CS322 Languages and Compiler Design II, Winter 2016 1/22/16

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 3: IR Code Optimization

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

- Implement the constant-folding optimization in an IR code generator.

Preparation

Download and unzip the file `lab3.zip`. You'll see a `lab3` directory with the following contents:

`ast/Ast0.java, ast/<other>.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

`geno, run` — scripts for testing programs

This set of programs are mostly the same as those in `lab2.zip`, except for the new `geno` script and the new `.ir.opt` files in `tst`.

Constant Folding

When seeing an AST expression `"(Binop + 2 4)"`, the baseline code generator faithfully generates an IR0 instruction `"t1 = 2 + 4"`.

Constant folding is to have the code generator evaluate constant expressions to their values, and/or to use the constant information to simplify the IR code. For the above example, the code generator would generate the instruction `"t1 = 6"`, instead.

Constant folding appear mostly in `Binop` and `Unop` expressions, and can be used to simplify `If` and `While` statements. Here are some general tips.

- Constant folding should be performed bottom up on an AST expression tree, so that cases such as the following can be recognized:

```
(Binop + (Binop * 2 3) (Binop - 4 1)) => (Binop + 6 3) => 9
```

- Constant folding can be applied to all types of constants and operations. Here are some examples:

```
(Binop < 1 2)              => true
(Binop == (Binop + 1 2) 3) => (Binop == 3 3) => true
(Binop || true false)      => true
```

- Constant folding can simplify Boolean expressions that contain non-constant components:

```
(Binop || true x)              => true
(Binop && false (Binop + x y)) => false
(Binop || x true)              => true
(Binop || x false)             => x
```

The last two examples show that even if a constant appears as the second operand, the code generator can use the information to simply the IR code.

- Constant folding can also simplify If and While statements:

```
If true Print 1                 => Print 1
If false Print 2 Else Print 3 => Print 3
If (Unop ! true) Print 4      => (empty!)
While false Print 1           => (empty!)
```

Exercise

Use your `IR0Gen.java` of Lab 2 (or the provided `IR0Gen0.java`) as a starter version, implement an IR generator that performs constant folding. Call your new program `IR0GenOpt.java`.

You should start with the simple cases, *i.e.* arithmetic expressions, then move on to logic expressions, and finally if you have time, work on the `If` and `While` statements.