Class 04: Exploring Compilation

Held: Friday, 2 September 2011

Summary: We explore compilation with an example, some discussion, and some laboratory exercises.

Related Pages:

- EBoard.
- Lab: Exploring Compilation.
- Reading: Aho et al., Chapter 2.

Notes:

- I am surprised to have received no questions on the Pascal assignment.
- EC for next Thursday’s CS Extra: TK on Alice.
- We may not have time to do the lab in class today. But you can finish it on your own.
- Due next Wednesday: Lab writeup: Exploring Compilation.

Overview:

- An example.
- Describing and translating languages.
- Chapter 2.
- Lab: Exploring Compilation.

A Compilation Example

Let’s explore the stages of compilation with a simple example (although one a little bit more uniform than the ones in the book).

Source Code

You should find this algorithm familiar.

```pascal
var
  x, z: real;
  i: integer;
begin
  write("x: ");
  readln(x);
  write("i: ");
  readln(i);
  z := i;
  while (i > 0) do
    begin
```

```pascal
```
if (odd(i)) then
    z := z*x;
    i := i div 2;
    x := sqr(x);
end;
println(z);

Tokenizing (Lexical Analysis)

VAR ID(x) COMMA ID(z) COLON REAL SEMI ID(i)
COLON INTEGER SEMI BEGIN WRITE LPAREN STR(x:) RPAREN
SEMI READLN LPAREN ID(x) RPAREN SEMI WRITE LPAREN
STR(i:) RPAREN READLN LPAREN ID(i) RPAREN SEMI
ID(z) ASSIGN INT(1) SEMI WHILE LPAREN ID(i) GT
INT(0) RPAREN DO BEGIN IF LPAREN ID(odd) LPAREN ....

Note: We might use ID(integer) and ID(real) instead of INTEGER and REAL. We might also used ID(write) and ID(readln) rather than WRITE and READLN.

Parsing (Syntactic Analysis)

...

StatementList
   \ StatementList
    \ Statement StatementList
     \ Assignment Statement StatementList
      \ ID(z) INT(1) WhileLoop Statement NIL
       \ GT Compound Output
        \ ID(i) INT(0) StatementList
         \ Statement
          \ Conditional

An AST

Assignment -> Output -> Input -> Output -> Input -> WhileLoop -> Output -> NIL
   \ ID(z) INT(1) STR(x:) ID(x) STR(i) ID(i) GT \ ID(z)
    \ \ ID(i) INT(0) Conditional
       -> Assignment
          -> Assignment
             -> NIL
Type Checking (Semantic Analysis)

Assignment -> WhileLoop -> Output -> NIL
real            real
/    \
ID(z) INT(1) GT \       ID(z)
real int int \    real
/    \
ID(i) INT(0) Conditional -> Assignment -> Assignment -> NIL
int   int

Intermediate Code Generation

.Data   S0      "x:"
.Data   S1      "i:"
SWrite  S0
Fread   X       # readln(X)
SWrite  S1
Iread   I
I2F     V1      $0    # v1 = real(0)
Fassign Z       V1    # z = v1
Label   Loop1
Icmp    V2      I      $0    # v2 = i > 0
Jmpzero V2      End1   # if (! v2) goto End1
Label   Body1
Ipush   I
Icall   V3      Odd    # v3 = Odd(i)
Jmpzero V3      Alt2   # if (! v3) goto Alt2
Label   Cons2
Fmul    V4      Z      X    # v4 = z * x
Fassign Z       V4    # z = v4
Label   Alt2
Idiv    V5      I      2    # v5 = i div 2
Iassign I       V5    # i = v5
...
Goto    Loop1
Label   End1
Fpush   Z
Call    P_write

Code Optimization Improvement

- Assignment propagation
  - From
    - I2R     V1      $0    # v1 = real(0)
    - Fassign Z       V1    # z = v1
  - To
    - I2R     Z       $0    # Z = real(0)

- Convert constant types
  - From
I2R Z $0  # Z = real(0)

○ To

FASSIGN Z $0.0  # Z = real(0)

● Call inlining
● Loop unrolling
● Etc.

Convert to Assembly

● From

IPUSH I
ICALL V3 ODD  # v3 = ODD(i)

● To

movl 8(%ebp), 0(%esp)  # Assume that i is 8 from the base
subl $4, %esp
call odd
movl -4(%esp), 20(%ebp)  # Assume that v3 is 20 from the base

Code Optimization Improvement

● Liveness analysis and register allocation

movl %eax, 0(%esp)  # i goes in register a
subl $4, %esp
call odd
movl -4(%esp), %ebx  # v3 goes in register b

● Loop unrolling
● Etc.

Describing Languages

● So, how do we describe languages to do all this?
● And how do we use those descriptions to implement languages?
● We use one language to describe the tokens.
  ○ Lexical analysis is often straightforward.
● We use BNF to describe the syntax.
  ○ We often rewrite our BNF grammars to deal with things that are hard to deal with computationally, such as ambiguity and left recursion.
● There are standard ways to turn certain BNF grammars into parsers.
● We add semantic actions to add meaning
  ○ Type checking
  ○ Code generation
There are standard ways to turn annotated BNF grammars into translators.

Note: As this suggests, some of the steps above are theoretical rather than actual; for example, we don’t really build the parse tree or AST.

Chapter 2

- What did you see as the other highlights of chapter 2?
- What questions do you have on chapter 2?