CSE443
Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall
Final Exam

5/17/2019, Friday
8:00AM - 11:00AM

Clemens 106
(this room!)
HW4 rubric snafu

Max grade is 20 points.
% in UBLearns should be correct now.
Project Due Date

Monday
5/13/2019
5:00 PM
Lexical analysis (flex, lex) → Token stream

Syntactic analysis (bison) → Parse tree

Semantic analysis → Attribute grammar (semantic rules)

Regular expressions → NFA → DFA → Min DFA

Chomsky hierarchy

Context-free language = regular language

Bison

Builds state machines corresponding to grammars

Output

Syntactic analyzer (c program)

Finite control

Action/Goto table

Symbol Table

Contains all names/symbols and their types: variables, constants, functions, etc.
**Semantic Analysis**
- Parse tree
- IR code generation
- IR code
- Machine-Independent optimization
- IR code
- Code generation
- Assembly code

**Attribute Grammar**
- Semantic rules
- Semantic predicates

**Action**
- GOTO

**Finite Control**

**Symbol Table**
- Contains all names/symbols used in program info about those symbols, such as kind of value it represents: type, variable, ...
- For a variable: its type, its scope ... how is this done?
- "one table per scope"

**Assign Node**
- `Var := expr`
- `type = {`attribute sets`}`
- `expr op expr {type = {`attribute sets`}`
- `expr {type = {`attribute sets`}`
- Keep track of types

**To look up a name, start here**
- and search through parent scopes to global scope
Character Stream → LEXER → Token Stream → PARSER → Syntax Tree → Sem. Ant

IR

+ 

(overloaded)

enum IR-op

if false x GOTO L2
if x GOTO L1
GOTO L1

int foo(int x, real y) { ... }
FRONT END

PARSER → syntax tree → SEM. ANALYSIS → syntax tree w/ annotations → IR CODE GEN

---

3-address code instructions

IR instructions

dst src1 op src2

1. \( X = Y \) \( \text{op} \) \( Z \)
2. \( X = \text{op} \ Y \)
3. \( X = Y \)
4. \( \text{Goto} \)
5. if \( X \) \( \text{Goto} \) / if \( \text{False} \) \( X \) \( \text{Goto} \)
6. if \( X \) \( \text{rel} \) \( Y \) \( \text{Goto} \)
7. param \( X \) / call \( f, n \)

invocation
8. \( X = a[i] \) \( a[i] = Y \)
9. \( X = \& Y \) \( X = \& Y \) \( \& X = Y \)

---

makeList(3) → [3]
mergelists ([0,5], [3]) → [0,3,5]
back patch ([0,3,5], [17])
gen creates an IR instruction
call [Q]

IR code arr

1. if x goto 17
2.
3. goto [17]
4.
5. goto -17
6.
7.
8.
9.
10.
11.
12.
13.
14.
15.
16.
17.
18.
...
Struct IR {
  struct IR * instruction = ...;
  struct STK * dst, src1, src2;
  enum IR_op op;
  int jump-target;
  int instruction-type;
}
Continue review

- Assembly code generation
- Instruction selection
- Register/address descriptors
- Optimizations
- (Machine independent)
register descriptor

"For each available register, a register descriptor keeps track of the variables names whose current value is in that register." [p. 543]
address descriptor

"For each program variable, an address descriptor keeps track of the location or locations where the current value of that variable can be found." [p. 543]
Updating register descriptors (RD) and address descriptors (AD)

1. LD R, x
   (a) Set RD of R to only x
   (b) Add R to AD of x
2. ST x, R
   (a) Add &x to AD of x
3. OP Rx, Ry, Rz for x = y op z
   (a) Set RD of Rx to only x
   (b) Set AD of x to only Rx (&x not in AD of x!)
   (c) Remove Rx from the AD of any variable other than x
4. "When we process a copy statement x = y, after generating the load for y into register Ry, if needed, and after managing descriptors as for all load statement (per rule 1):"  [p. 545]
   (a) Add x to the RD of Ry
   (b) Set AD of x to only Ry
Algorithm 8.7 [p. 528]
Determining the liveness and next-use information for each statement in a basic block.

INPUT: A basic block $B$ of three address instructions. Assume the symbol table initially shows all non-temporary variables in $B$ as being live on exit.

OUTPUT: At each statement $i$: $x = y + z$ in $B$, we attach to $i$ the liveness and next-use information for $x$, $y$, and $z$.

METHOD: We start at the last statement in $B$ and scan backwards to the beginning of $B$. At each statement $i$: $x = y + z$ in $B$ do the following:

1) attach to statement $i$ the information currently found in the symbol table regarding the next-use and liveness of $x$, $y$, and $z$.
2) In the symbol table, set $x$ to "not live" and "no next use".
3) In the symbol table, set $y$ and $z$ to "live" and the next uses of $y$ and $z$ to instruction $i$.
Exam format

- Expect about 4 short essay questions (choose from about 6). We will use BlueBooks.
- You will have 3 hours to write the exam.
- I would expect you to take about 30 minutes per question, leaving about an hour to check over your work.
Possible Exam Questions

- Anything from homeworks,

...and...
Type checking
(Semantic processing)

- Explain how type errors are detected. Discuss how type information is gathered, stored and checked. Pick a concrete syntactic construct that can contain a type error, and explain how type checking detects the error.
Intermediate Code Generation

Explain how short-circuit Boolean expressions are translated into intermediate code. Discuss how jump targets can be determined. Illustrate by showing how a concrete Boolean expression involving at least two Boolean operators is translated into intermediate code.
Register Allocation and Assignment

Describe the getReg(I) algorithm, answering the questions of what data structures it uses, when and how these structures are updated. What is meant by "spill", when does it occur, and why is it needed? Demonstrate with a concrete example.
Symbol Table Usage

Describe the structure and use of a symbol table. Explain which phases of the compiler use the table, including what data is written to or read from the table during each phase. Give a concrete code example and the corresponding ST.
Invocation Records

- Describe a typical layout for an invocation record, detailing what information is stored in the record. Explain how variable length parameters and variable length local data can be accommodated. Discuss the location and use of the stack and top pointers. Give concrete example.
Function Calls

- Explain how a function call takes place. Include in your discussion mention of the roles of the caller and callee in setting up the invocation record, and how machine state is remembered at the call and restored at return. Explain how recursive calls are handled (do NOT discuss tail-call optimization). Give concrete example.
Optimizations

- Pick an optimization and explain the benefit(s) of having the compiler apply it to code, and sketch how it works for a concrete example.

- Ex:
  - tail-call optimization
  - code motion
  - dead code elimination
Final exam questions?

Project questions?
Thanks for a great semester!
You're coming to the May 18th graduation ceremony, right?

Congrats to everyone graduating!!
Check out my cool hat!

There will be people in funny outfits!
Have a wonderful summer!
(and see you at the final)