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

ASSUME BOTTOM-UP EVALUATION!

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 Ch. 12
- 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 chapter)
  - 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 (Lectures 9 and 10 plus Chapters)
  - Attributes
  - Attribute rules
  - Decorated parse trees
  - Bottom-up (i.e., S-attributed) grammars
  - Top-down (i.e., L-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`?

(a) 0

(b) 1

(c) 2

[✓ (d) 5]

Expr Expr

id x id # id x id 2

id x id # id x id 1

expr expr

id x id # id x id 2

expr expr

5

Quiz 1

id x id x id

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

$$\begin{aligned} &expr \rightarrow expr \ \mathtt{x} \ expr \mid term \\ &term \rightarrow term \ \mathtt{\#} \ \mathtt{id} \mid \mathtt{id} \end{aligned}$$

The following derivation

$$expr \Rightarrow \underline{expr} \ \mathtt{x} \ expr \Rightarrow \underline{term} \ \mathtt{x} \ expr \Rightarrow \mathtt{id} \ \mathtt{x} \ \underline{expr} \Rightarrow \mathtt{id} \ \mathtt{x} \ \underline{expr} \ \mathtt{x} \ expr$$

$$\Rightarrow \mathtt{id} \ \mathtt{x} \ \underline{term} \ \mathtt{x} \ expr \Rightarrow \mathtt{id} \ \mathtt{x} \ \mathtt{id} \ \mathtt{x} \ \underline{expr} \Rightarrow \mathtt{id} \ \mathtt{x} \ \mathtt{id} \ \mathtt{x} \ \underline{term} \Rightarrow \mathtt{id} \ \mathtt{x} \ \mathtt{id} \ \mathtt{x} \ \mathtt{id}$$

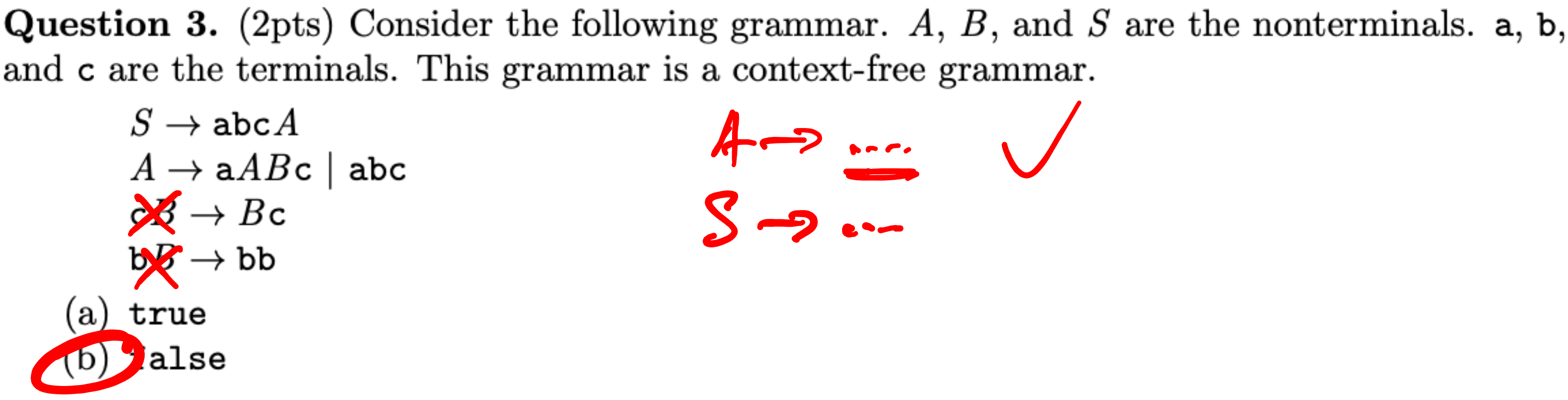
is

(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.

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

(a) true

[✓ (b) false]

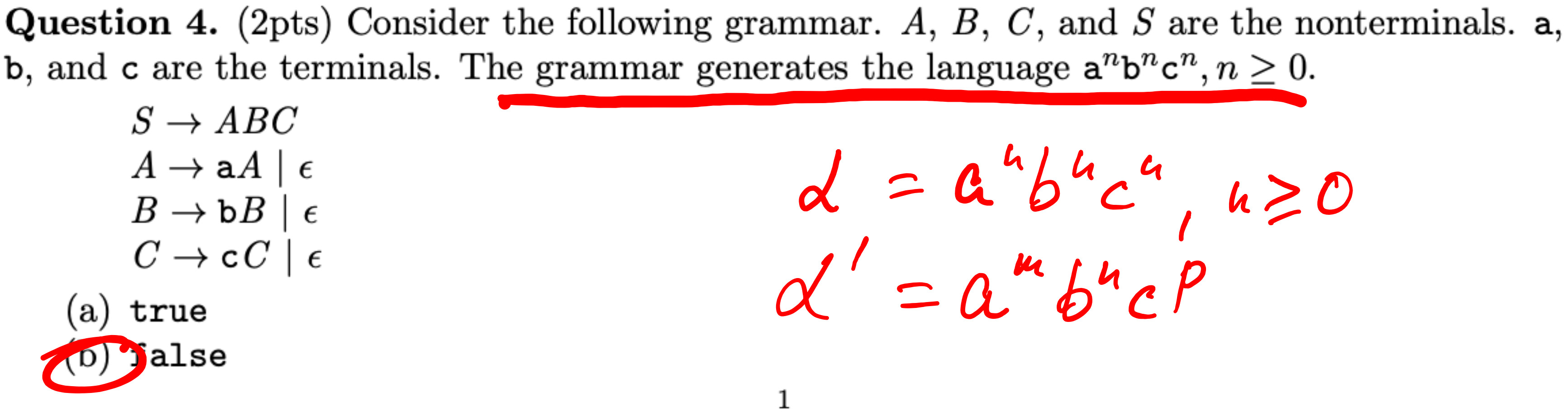
$A \rightarrow \underline{\ldots}$ ✓

$S \rightarrow \ldots$

Quiz 1

a^n $aaaa \dots a$

ε



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

$\alpha = a^n b^n c^n, n \geq 0$

$\alpha' = a^m b^n c^p$

Quiz 1

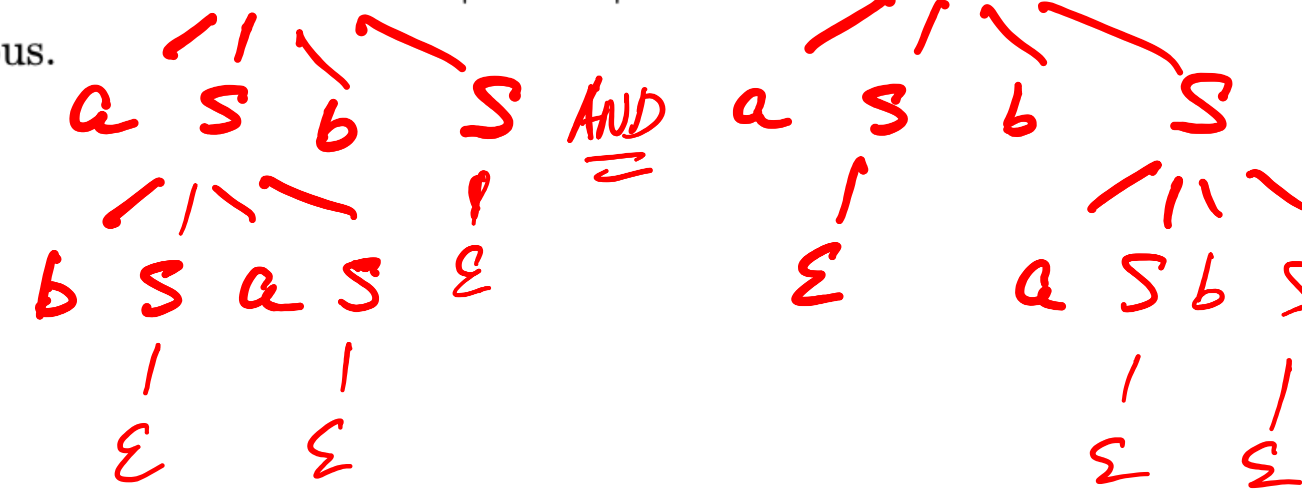
Question 5. (2pts) Consider the grammar

abab

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

The grammar is ambiguous.

[✓ (a) `true`]
(b) `false`



Quiz 2

id or id and id

Questions 1-4 refer to the following boolean expression grammar:

start → *bexp* $$

bexp → id or *bexp* | *bexp* and id | not *bexp* | id

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

[✓ (a) true]

(b) false

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

(a) true

[✓ (b) false]

NO AMBIGUOUS GRAMMAR IS LL(1).

Question 3. (1pts) id is in the FIRST(*bexp*).

[✓ (a) true]

(b) false

FIRST(bexp) = {id, not}

Question 4. (1pts) id is in the FOLLOW(*bexp*).

(a) true

[✓ (b) false]

FOLLOW(bexp) = {and, $$}

Quiz 2

start → stmt $$

stmt → **if b then** stmt else_part | **a**

else_part → **else** stmt | ε

FOLLOW(stmt) ⊆ FOLLOW(else_part)

Question 4. (2pts) Recall that there is a conflict in LL(1) table entry [*else_part*, `else`] as both *else_part* → `else` *stmt* and *else_part* → ϵ 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* → `else` *stmt* on `else`.]

(b) Always expand by *else_part* → ϵ on `else`.

	else
else_part	else_part → else stmt // else is in FIRST(else stmt) else_part → ε // else is in FOLLOW(else_part)

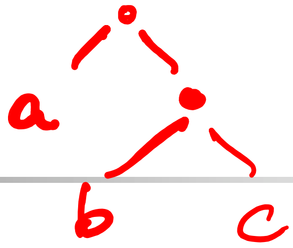
Question 5. (2pts) There exist unambiguous grammars that are not LL(1) grammars.

[✓ (a) `true`]

(b) `false`

ε is in FIRST(α) if α generates ε.

Quiz 3

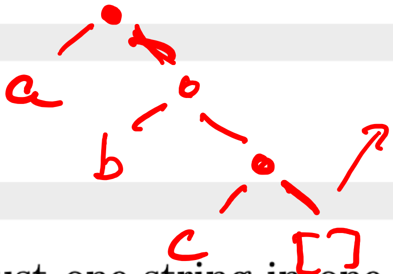


atom vs. []

atom is improper.

Question 1. (1pts) The list `[a,b|c]` is a proper list. FALSE

[a, b|c] = [a|[b|c]]



H T

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

Question 3. (2pt) Write the tail (rest-of-list) of `[1,2|3]`. Note: Enter just one string in one line in the first line of the text area below with no whitespace.

T = [2|3] NOT 2|3 NOT 3

[1, 2|3] = [H|T]

H = 1

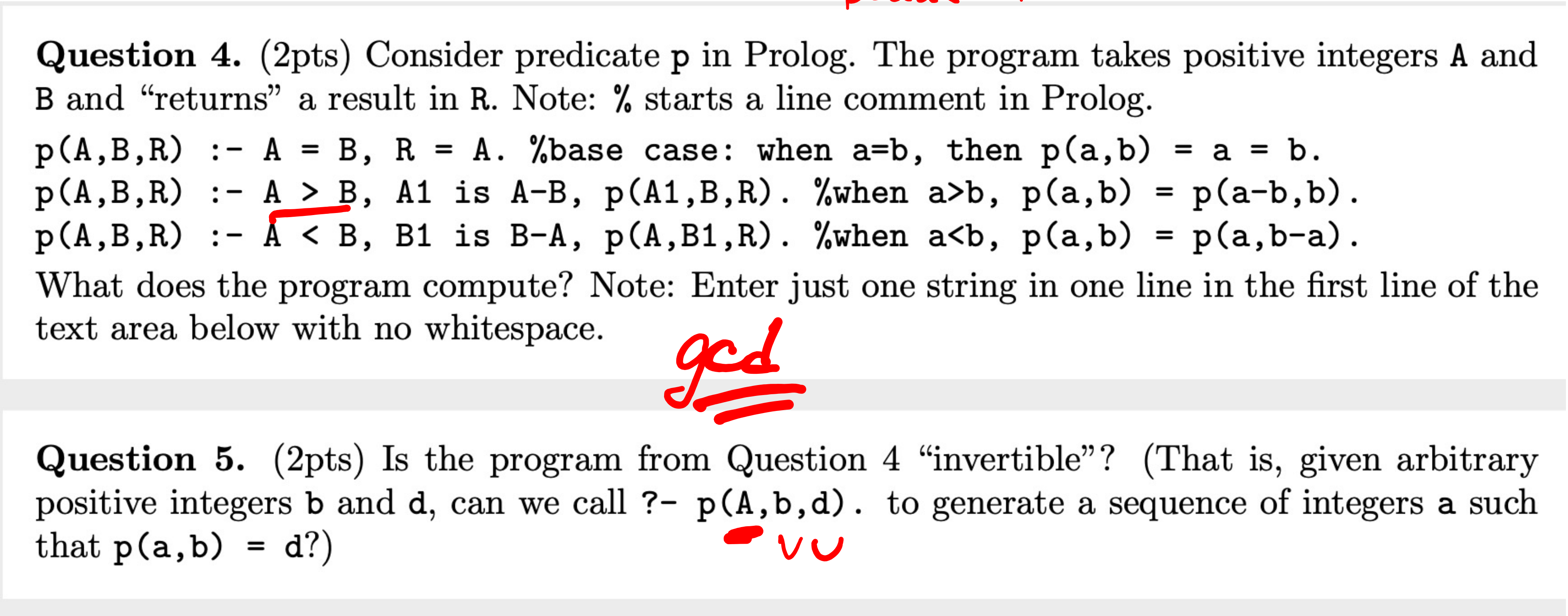
T = [2|3] ≠ [2, 3]

[1,2,3|4] =

[1|[2,3|4]]

Quiz 3

A>B and A1 is A-B requires that A and B are bound to numbers



Question 4. (2pts) Consider predicate p in Prolog. The program takes positive integers A and B and "returns" a result in R. Note: % starts a line comment in Prolog.

```
p(A,B,R) :- A = B, R = A. %base case: when a=b, then p(a,b) = a = b.
p(A,B,R) :- A > B, A1 is A-B, p(A1,B,R). %when a>b, p(a,b) = p(a-b,b).
p(A,B,R) :- A < B, B1 is B-A, p(A,B1,R). %when a<b, p(a,b) = p(a,b-a).
```

What does the program compute? Note: Enter just one string in one line in the first line of the text area below with no whitespace.

gcd

Question 5. (2pts) Is the program from Question 4 "invertible"? (That is, given arbitrary positive integers b and d, can we call ?- p(A,b,d). to generate a sequence of integers a such that p(a,b) = d?)

vu

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).
```

jane cs   ajit cs
jane cs   ~~jane~~ cs
jane ~~his~~   jane ~~his~~

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.

?- A=jane, B=jane ; // Z=his

?- A=jane, B=jane ; // Z=cs

?- A=jane, B=ajit ; // Z=cs

?- A=ajit ...

] 3 more cases

6

Quiz 4 Answers

- Question 1. the frame of main
- Question 2. the most recent frame of A before the current frame
- Question 3. x in main
- Question 4. 201
- Question 5. sometimes it binds to x in main and sometimes to B’s x
- Question 6. static semantic analysis