Syntax Directed Translation

66.648 Compiler Design Lecture (03/16//98)

Computer Science

Rensselaer Polytechnic

Lecture Outline

- Syntax Directed Translation
- Java Virtual Machine
- Examples
- Administration

Phases of a Compiler

1. Lexical Analyzer (Scanner)

Takes source Program and Converts into tokens

2. Syntax Analyzer (Parser)

Takes tokens and constructs a parse tree.

3. Semantic Analyzer

Takes a parse tree and constructs an abstract syntax tree with attributes.

Phases of a Compiler- Contd

4. Syntax Directed Translation

Takes an abstract syntax tree and produces an Interpreter code (Translation output)

5. Intermediate-code Generator

Takes an abstract syntax tree and produces un-optimized Intermediate code.

Syntax Directed Translation Scheme

A syntax directed translation scheme is a syntax directed definition in which the net effect of semantic actions is to print out a translation of the input to a desired output form.

This is accomplished by including "emit" statements in semantic actions that write out text fragments of the output, as well as string-valued attributes that compute text fragments to be fed into emit statements.

Examples

1. Postfix and Prefix notations:

We have already seen how to generate them.

Let us generate Java Byte code.

E -> E'+' E { emit("iadd");}

E-> E '* ' E { emit("imul");}

E-> T

T -> ICONST { emit("sipush ICONST.string);}

T-> '(' E ')'

Abstract Stack Machine

We now present (Read Java Virtual Machine Spec) a simple stack machine and illustrate how to generate code for it via syntax-directed translations.

The abstract machine code for an expression simulates a stack evaluation of the postfix representation for the expression. Expression evaluation proceeds by processing the postfix representation from left to right.

Evaluation

1. Pushing each operand onto the stack when encountered.
2. Evaluating a k-ary operator by using the value located k-1 positions below the top of the stack as the leftmost operand, and so on, till the value on the top of the stack is used as the rightmost operand.
3. After the evaluation, all k operands are popped from the stack, and the result is pushed onto the stack (or there could be a side-effect)

Example

Stmt -> ID '=' expr { stmt.t = expr.t || 'istore a'}

applied to a = 3*b -c

bipush 3

iload b

imul

iload c

isub

istore a

Java Virtual Machine

Analogous to the abstract stack machine, the Java Virtual machine is an abstract processor architecture that defines the behavior of Java Bytecode programs.

The stack (in JVM) is referred to as the operand stack or value stack. Operands are fetched from the stack and the result is pushed back on to the stack.

Advantages: VM code is compact as the operands need not be explicitly named.

Data Types

The int data type ca nold 32 bit signed integers in the range -2^31 to 2^(31) -1.

The long data type can hold 64 bit signed integers.

Integer instructions in the Java VM are also used to operate on Boolean values.

Other data types that Java VM supports are byte, short, float, double. (Your project should handle at least three data types).

Selected Java VM Instructions

Java VM instructions are typed I.e., the operator explicitly specifies what operand types it expects.

Expression Evaluation

sipush n	push a 2 byte signed int on to stack
iload v	load/push a local variable v
istore v	store top of stack onto local var v
iadd	pop tow elements and push their sum
isub	pop two elements and push their difference

Selected Java VM Instructions

Imul pop two elements and push their product

iand pop two lements and push their bitwise and

ior

ineg pop top element and push its negation

lcmp pop two elements (64 bit integers), push the comparison result. 1 if Vs[0]<vs[1], 0 if vs[0]=vs[1] otherwise -1.

I2L convert integers to long

L2i

Selected Java VM Instructions

Branches:

GOTO L unconditional transfer to label l

ifeq L transfer to label L if top of stack is 0

ifne transfer to label L if top of stack !=0

Call/Return: Each method/procedure has memory space allocated to hold local variables (vars register), an operand stack (optop register) and an execution environment (frame register)

Selected Java VM Instructions

Invokestatic p invoke method p. pop args from stack as initial values of formal parameters (actual parameters are pushed before calling).

Return return from current procedure

ireturn return from current procedure with integer value on top of stack.

Areturn return from current procedure with object reference return value on top of stack.

Selected Java VM Instructions

Array Manipulation: Java VM has an object data type reference to arrays and objects

newarray int create a new arrae of integers using the top of the stack as the size. Pop the stack and push a reference to the newly created array.

Iaload pop array subscript expression on top of stack and array pointer (next stack element). Push value contained in this array element.

iastore

Selected Java VM Instructions

Object Manipulation

new c create a new instance of class C (using heap) and push the reference onto stack.

Getfield f push value from object field f of object pointed by object reference at the top of stack.

Putfield f store value from vs[1] into field f of object pointed by the object reference vs[0]

Selected Java VM Instructions

Simplifying Instructions:

ldc constant is a macro which will generate either bipush or sipush depending on c.

For the project, we will use the java assembler of Jason Hunt (washington university).

Advantages:

No need to worry about binary class files. They get generated automatically.Named local variables. Labels instead of byte offsets.

Byte Code (JVM Instructions)

No-arg operand: (instructions needing no arguments hence take only one byte.)

examples: aaload, aastore,aconsta_null, aload_0, aload_1, areturn, arraylength, astore_0, athrow, baload, iaload, imul etc

One-arg operand: bipush, sipush,ldc etc

methodref op:

invokestatic, invokenonvirtual, invokevirtual

Byte Code (JVM Instructions)

Fieldref_arg_op:

getfield, getstaic, putfield, pustatic.

Class_arg_op:

checkcast, instanceof, new

labelarg_op (instructions that use labels)

goto, ifeq, ifne, jsr, jsr_w etc

Localvar_arg_op:

iload, fload, aload, istore

Translating an if statement

Stmt -> if expr then stmt1 { out = newlabel();

stmt.t = expr.t|| 'ifnnonnull' || out || stmt1.t ||

'label' out: 'nop' }

example:

if (a +90==7) { x = x+1; x = x+3;}

Translating a while statement

Stmt -> WHILE (expr) stmt1 { in =newlabel(); out= neewlabel();

stmt.t = 'label' || in|| 'nop' || expr.t || 'ifnonnull'|| out|| stmt1.t || 'goto' || in|| 'label' || out }

Comments and Feedback

Project 3 is out. Please start working. PLEASE do not wait for the due date to come.

We are looking at the relevant portion of Java. Please keep studying this material.