

Some Thoughts on Grad School

Goal

- learn how to learn a subject in depth
- learn how to organize a project, execute it, and write about it

Iterate through the following:

- read the background material
- try some examples
- ask lots of questions
- repeat

You will have too much to do!

- learn to prioritize
- it is not possible to read ALL of the background material
- spend 2+ hours of dedicated time EACH day on each class/project
- what grade you get is not the point
- have fun and learn a ton!

Undergraduate Compilers Review and Intro to MJC

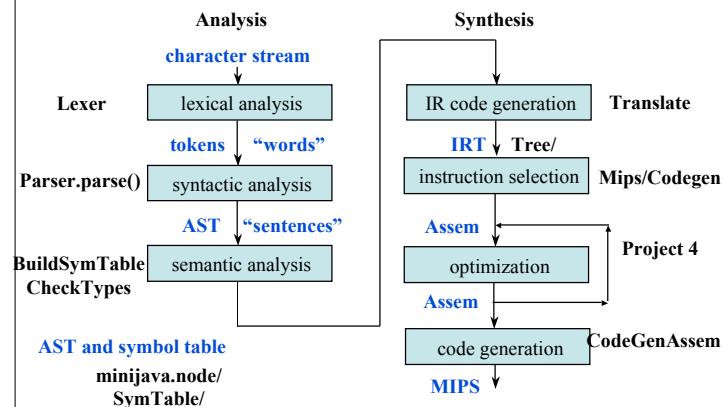
Announcements

- Mailing list is in full swing, go ahead and share test cases

Today

- Semantic analysis
- Visitor pattern for abstract syntax trees
- IRT Trees
- Assem

Structure of the MiniJava Compiler (CodeGenAssem.java)



Lexing and Parsing

Lexing

- theoretical tool: regular expressions
- recognizing substrings instead of strings so need longest match and rule priority
- implementation tools: flex, lex, SableCC, etc. generate code that implements a deterministic finite automata that recognizes the specified tokens

Parsing

- theoretical tool: context free grammars
- recognizing a whole program of tokens
- implementation tools: bison, yacc, SableCC, etc. generate a LALR(1) or bottom-up parser that uses shift-reduce parsing to recognize the program and uses syntax-directed translation to generate an AST

Compiler Data Structures

Symbol Tables

- Compile-time data structure
- Holds names, type information, and *scope* information for variables

Scopes

- A name space
- e.g., In Pascal, each procedure creates a new scope
- e.g., In C, each set of curly braces defines a new scope
- Can create a separate symbol table for each scope
- What are the scopes in MiniJava?

Using Symbol Tables

- For each variable declaration:
 - Check for symbol table entry
 - Add new entry; add type info
- For each variable use:
 - Check symbol table entry

CS553 Lecture

Undergraduate Compilers Review

10

Using the Visitor Pattern for semantic analysis

```

public class DepthFirstAdapter extends AnalysisAdapter {
    ...
    public void inAPlusExp(APlusExp node) {
        defaultIn(node);
    }
    public void outAPlusExp(APlusExp node) {
        defaultOut(node);
    }
    public void caseAPlusExp(APlusExp node) {
        inAPlusExp(node);
        if(node.getLExp() != null) {
            node.getLExp().apply(this);
        }
        if(node.getRExp() != null) {
            node.getRExp().apply(this);
        }
        outAPlusExp(node);
    }
}
    
```

```

public final class APlusExp extends PExp {
    ...
    public void apply(Switch sw) {
        ((Analysis) sw).caseAPlusExp(this);
    }
    ...
}
    
```

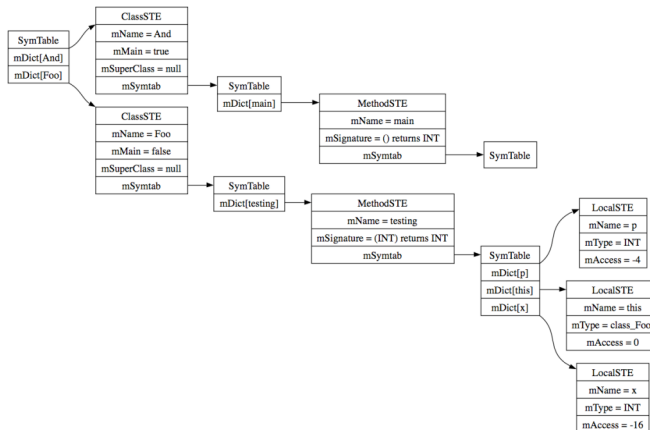
The BuildSymTable is an example visitor that uses this visitor pattern.

CS553 Lecture

Undergraduate Compilers Review

11

Symbol Table in the MiniJava Compiler

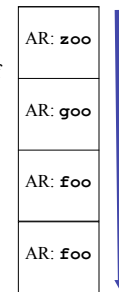


Compiling Procedures

Properties of procedures

- Procedures/methods/functions define scopes
- Procedure lifetimes are nested
- Can store information related to **dynamic invocation** of a procedure on a call stack (*activation record* or AR or stack frame):
 - Space for saving registers
 - Space for passing parameters and returning values
 - Space for local variables
 - Return address of calling instruction

higher addresses



Stack management

- Push an AR on procedure entry (caller or callee)
- Pop an AR on procedure exit (caller or callee)
- Why do we need a stack?

lower addresses

stack

CS553 Lecture

Undergraduate Compilers Review

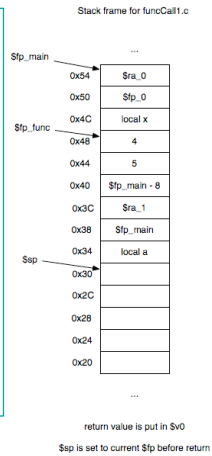
13

Stack Frame for MiniJava Compiler

```
int foo(int x,int y,int *z) {
    int a;
    a = x * y - *z;
    return a;
}
void main() {
    int x;
    x = 2;
    cout << foo(4,5,&x);
    cout << "\n";
}
```

```
.text
_foo:
    sw    $ra, 0($sp)    #PUSH
    subu $sp, $sp, 4
    sw    $fp, 0($sp)    #PUSH
    subu $sp, $sp, 4
    addu $fp, $sp, 20
    subu $sp, $sp, 24
    ...
    lw    $t0, -20($fp)
    move $v0, $t0
    lw    $ra, -12($fp)
    move $t0, $fp
    lw    $fp, -16($fp)
    move $sp, $t0
    jr    $ra
```

```
.text
.globl main
main:
    sw    $ra, 0($sp)    #PUSH
    subu $sp, $sp, 4
    sw    $fp, 0($sp)    #PUSH
    subu $sp, $sp, 4
    addu $fp, $sp, 8
    subu $sp, $sp, 12
    li    $t0, 2
    sw    $t0, -8($fp)
    li    $t0, 4
    sw    $t0, 0($sp)    #PUSH
    subu $sp, $sp, 4
    li    $t0, 5
    sw    $t0, 0($sp)    #PUSH
    subu $sp, $sp, 4
    subu $t0, $fp, 8
    sw    $t0, 0($sp)    #PUSH
    subu $sp, $sp, 4
    jal   _foo
    move  $a0, $v0
    ...
    lw    $ra, 0($fp)
    move $t0, $fp
    lw    $fp, -4($fp)
    move $sp, $t0
    jr    $ra
```



Undergraduate Compilers Review

Wisconsin C-- calling convention

Calling convention (contract between caller and callee)

- \$sp must be divisible by 4
- caller should pass parameters in order on the stack
- upon callee entry, the stack pointer \$sp should be pointing at the first empty slot past the last parameter
- upon callee exit, the stack pointer \$sp should be pointing at the first parameter
- upon callee exit, return value should be in \$v0

Rules to follow for PA6 (to standardize frame usage)

- \$sp should always be pointing at next empty slot on the stack
- \$ra and \$fp should be stored right after the parameters on stack, you can't use any other callee-saved registers
- \$fp should be made to point at the first parameter, so that the address for the first parameter is \$fp-0, the address for the second parameter is \$fp-4, ...
- locals should be stored in order, right after \$ra and \$fp

CS553 Lecture

Undergraduate Compilers Review

15

Compiling Procedures (cont)

Code generation for procedures

- Emit code to manage the stack
- Are we done?

Translate procedure body

- References to local variables must be translated to refer to the current activation record
- References to non-local variables must be translated to refer to the appropriate activation record or global data space

CS553 Lecture

Undergraduate Compilers Review

16

Code Generation

Conceptually easy

- IRT Tree is a generic machine language, 3-address code is another example of an intermediate representation
- Instruction selection converts the low-level IR to real machine instructions

The source of heroic effort on modern architectures

- Alias analysis
- Instruction scheduling for ILP
- Register allocation
- More later. . .

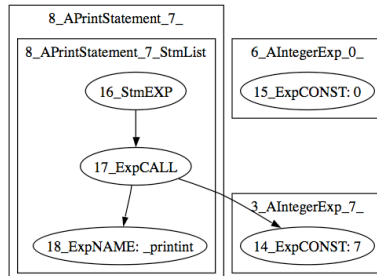
CS553 Lecture

Undergraduate Compilers Review

17

PrintSeven testing method (translating to IR Tree)

```
public int testing() {
    System.out.println(7);
    return 0;
}
```



CS553 Lecture

18

MIPS instruction selection in MiniJava compiler

Assem data structure

- has string with source and destination spots to represent assembly instruction
- has list of uses, defs, and jump targets

add rd, rs, rt
"add `d0, `s0, `s1"

beq rs, rt, label
"beq `s0, `s0, `j0"

lw rt, address
"lw `d0, #(`s0)"

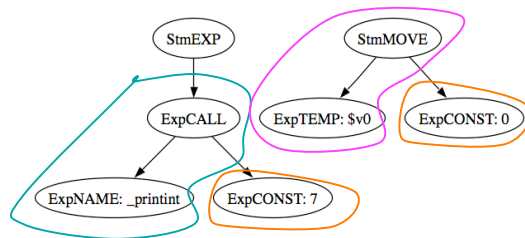
sw rt, address
"sw `s0, #(`s1)"

CS553 Lecture

Undergraduate Compilers Review

19

PrintSeven testing, instruction selection



```
# ExpCALL
# ExpCONST
li t36, 7
# push parameter onto stack
sw t36, 0($sp)
subu $sp, $sp, 4
jal _printint
```

```
# ExpCONST
li t37, 0
#StmMOVE(ExpTEMP(t1), e)
move $v0, t37
```

CS553 Lecture

Undergraduate Compilers Review

20

SpillAll

Before spill

```
# ExpCALL
# ExpCONST
li t36, 7
# push parameter onto stack
sw t36, 0($sp)
subu $sp, $sp, 4
jal _printint
```

```
# ExpCONST
li t37, 0
#StmMOVE(ExpTEMP(t1), e)
move $v0, t37
```

After spill

```
# ExpCALL
# ExpCONST
li $t0, 7
sw $t0, -12($fp)
lw $t0, -12($fp)
# push parameter onto stack
sw $t0, 0($sp)
subu $sp, $sp, 4
jal _printint
```

```
# ExpCONST
li $t0, 0
sw $t0, -16($fp)
lw $t0, -16($fp)
#StmMOVE(ExpTEMP(t1), e)
move $v0, $t0
```

s Review

21

Prologue and Epilogue

Prologue

```
.text
Foo_testing:
Foo_testing_frameSize=20
Foo_testing_paramsNregsaves=12
    sw $ra, 0($sp)
    subu $sp, $sp, 4
    sw $fp, 0($sp)
    subu $sp, $sp, 4
    addu $fp, $sp, Foo_testing_paramsNregsaves
    subu $sp, $fp, Foo_testing_frameSize
```

... # spilled instructions for body

Epilogue

```
# epilogue
done2:
    lw $ra, -4($fp)
    move $t0, $fp
    lw $fp, -8($fp)
    move $fp, $sp
    jr $ra
```

Concepts

Compilation stages

- Scanning, parsing, semantic analysis, intermediate code generation, optimization, code generation

Parsing

- generating an AST
- shift-reduce parsing

Semantic Analysis

- symbol tables
- using visitors over the AST

Intermediate Representations

- IRT Tree
- Assem

Next Time

Suggested Exercises

- from book: 2.2.1, 2.2.2, 2.3.1
- follow a while loop in MiniJava through to code gen
 - what does AST look like?
 - what does IRT Tree look like?
 - what is the MIPSnoereg code?
- how would we implement a do while loop?

Lecture

- Compiling OOP

Parsing Terms (Definitely know these terms)

Lexical Analysis

- longest match and rule priority
- regular expressions
- tokens

CFG (Context-free Grammar)

- production rule
- terminal
- non-terminal

Syntax-directed translation

- inherited attributes
- synthesized attributes