CSE443 Compilers

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

5/15/2024, Wednesday
Park 440 (this room)

Start @ 8:00 AM
End @ 11:00 AM

Arrive by 7:50 AM
Entry not guaranteed after 8:30 AM

Bring your UB card (or other government-issued photo ID)
Exam format

- Expect 4 short essay questions (choose from ~6). We will use BlueBooks.

- I expect you to take about 30 minutes per question (about 2 hours total).

- This leaves you with about 1 hour to proofread/edit your responses.
Sample Exam Questions
(not a comprehensive or exhaustive list)

- Type checking
- Intermediate Code Generation
- Register Allocation and Assignment
- Symbol Table Usage
- Invocation Records
- Function Calls
- Optimizations
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 during backpatching. 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: discuss both calling and return sequences, and the division of labor between caller and callee. Explain how machine state is remembered at the call and restored at return. Cover 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
2022 course overview
Character stream → LEXICAL ANALYSIS (LEX, FLEX) → Token stream → SYNTACTIC ANALYSIS (BISON) → Parse tree → SEMANTIC ANALYSIS

Lexical structure (keywords, identifiers, etc.) described by a regular grammar

Regular expressions → FLEX → lex analyzer (C program) → lexical analyzer (C program)

Chomsky hierarchy

Type 0 → Type 1 → Type 2 → Type 3

Context-free grammar → BISON

SYMBOL TABLE

Contains all names/symbols, stores info about the symbols, the kind of value it represents, etc., e.g., for a variable: its type, its scope, etc.

 Builds state machine corresponding to grammar

Output

SYNTACTIC ANALYZER (C program)
**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"

**Keep track of types**
- Global scope
- To look up a name, start here and search through parent scopes to global scope
Character stream → LEXER → token stream → PARSER → syntax tree

3-address code instructions

1. \( X = Y \) op \( Z \)
2. \( x = \text{op} \ y \)
3. \( x = y \)
4. \( \text{GOTO} \ L2 \)
5. if false \( \text{GOTO} \ L2 \)
6. if \( x \ \text{rel} \ y \) \( \text{GOTO} \ L1 \)
7. \( \text{param} \ x / \text{call} \ f, x \)
8. \( x = a[i] \) \( a[i] = y \)
9. \( x = \& y \) \( x = xy \) \( xk = y \)

int foo(int x, real y) {}...3

enum 1R-op

\( + \quad i + i = i \quad i / i \quad i < \quad r < \)
赦 r - r = r x r /

HLL

(overloaded)

iffalse x GOTO L2 if x GOTO L1
GOTO L1

RO

param X

RI

param Y
call f, 2
Front End

Parser → Syntax Tree → Semantic Analysis → Syntax Tree w/ Annotations → IR Code Generation

3-address code instructions

IR instructions

dst src1 op src2

1. X = Y op Z
2. X = op y
3. X = y
4. GOTO __
5. if x GOTO 1 / ifFalse x GOTO __
6. if x relop y GOTO __
7. param x / call f, n

Invocation

Record

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

nextInstr: 18

if x goto 17

if x goto 17

goto 17

...
gen:

struct IR * instruction = ...

A[nextInstr++] = instruction;

struct IR {
    struct STE * dst, src1, src2;
    enum IR_op op;
    int jump-target;
    int instruction-type;
};
Course Evaluation

Please complete, as your feedback is very meaningful and drives improvements to the course!

Let me know what worked and what didn’t work well for you (and why).

If something didn’t work well for you please share what might have made things better for you.
Thanks for a great semester!

Have a wonderful summer!

Congrats to everyone graduating!!