Prof. Jingke Li (FAB120-06, lij@pdx.edu), Tue & Thu 12:00-13:15 @ ASRC 230, Lab: Fri 10:00-11:15/11:30-12:45 @ FAB 170

Lab 2: IR Code Generation

Learning Objectives Upon successful completion, students will be able to:

- Use syntax-directed translation scheme to implement an IR code generator from a simple AST representing common expressions and statements.

Preparation

Download and unzip the file `lab2.zip`. You'll see a `lab2` directory with the following contents:

`ast/Ast0.java, ast/<other>.java.java` — the source AST representation and its parser code

`ir/IR0.java` — the target IR representation

`IR0Gen0.java` — a starter version of the IR code-generator

`IR0Interp.jar` — an interpreter for the IR language

`tst/` — a set of tests

`Makefile` — for compiling your program

`gen, run` — scripts for testing programs

IR Code-Gen Implementation

The IR code-gen implementation follows the syntax-directed translation scheme. For an input AST, it traverses the tree top-down starting with the root node `Ast0.Program`. At each node, it generate IR code using local information and the recursive results from its children.

In CS321 Homework 4 of last term, we used this approach for implementing two static analysis. This time, the program organization is a little different. Instead of inserting new methods to the AST node classes, we have all the code-gen routines placed in a separate program file, `IR0Gen.java`, and have the program traverse the AST explicitly. The advantege of this organization is that there is a clear separation between the definition and the use of the AST.

The `main` method in `IR0Gen.java` reads in an AST program through an AST parser. It then invokes the `gen` routine on the top-level `Ast0.Program` node. The rest of the program is a collection of (overloaded) `gen` routines, one for each type of AST nodes. Each individual `gen` routine implements the attribute grammars developed for IR code-gen. (*Hint:* You may want to review this week's lecture notes.)

- For an `Ast0.Stmt` node, the `gen` routine returns a list of IR0 instructions (`List<IR0.Inst>`). This list corresponds to the `Stmt.c` attribute discussed in class.
- For an `Ast0.Exp` node, the `gen` routine returns a list of IR0 instructions and an `IR0.Src` object (for holding the `Exp`'s value), represented together by a `CodePack` object. These two items correspond to the two attributes, `Exp.c` and `Exp.v`, discussed in class.

Your task is to complete the `gen` routine implementation for all the AST0 nodes. After finishing coding, you can compile and test your `IR0Gen` program by using the following commands:

```
linux> make gen
linux> ./gen tst/test*.ast
linux> ./run tst/test*.ir
```

Grammars of AST0 and IR0

The grammars of AST0 and IR0 are included below for your reference. You need to get familiar with the two corresponding programs, `ast/Ast0.java` and `ir/IR0.java`.

"AST0 Grammar"

```
Program -> {Stmt}
Stmt    -> "{" {Stmt} "}"
        |  "Assign" Exp Exp
        |  "If" Exp Stmt ["Else" Stmt]
        |  "While" Exp Stmt
        |  "(" "NewArray" <IntLit> ")"
        |  "(" "ArrayElm" Exp Exp ")"
        |  "Print" Exp
Exp     -> "(" "Binop" BOP Exp Exp ")"
        |  "(" "Unop" UOP Exp ")"
        |  <Id>
        |  <IntLit>
        |  <BoolLit>
BOP     -> "+" |"-" | "*" | "/" | "&&" | "||"
        |  "==" | "!=" | "<" | "<=" | ">" | ">="
UOP     -> "-" | "!"

<Id>      = [A-Za-z][A-Za-z0-9]*
<IntLit>  = [0-9]+
<BoolLit> = true|false
```

"IR0 Grammar"

```
Program -> {Inst}
Inst    -> ( Dest "=" Src BOP Src                // Binop
           | Dest "=" UOP Src                    // Unop
           | Dest "=" Src                        // Move
           | "print" "(" [Src] ")"               // Print
           | "if" Src ROP Src "goto" <Label>     // CJump
           | "goto" <Label>                      // Jump
           | <Label> ":"                         // LabelDec
           ) <EOL>
Src     -> <Id> | <Temp> | <IntLit> | <BoolLit>
Dest    -> <Id> | <Temp>
Addr    -> [<IntLit>] "[" Src "]"
BOP     -> AOP | ROP
AOP     -> "+" | "-" | "*" | "/" | "&&" | "||"
ROP     -> "==" | "!=" | "<" | "<=" | ">" | >="
UOP     -> "-" | "!"

<Label>   = [A-Za-z][A-Za-z0-9]*
<Id>      = [A-Za-z][A-Za-z0-9]*
<Temp>    = t[0-9]+
<IntLit>  = [0-9]+
<BoolLit> = true|false
```