Generation of Intermediate Code

66.648 Compiler Design Lecture (03/19//97)

Computer Science

Rensselaer Polytechnic

Lecture Outline

- Intermediate Code
- Examples
- Administration

Intermediate Languages

Different Language front ends ===> Intermediate Code

Intermediate Code ==> Machine codes for different architectures.

Ideally an intermediate language is a language independent intermediate representation of the program being compiled.

Advantages of using intermediate language:

1. Retargeting -build a compiler for a new machine quickly.

Intermediate Languages

2. Optimization - reuse intermediate code optimizers in compilers for different languages and different machines.

One such intermediate representation is byte code (post fix notation)

Some Other Representations:

1. Abstract Syntax tree

2. Three Address Statements

Abstract Syntax Tree

The AST is a condensed/simplified/abstract form of the parse tree in which:

1. Operators are associated with interior nodes rather than leaf nodes.
2. Chain of single productions are collased.

Example: a+78 * 6

Syntax-Directed Translation into AST

The AST is usually constructed via syntax directed translation with tree operations performed in the semantic actions.

T -> num { T.n = mkleaf(num.val);}

T -> T1 * num { tnode =mkleaf(num.val); T.n =mknode('*',T1.n,tnode); }

E-> T E.n=T.n;}

E-> E1+T { E.n=mknode('+'E1.n,T.n);}

Good News-Bad News

Advantages:

1. Easy to attach a code generator for a new target machine.
2. Useful for other tools such as syntax directed editor.

Disadvantages:

1. Specific to a single programming language.
2. Hard to perform optimizing transforms

Three Address Statements

An intermediate representation consisting of a sequence of statements of the form:

operator, operands(two) and a result.

1. Assignment statement:

x = y op z /* Binary operator */

x = op z /* Unary operator */

x= a+b-c*d

t1 = c*d

t2 = a+b; t3 = t2 -t1; x= t3

Indexed Assignment

Indexed assignments of the form x = y[I] and y[I]=x, where y[I] represents the memory location at address Y+I. Normally, Y is the base address of the array.

A[j+I] = b[k]

t1 = b[k]

t2 = j+I /* some times t2 = t2*4 statement has to be added */

a[t2]=t1

Label Statement

Label L marks the following three address statement as the target of a jump instruction.

Conditional and Unconditional Jump statements are similar to the bytecode. Goto L transfers control to the statement L.

if X relop Y goto L

These are not Typed.

Call/return Statements

Return x /* Returns from a procedure with an optional return value x */

param x1

param x2

..

Call P,n /* calls a procedure P(x1,x2,..Xn)

Address/Pointer Assignements

X = &y content of x is the address of location y

x = *y content of x is the content of content of loc y

*x = y contents of location x = y

For generation of three address codes (3AC), we need two attributes place and code.

1. .place - the location that holds the value

2. . Code - A sequence of 3 AC

Syntax Directed Translation

T --> id { T.place =id.place; T.code="";}

T--> T1 * id { T.place =newtemp(); T.code=T1.code|| id.code ||emit("T.place = T2.place * id.place}

E--> T { E.code = T.code; E.place = T.place;}

E--> E1+T { E.place=newtemp(); E.code = E1.code||T.code||emit(E.place=E1.place + T.place;}

Local Information for Variables

3 AC untermediate representations use a symbol table data structure to store the following information about variables.

1. Name - A symbolic name identifying the variable
2. Tag - Additional information of the variable such as type.
3. Location - A memory offset relative to the base address for all variables in the procedure (the location may be replaced by a register after optimization.)

Examples

3 AC Statements

Translation of Boolean Expression

For Boolean Expressions of the form (I<j) and b or a, what sort of 3AC has to be generated.

There are two methods:

1. Evaluate all terms and combine via logical operators to get the expression.
2. Short Circuit Code - Evaluate terms conditionally when evaluating boolean expression.

In conditional statements (for correctness reasons)

Boolean Expression

One prefers short circuit evaluation.

E.g., searching for an element in an array of elements (of given size).

Language Specification usually specifies what evaluation it uses.

Short Circuit code can be faster if one of the terms forces one to decide quickly about the outcome.

In pipelined architectures, this may result in slow codes.

Expression Evaluation

Idea: Use 1 to denote true and 0 to denote false, evaluate logical (Boolean) expressions similar to arithmetic expressions.

E.g. a and b or not c

t1 = a and b

t2 = not c

t3 = t1 or t2.

Relational Expression

A relational expression such as (a < b) may appear as a logical expression. It may be evaluated as an expression such as

t1 = a < b

Otherwise, it can be translated into:

if a < b goto L1

t1 = 0

goto L2

label L1:

Relational Operation

T1 = 1

label L2:

..

Comments and Feedback

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

We are in chapter 8. Please reda that chapter and read the relevant portion of Java. Please keep studying this material and work exercises.