Print Statements

```
class PrintLnStmt extends Stmt {
    Expr arg;
    void lower(ILFunDecl fun) {
        ILEExpr arg_expr = arg.lower(fun);
        ILEExpr call_expr =
            new ILRuntimeCallExpr("println_int",
                arg_expr);
        fun.addStmt(new ILEExprStmt(call_expr));
    }
}
```

- What about printing doubles?

```
// print out an integer value
void println_int(int value) {
    printf("%d\n", value);
}
```

CSE401 Au08

25

ICG of new Expressions

- What IL code to generate for a new expression?

```
class C extends B {
    inst var decls
    method decls
}
... new C() ...
```

CSE401 Au08

26

ICG of new Expressions

```
class NewExpr extends Expr {
    String class_name;
    ILEExpr lower(ILFunDecl fun) {
        generate code to:
            allocate instance record
            initialize vtbl field with class's method record
            initialize inst vars to default values
            return reference to allocated record
    }
}
```

CSE401 Au08

27

An Example ICG Operation

```
class MethodCallExpr extends Expr {
    String class_name;
    ILEExpr lower(ILFunDecl fun) {
        generate code to:
            evaluate receiver and arg exprs
            test whether receiver is null
            load vtbl member of receiver
            load called method member of vtbl
            call fun ptr, passing receiver and args
            return call expr
    }
}
```

CSE401 Au08

28

"allocate instance record"

```
// allocate bytes of memory, and return a pointer to it
void* allocate(int bytes) {
    return malloc(bytes);
}

// allocate base_bytes + num_elems*elems_bytes bytes of cleared
// memory, and return a pointer to it
void* allocate_sequence(int base_bytes, int num_elems, int elems_bytes)
{
    return calloc(base_bytes + num_elems * elems_bytes, 1);
}

// report an exception and die
void throw_exception(char* message) {
    fprintf(stderr, "Unhandled exception: %s\n", message);
    exit(-1);
}
```

CSE401 Au08

29

Run-time systems

- So, what are run-time systems really
 - With respect to compilers?
 - Operating systems?

CSE401 Au08

30

Preview

- Target code generation
- Optimization