

# Exam 1 Review

---

EECS 483 – Lecture 15  
University of Michigan  
Monday, October 30, 2006

# Logistics

---

- ❖ When, Where:

- » Wednesday, Nov 1, 10:40am – 12:30pm
- » Room TBA

- ❖ Type:

- » Open book/note

- ❖ What to bring:

- » Text book, reference books, lecture notes
  - But must print these out
- » Pencils
- » No laptops or cell phones

# Topics Covered

---

- ❖ Lexical analysis: ~25%
- ❖ Syntax analysis: ~50%
- ❖ Semantic analysis: ~25%
  - » Percentages are VERY approximate
- ❖ Not covered
  - » MIRV specific stuff
  - » Detailed flex/bison syntax
    - However, project 1 and 2 material is fair game
    - And general questions about flex/bison are possible

# Textbook

---

- ❖ What have we covered: Ch 1 - 6
- ❖ Things you should know / can ignore
  - » Ch 1, 2 – Just overviews, don't worry about it
  - » Ch 3 – Lexical analysis
  - » Ch 4 – Syntax analysis
  - » Ch 5.1-5.6 – Syntax-directed translation
    - No 5.7-5.10
    - Detailed algorithms in 5.5, 5.6 not important
  - » Ch 6.1-6.2, 7.6
    - Rest of 6/7 you are not responsible for

# Open Book Exams

---

❖ OPEN BOOK != DON'T STUDY

❖ Open book means

- » Memorizing things is not that important
- » If you forget something, you can look it up

❖ But

- » If you are trying to learn (or re-learn) stuff during the test, you won't finish
- » Assume the test is not open book
  - Don't rely on book/notes
  - Treat them as a backup

# How to Study

---

- ❖ Re-familiarize yourself with all the material
  - » Where to find things if you get confused
- ❖ Practice solving problems
  - » Do them w/o looking at the answer!
  - » Class problems/examples from lectures
  - » Fall 2004 exam 1
  - » Examples in book
    - If you are ambitious, exercises at the end of each chapter
  - » Practice so that you can do them without thinking much

# Exam Format

---

- ❖ Short answer: ~40%
  - » Explain something
  - » Short problems to work out
- ❖ Longer design problems: ~60%
  - » E.g., construct a parse table
- ❖ Range of questions
  - » Simple – Were you conscience in class?
  - » Grind it out – Can you solve problems
  - » Challenging – How well do you really understand things
- ❖ My tests tend to be long – so move on if you get stuck!

# Lexical Analysis (aka Scanning)

---

- ❖ Ch 3
- ❖ Regular expressions
  - » How to write an RE from a textual description
- ❖ NFAs
  - » RE to NFA (Thompson construction)
- ❖ DFAs
  - » NFA to DFA
- ❖ State minimization
- ❖ How does lex/flex work

# Regular Expressions

---

Construct a regular expression over the alphabet  $\{a,b,c\}$  that are at least 3 characters in length and end with an 'a'.

# NFAs

---

Construct an NFA for the following regular expression:

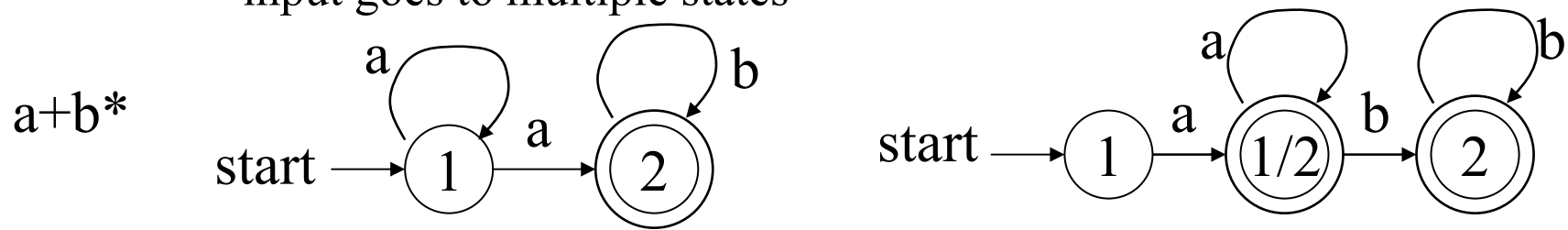
$$a^*(b|a)^*c^+$$

# Recall: Convert the NFA to a DFA

---

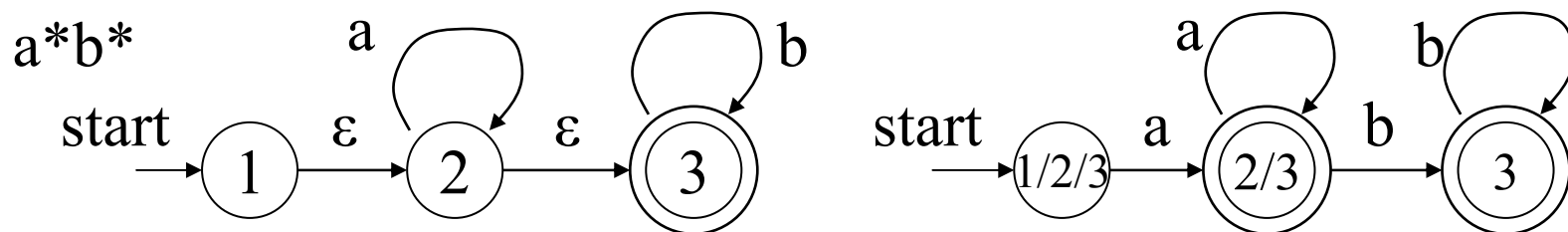
## ❖ Problem 1: Multiple transitions

- » Move  $(S,a)$  is relabeled to target a new state whenever single input goes to multiple states



## ❖ Problem 2: $\epsilon$ transitions

- » Any state reachable by an  $\epsilon$  transition is “part of the state”
- »  $\epsilon$ -closure - Any state reachable from  $S$  by  $\epsilon$  transitions is in the  $\epsilon$ -closure; treat  $\epsilon$ -closure as 1 big state



# NFA to DFA Conversion

---

Convert the previous NFA into a DFA

# Syntax Analysis (aka Parsing)

---

- ❖ Ch. 4
- ❖ Context free grammars
  - » Derivations, ambiguity, associativity
- ❖ Parsing
  - » Top-down
    - LL(1), building parse tables (FIRST, FOLLOW)
  - » Bottom-up
    - LR(0), LR(1), SLR, LALR
    - Building parse tables (Closure, Goto), shift/reduce
- ❖ Abstract syntax tree

# Context Free Grammars

---

A context free grammar is capable of representing more languages than a regular expression. What is the major reason for this?

# Context Free Grammars

---

Write a grammar to parse all strings consisting of the symbols  $\{a,b,c,d,e\}$  of the form:

$$a^n b^{2m} c^m d^n e^o, \text{ where } m, n, o \text{ are } \geq 0$$

# LL(1) Grammars

---

$$S \rightarrow E \$$$
$$E \rightarrow E (E) \mid E [E] \mid \text{id}$$

Is the above Grammar LL(1) Explain your answer.

# Recall: Computing FIRST/FOLLOW

---

## ❖ Determining FIRST(X)

1. if  $X$  is a terminal, then add  $X$  to FIRST(X)
2. if  $X \rightarrow \varepsilon$  then add  $\varepsilon$  to FIRST(X)
3. if  $X$  is a nonterminal and  $X \rightarrow Y_1 Y_2 \dots Y_k$  then  $a$  is in FIRST(X) if  $a$  is in FIRST( $Y_i$ ) and  $\varepsilon$  is in FIRST( $Y_j$ ) for  $j = 1 \dots i-1$
4. if  $\varepsilon$  is in FIRST( $Y_1 Y_2 \dots Y_k$ ) then  $\varepsilon$  is in FIRST(X)

## ❖ Determining FOLLOW(X)

1. if  $S$  is the start symbol then  $\$$  is in FOLLOW(S)
2. if  $A \rightarrow \alpha B \beta$  then add all FIRST( $\beta$ )  $\neq \varepsilon$  to FOLLOW(B)
3. if  $A \rightarrow \alpha B$  or  $\alpha B \beta$  and  $\varepsilon$  is in FIRST( $\beta$ ) then add FOLLOW(A) to FOLLOW(B)

# First/Follow Computation

---

Construct First/Follow sets for the following LL(1) grammar for each non-terminal: S, A, B, C

$S \rightarrow AB$

$A \rightarrow bC$

$B \rightarrow aAB \mid \varepsilon$

$C \rightarrow (S) \mid c$

# Recall: Closure and Goto

---

- ❖ Closure of a parser state:
  - » Start with  $\text{Closure}(S) = S$
  - » Then for each item in  $S$ :
    - $X \rightarrow \alpha \cdot Y \beta$
    - Add items for all the productions  $Y \rightarrow \gamma$  to the closure of  $S$ :  $Y \rightarrow \cdot \gamma$
- ❖ Goto operation = describes transitions between parser states, which are sets of items
  - » If the item  $[X \rightarrow \alpha \cdot Y \beta]$  is in  $I$ , then
  - »  $\text{Goto}(I, Y) = \text{Closure}([X \rightarrow \alpha Y \cdot \beta])$

# DFA Construction for LR(0)

---

Construct the LR(0) DFA for the following grammar:

$E \rightarrow E + T \mid T$

$T \rightarrow TF \mid F$

$F \rightarrow F^* \mid a \mid b$

Note, \* is not the closure operator!

Are there any shift/reduce conflicts in table that you would construct from the DFA? How do you know?

# Bottom-up Parsing

---

Which is capable of parsing a larger set of languages, LR(0) or SLR?

# Abstract Syntax Trees

---

Draw the AST for the following C function

```
int foo()
{
    int a, b, c, i;
    a = 20;
    b = a * a;
    for (i=0; i<10, i++) {
        c = (10 + a) * b - (c / (a * i - b));
    }
}
```

# Semantic Analysis

---

- ❖ Ch 5, 6, 7
- ❖ Syntax-directed translation
  - » Semantic actions, building the AST
- ❖ Scope information
  - » Hierarchy of symbol tables
- ❖ Type checking
  - » Static/dynamic, strong/weak
  - » Static semantics, type judgments, proof tree

# Static Semantics

---

Consider the following language that manipulates ints, bools, and stacks

$T \rightarrow \text{int} \mid \text{bool} \mid \text{stack}(T)$

$E \rightarrow \text{id} \mid \text{num} \mid E + E$

$E \rightarrow \text{top}(E) \mid \text{pop}(E) \mid \text{push}(E,E) \mid \text{newstack}(T) \mid \text{empty}(E)$

Notes:

1. Add: can only be applied to ints or stacks,  $\text{int1} + \text{int2} = \text{sum}$ ,  $\text{stack1} + \text{stack2} = \text{stack of concatenation of elements (note elements must be same type)}$
2.  $\text{top}(E) = \text{top element of stack } E$
3.  $\text{pop}(E) = \text{resultant stack after removing top element}$
4.  $\text{push}(E, E') = \text{add } E' \text{ to stack, returns resulting stack}$
5.  $\text{newstack}(T) = \text{produces new, empty stack of type } T$
6.  $\text{empty}(E) = \text{is true if } E \text{ has no elements, false otherwise}$

## Static Semantics (2)

---

a) Write the static type-checking rules for all expressions in this language.

b) Is the following well typed in any context?

`pop(push(x,y+5))`

c) Is the following well typed in any context?

`push(newstack(int), top(x)) + push(y,empty(z))`

# Important Notes

---

- ❖ Just because I did not go over a problem on a topic does not mean its not important. There is limited time, so I cannot go over everything!!
- ❖ Extended office hours
  - » Tomorrow, Tuesday, Oct 31, 4-5:30pm