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

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 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 (FLEX) → token stream (keywords, identifiers, etc) → SYNTACTIC ANALYSIS (BISON) → parse tree → SEMANTIC ANALYSIS

Lexical structure described by a regular grammar

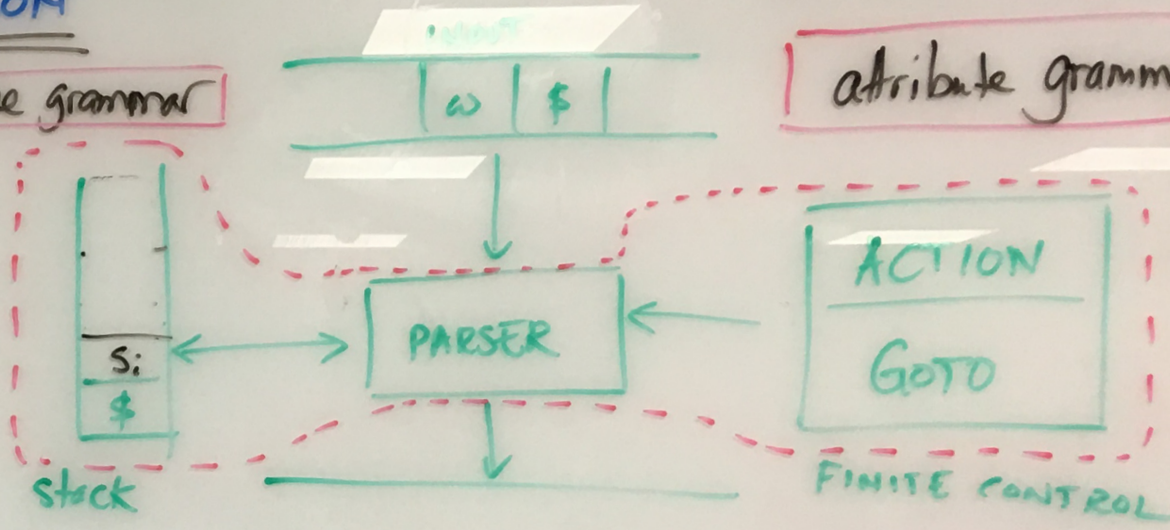
regular expressions → FLEX → lexical analyzer (C program)

i.e. → NFA → DFA → min DFA ...

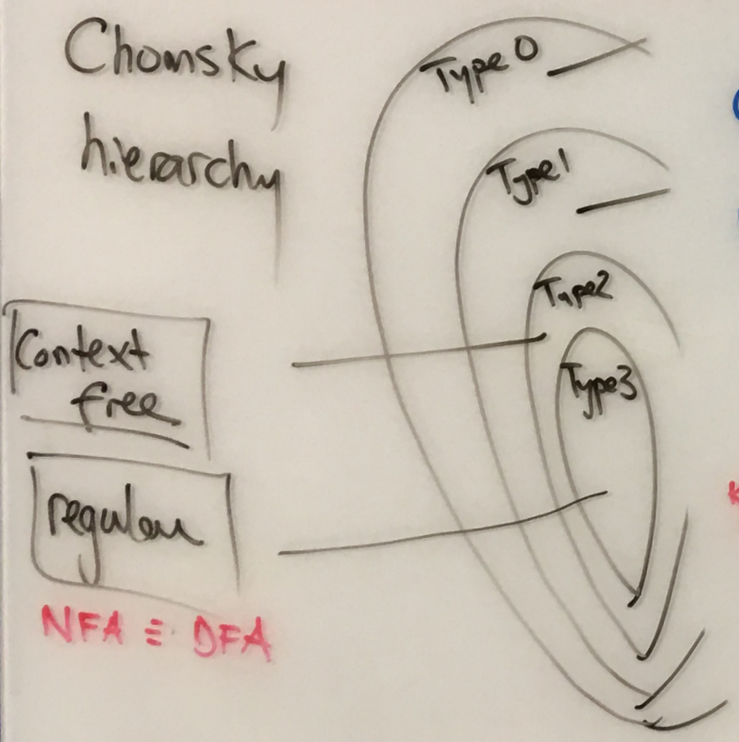
Context-free grammar

attribute grammar

Semantic ru
Semantic pr



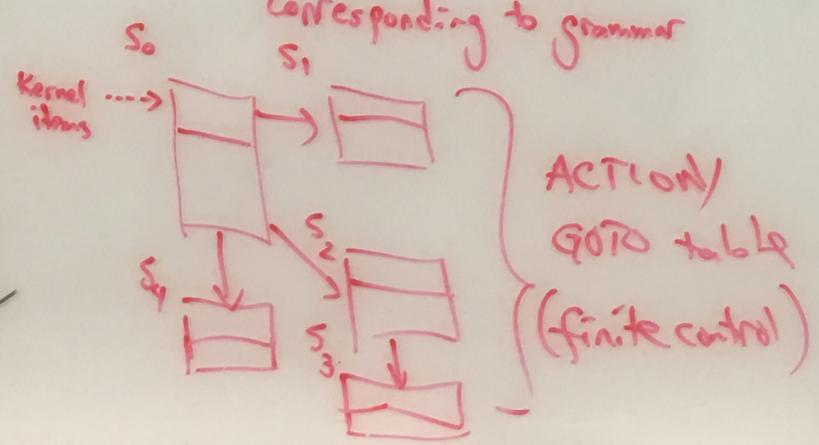
Chomsky hierarchy



Context-free grammar → BISON → Syntactic analyzer (C program)

OUTPUT

Builds state machine corresponding to grammar



SYMBOL TABLE

Contains all names/symbols
Stores info about those symbo
the kind of value it represents: t

e.g. for a variable: its type
its scope
" one

SEMANTIC ANALYSIS → parse tree → IR code generation → IR code → Machine-Indep optimization → IR code → Code gen. → Assembly code

Reg. Descr.
Addr. Descr.

attribute grammar

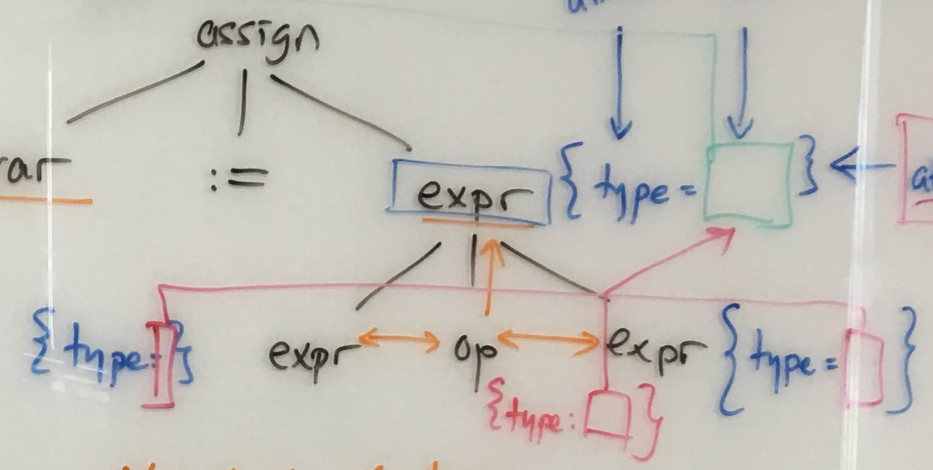
Semantic rules
Semantic predicates

semantic predicate: var.type == expr.type

ACTION
GOTO

FINITE CONTROL

attribute value — a type checking predicate

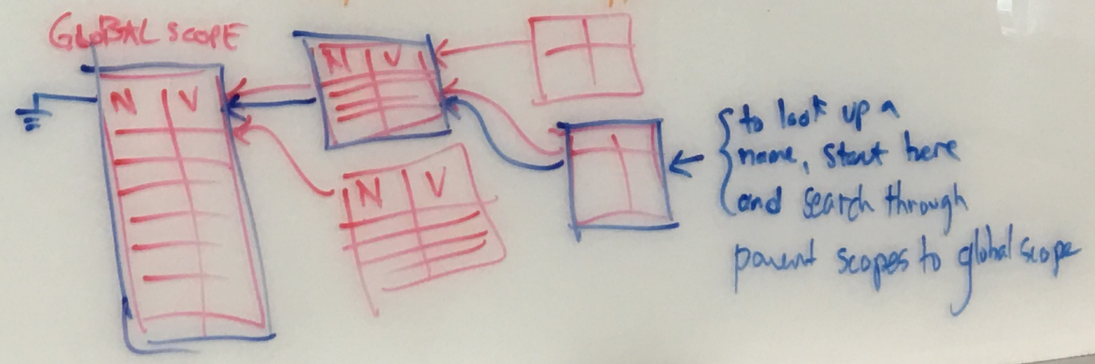


SYMBOL TABLE

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



character stream → LEXER → token stream → PARSER → syntax tree →

FRONT END

IR

HLL

\+ (overloaded) → i+ i- i* i/ i% i< / r+ r- r* r/ r< c< b<

enum IR-op

3-address code instructions

IR instructions

	dst	src1	op	src2
1.	x	y	op	z

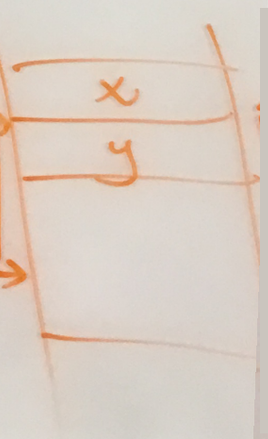
2. x = op y
3. x = y
4. GOTO ___
5. if x GOTO ___ / ifFalse x GOTO ___
6. if x relop y GOTO ___ (relop: < =)
7. param x / call f, n
8. x = a[i] a[i] = y
9. x = &y x = *y *x = y

ifFalse x GOTO L2
GOTO L1

if x GOTO L1
GOTO L2

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

R0 param x
R1 param y
call f, 2
→ assembly



invocation record

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	op	z

2. x = op y
3. x = y
4. GOTO _
5. if x GOTO _ / ifFalse x GOTO _
6. if x relop y GOTO _ (relop: <, =)
7. param x / call f, n
8. x = a[i] a[i] = y
9. x = &y x = *y *x = y

invocation record

makeList(3) → [3]

mergeLists([0,5],[3]) → [0,3,5]

backpatch([0,3,5], 17)

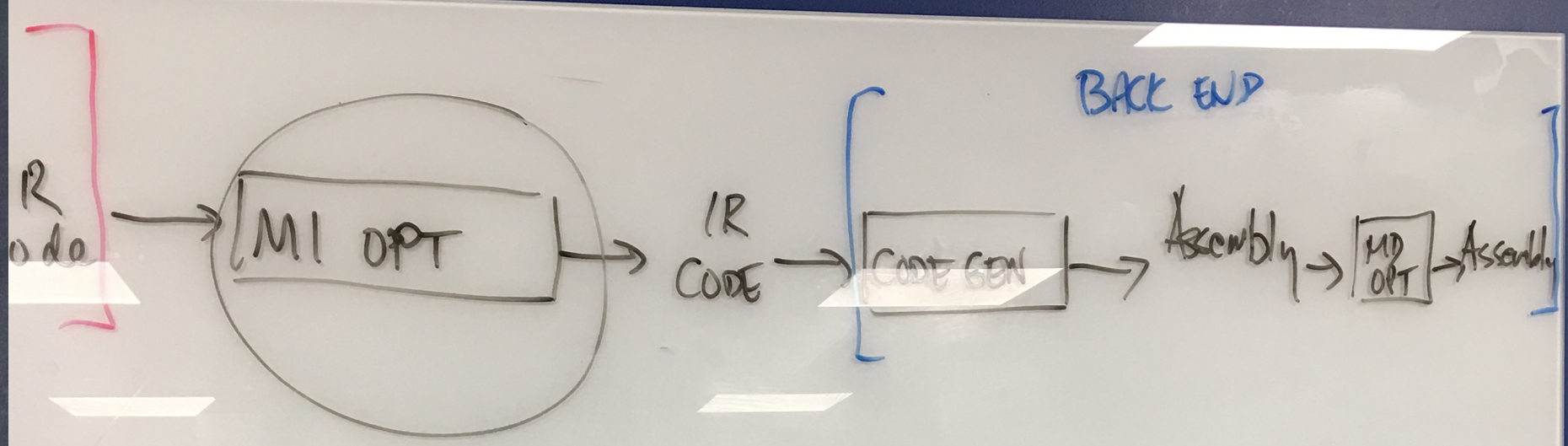
gen creates an IR instruction

arr

IR code arr

0	if x goto 17
1	✓
2	✓
3	goto 17
4	
5	goto 17
6	
⋮	⋮
17	....
18	

nextInstr: 18



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!

Congrats to everyone graduating!!

Have a wonderful summer!