

- Formal languages (Lecture 2 plus chapters)
  - Regular languages
    - Regular expressions
    - DFAs
    - Use of regular languages in programming languages
  - Context-free languages
    - Context-free grammars
    - Derivation, parse, ambiguity
    - Use of CFGs in programming languages
    - Expression grammars, precedence, and associativity

Parsing (Lecture 3 plus chapters)

- LL Parsing (Lectures 3 and 4 plus chapters)
  - Recursive-descent parsing, recursive-descent routines
  - LL(1) grammars
  - LL(1) parsing tables
  - FIRST, FOLLOW, PREDICT
  - LL(1) conflicts

- Logic programming concepts (Lecture 5 plus chapters
  - Declarative programming
  - Horn clause, <u>resolution principle</u>
- Prolog (Lectures 5, 6, and 7 plus chapters)
  - Prolog concepts: search tree, rule ordering, unification, backtracking, backward chaining
  - Prolog programming: lists and recursion, arithmetic, backtracking cut, negation-by-failure, generate-and-test

- Binding and scoping (Lecture 8 plus reading)
  - Object lifetime
  - Combined view of memory
  - Stack management

- Scoping (in languages where functions are thirdclass values)
- Static and dynamic links
- Static (lexical) scoping
- Dynamic scoping

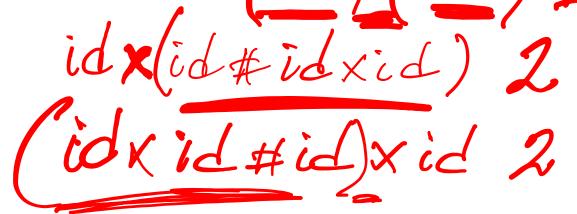
- Attribute grammars
  - Attributes
  - Attribute rules
  - Decorated parse trees
  - Bottom-up (i.e., S-attributed) grammars

Question 1. (2pts) Consider the expression grammar below.



How many parse trees are there for string id x id # id x id?

- (a) 0
- (b) 1
- (c) 2
- (d) 5



Question 2. (2pts) Below is a slightly modified version of the grammar from question 1.

 $expr o expr extbf{x} expr \mid term$   $term o term # id \mid id$ 

The following derivation

is

- (a) rightmost
- (b) leftmost
- (c) neither

Question 3. (2pts) Consider the following grammar. A, B, and S are the nonterminals. a, b, and c are the terminals. This grammar is a context-free grammar.

- $S o ext{abc}A$   $A o ext{a}AB ext{c} \mid ext{abc}$   $cB o B ext{c}$   $bB o ext{bb}$ 
  - (a) true
  - (D) false

Question 4. (2pts) Consider the following grammar. A, B, C, and S are 1. ponterminals. a, b, and c are the terminals. The grammar generates the language  $a^nb^nc^n$ ,  $n \ge 0$ .

1

- S 
  ightarrow ABC  $A 
  ightarrow aA \mid \epsilon$   $B 
  ightarrow bB \mid \epsilon$   $C 
  ightarrow cC \mid \epsilon$
- (a) true(b) false



Question 5. (2pts) Consider the grammar

$$S 
ightarrow {
m a} S {
m b} S \mid {
m b} S {
m a} S \mid \epsilon$$

The grammar is ambiguous.

- (a) true
- (b) rarse

Questions 1-4 refer to the "Dangling else" grammar we discussed in class:  $start \rightarrow stmt \$\$$  $stmt 
ightarrow ext{if b then } stmt \ else\_part \ | \ ext{a} \ else\_part 
ightarrow ext{else } stmt \ | \ \epsilon$ Question 1. (2pts) The grammar is ambiguous. then if b then a else a (a) true (D) ralse Question 2. (2pts) The grammar is LL(1). (a) true No AMBUGUOUS GRAMMAR IS LL Question 3. (2pts) How many parse trees for string if b then a else if b then a (a) 1

Question 4 (2pts) Recall that there is a conflict in LL(1) table entry [ $else\_part$ , else] as both  $else\_part \rightarrow else$  stmt and  $else\_part \rightarrow \epsilon$  apply on token else. (Or in other words, else is in the FREDICT set of both productions.) How can you resolve the conflict, so that an else would associate with the nearest unmatched then?

- (a) A ways expand by  $else\_part \rightarrow else\ stmt$  on else.
- (b) Always expand by  $else\_part \rightarrow \epsilon$  on else.

Question 5. (2pts) There exist unambiguous grammars that are not LL(1) grammars.

- (a) true
  - (b) false





Question 1. (1pt) Which inference Method does Prolog use?

Question 2. (2pts) The list [1,2|3] is a proper list.



Question 3. (2pts) The list [1,2|[3]] is a proper list.



Question 4. (1pt) The unification [1,2|3] = [1,2|[3]] succeeds.



3 = 1,3

Question 5. (2pts) Consider gcd (the Greatest Common Divisor algorithm) in Prolog. The program takes positive integers A and B and "returns" their greatest common divisor in R. Note: % starts a line comment in Prolog.

```
gcd(A,B,R):- A = B, R = A. %base case: when a=b, then GCD(a,b) = a = b. gcd(A,B,R):- A > B, A1 if A-B, gcd(A1,B,R). %when a>b, GCD(a,b) = GCD(a-b,b). gcd(A,B,R):- A < B, B1 is B A, gcd(A,B1,R). %when a<br/>b, GCD(a,b) = GCD(a,b-a). Is this program "invertible"? (That is, given arbitrary positive integers b and d, can we call ?- gcd(A,b,d). to generate a sequence of integers a such that GCD(a,b) = d?)
```

ged (A, b, d)

```
Question 6. (2pts) Recall our favorite classmates Prolog program:

takes(jane, his).

takes(jane, cs).

takes(ajit, art).

takes(ajit, cs).

classmates(X,Y):- takes(X,Z),takes(Y,Z).

Query ?- classmates(A,B). has this many answers (an answer is a pair of bindings A = ...,

B = ...):

Enter just one number on a single line in the first line of the text area below with no whitespace.
```

QUIZ4 1: c, 2: a, 3: a, 4: 201, 5: c, 6: c