Exam 1 Topics

Topics

- 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

Topics

- 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

Topics

- Logic programming concepts (Lecture 5 plus chapters
  - Declarative programming
  - Horn clause, resolution principle
- 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

Topics

- Binding and scoping (Lecture 8 plus reading)
  - Object lifetime
  - Combined view of memory
  - Stack management
  - Scoping (in languages where functions are third-class values)
  - Static and dynamic links
  - Static (lexical) scoping
  - Dynamic scoping

Topics

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

Quiz 1

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

$$expr \rightarrow expr \ \texttt{x} \ expr \mid expr \ \texttt{\#} \ expr \mid \texttt{id}$$

How many parse trees are there for string `id x id # id x id`? 1

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

id x (id # id x id) 2

(id x id # id) x id 2

Quiz 1

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

$$expr \rightarrow expr \texttt{ x } expr \mid term$$
$$term \rightarrow term \texttt{ \# id} \mid \texttt{id}$$

The following derivation

$$\textit{expr} \Rightarrow \underline{expr} \texttt{ x } expr \Rightarrow \underline{term} \texttt{ x } expr \Rightarrow \texttt{id x } \underline{expr} \Rightarrow \texttt{id x } \underline{expr} \texttt{ x } expr$$
$$\Rightarrow \texttt{id x } \underline{term} \texttt{ x } expr \Rightarrow \texttt{id x id x } \underline{expr} \Rightarrow \texttt{id x id x } \underline{term} \Rightarrow \texttt{id x id x id}$$

is

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

Quiz 1

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.

$$S \rightarrow \texttt{abc}A$$
$$A \rightarrow \texttt{a}AB\texttt{c} \mid \texttt{abc}$$
$$\texttt{c}B \rightarrow B\texttt{c}$$
$$\texttt{b}B \rightarrow \texttt{bb}$$

(a) `true`

(b) `false`

Quiz 1

Question 4. (2pts) Consider the following grammar. A, B, C, and S are the nonterminals. **a**, **b**, and **c** are the terminals. The grammar generates the language $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n, n \geq 0$.

$$\begin{aligned} S &\rightarrow ABC \\ A &\rightarrow \mathtt{a}A \mid \epsilon \\ B &\rightarrow \mathtt{b}B \mid \epsilon \\ C &\rightarrow \mathtt{c}C \mid \epsilon \end{aligned}$$

(a) `true`

(b) `false`

1

$a^k b^l c^m \supseteq a^n b^n c^n$

Quiz 1

Question 5. (2pts) Consider the grammar

$$S \rightarrow \mathtt{a}S\mathtt{b}S \mid \mathtt{b}S\mathtt{a}S \mid \epsilon$$

The grammar is ambiguous.

(a) `true`

(b) `false`

Quiz 2

10 points total

Questions 1-4 refer to the "Dangling **else**" grammar we discussed in class:

$start \rightarrow stmt$ `$$`
$stmt \rightarrow$ `if b then` $stmt$ $else\_part$ | `a`
$else\_part \rightarrow$ `else` $stmt$ | ϵ

Question 1. (2pts) The grammar is ambiguous.

(a) `true`
(b) `false`

if b then if b then a else a

Question 2. (2pts) The grammar is LL(1).

(a) `true`
(b) `false`

NO AMBIGUOUS GRAMMAR IS LL(1).

Question 3. (2pts) How many parse trees for string `if b then a else if b then a`?

(a) 1
(b) 2
(c) 3

Quiz 2

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 PREDICT set of both productions.) How can you resolve the conflict, so that an **else** would associate with the nearest unmatched **then**?

(a) Always 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`

$E \rightarrow E + T \mid T$
$T \rightarrow T * id \mid id$

Quiz 3

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

backward chaining

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

FALSE

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

TRUE

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

FALSE

[2|3]

1 2 3

3 = [3]

Quiz 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 is 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<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`?)

gcd(A, b, d)   No

Quiz 3

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.

6

QUIZ 4

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