

EXAM

Please read all instructions, including these, carefully

- There are 7 questions on the exam, with multiple parts. You have 3 hours to work on the exam.
- The exam is open book, open notes.
- Please write your final answers in the space provided on the exam. You may use the backs of the exam pages as scratch paper, or use additional pages (available at the front of the room).
- Each problem has a straightforward solution. Solutions will be graded on correctness and clarity. Partial solutions will be given partial credit.

NAME : _____

Problem	Max points	Points
1	15	
2	10	
3	10	
4	10	
5	25	
6	15	
7	15	
TOTAL	100	

1. Consider the following augmented CFG for prefix expressions.

$$E' \rightarrow E$$

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

(a) Compute the following closures of items.

i) $I_a = \text{closure}(\{E' \rightarrow \bullet E\})$

$$\{ E' \rightarrow \bullet E$$

$$E \rightarrow \bullet + E E$$

$$E \rightarrow \bullet - E E$$

$$E \rightarrow \bullet \text{id} \}$$

ii) $I_b = \text{closure}(\{E \rightarrow + \bullet EE\})$

$$\{ E \rightarrow + \bullet EE$$

$$E \rightarrow \bullet + E E$$

$$E \rightarrow \bullet - E E$$

$$E \rightarrow \bullet \text{id} \}$$

iii) $I_c = \text{closure}(\{E \rightarrow +E \bullet E\})$

$$\{ E \rightarrow + E \bullet E$$

$$E \rightarrow \bullet + E E$$

$$E \rightarrow \bullet - E E$$

$$E \rightarrow \bullet \text{id} \}$$

iv) $I_d = \text{closure}(\{E \rightarrow \text{id} \bullet\})$

$$\{ E \rightarrow \text{id} \bullet \}$$

(b) Show the following entries of the SLR(1) parsing table M , where I_b and I_d indicate the states corresponding to the items sets computed in (a).

- i. $M[I_b, E]$ [goto] I_c
- ii. $M[I_d, +]$ Reduce $E \rightarrow \text{id}$
- iii. $M[I_d, \$]$ Reduce $E \rightarrow \text{id}$

2. Consider extending the LR(1) case to the general k lookahead symbols. Given an LR(k) grammar with N nonterminals and T terminals, how many columns would we have in a corresponding LR(k) action/goto table? State your assumptions and show your work for full credit.

With one symbol lookahead, we have one column for each terminal in the action table, and one for each non-terminal in the goto table, so $N + T$ columns. For k lookahead, we still have one column for each non-terminal and each individual terminal, but a column must be added for each possible combination of k symbols. So when $k = 2$, we have $T + T^2 + N$ columns. For larger values of k , we have $T + T^2 + T^3 \dots + T^{k-1} + T^k + N$ columns, which is

$$\sum_{n=1}^k T^n + N$$

3. Consider the following grammar G .

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow \text{id} \mid \text{id} () \mid \text{id} (L) \\ L &\rightarrow E ; L \mid E \end{aligned}$$

- (a) Give an LL(1) grammar that generates the same language as G .

First, eliminate left recursion:

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow + TE' \mid \varepsilon \\ T &\rightarrow \text{id} \mid \text{id} () \mid \text{id} (L) \\ L &\rightarrow E ; L \mid E \end{aligned}$$

Then eliminate common prefixes by left factoring:

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow + TE' \mid \varepsilon \\ T &\rightarrow \text{id} T' \\ T' &\rightarrow (T'' \mid \varepsilon \\ T'' &\rightarrow) \mid L) \\ L &\rightarrow E L' \\ L' &\rightarrow ; L \mid \varepsilon \end{aligned}$$

- (b) Prove that your grammar is LL(1) by whatever method you like. (Hint: This does not require an exhaustive construction.) Don't just restate the definition of LL(1).

Remember a grammar G is LL(1) iff whenever $A \rightarrow u \mid v$ are two distinct productions of G , the following conditions hold:

- For no terminal a do both u and v derive strings beginning with a .
- At most one of u and v can derive the empty string (ϵ), and
- If $v \Rightarrow^* \epsilon$ then u does not derive any string beginning with an element in $\text{FOLLOW}(A)$.

By constructing these FIRST and FOLLOW sets, we can prove the three requirements are satisfied:

$E' \rightarrow + T E' \mid e$	$\text{FIRST} (+ T E')$	$= \{ + \}$
	$\text{FOLLOW}(E')$	$= \{ \$,), ; \}$
$T' \rightarrow \epsilon \mid (T''$	$\text{FIRST}(T'')$	$= \{ (\}$
	$\text{FOLLOW}(T')$	$= \{ +, \$, ; \}$
$T'' \rightarrow) \mid L$	$\text{FIRST}())$	$= \{) \}$
	$\text{FIRST}(L)$	$= \{ \text{id} \}$
$L' \rightarrow ; L \mid \epsilon$	$\text{FIRST}(; L)$	$= \{ ; \}$
	$\text{FOLLOW}(L')$	$= \{) \}$

E' , T' , T'' , and L' are the only nonterminals with two productions.

1. Only T'' has 2 rules deriving non-null strings, and their FIRST sets are distinct.
2. At most of one rule of E' , T' and L' derives ϵ , since the other always includes a terminal.
3. E' , T' and L' are nullable, but the non-null rule of each RHS doesn't derive any string beginning with a symbol in the FOLLOW set of the LHS.

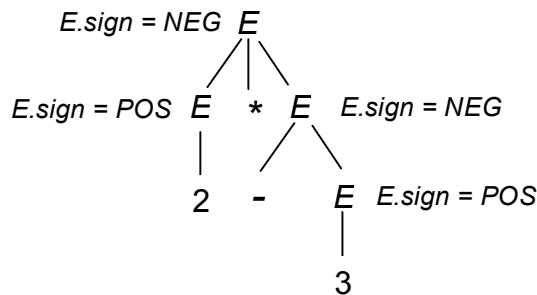
4. Here is an attribute grammar G for arithmetic expressions using multiplication, unary -, and unary +.

$$\begin{aligned}
 S &\rightarrow E \{ \text{if } (E.\text{sign} == \text{POS}) \text{ print}(\text{"result is positive"}); \\
 &\hspace{15em} \text{else print}(\text{"result is negative"}); \} \\
 E &\rightarrow \text{unsigned_int} \\
 &\quad | + E \\
 &\quad | - E \\
 &\quad | E * E
 \end{aligned}$$

(a) Add semantic actions to compute an attribute $E.\text{sign}$ for non-terminal E to record whether the arithmetic value of E is positive or negative. The attribute sign can have two values, either POS or NEG. To get you started, the first rule is provided.

$$\begin{aligned}
 E &\rightarrow \text{unsigned_int} && \{ E.\text{sign} = \text{POS} ; \} \\
 &\quad | + E_1 && \{ E.\text{sign} = E_1.\text{sign} ; \} \\
 &\quad | - E_1 && \{ \text{if } (E_1.\text{sign} == \text{POS}) \\
 &&& \hspace{2em} E.\text{sign} = \text{NEG} ; \\
 &&& \hspace{2em} \text{else } E.\text{sign} = \text{POS} ; \} \\
 &\quad | E_1 * E_2 && \{ \text{if } (E_1.\text{sign} == E_2.\text{sign}) \\
 &&& \hspace{2em} E.\text{sign} = \text{POS} ; \\
 &&& \hspace{2em} \text{else } E.\text{sign} = \text{NEG} ; \}
 \end{aligned}$$

(b) Show the parse tree for input $2 * - 3$ (where 2 and 3 are “unsigned_ints”). Indicate at each node what the values of associated attributes are.



(c) Is $E.\text{sign}$ an inherited attribute or a synthesized attribute?

Synthesized, as it is propagated from the leaf nodes upwards through the tree.

5. Consider the following grammar G :

$$\begin{aligned} S &\rightarrow Sa \mid bL \\ L &\rightarrow ScL \mid Sd \mid a \end{aligned}$$

(a) Remove left recursion from G . The result is called G' .

$$\begin{aligned} S &\rightarrow bLS' \\ S' &\rightarrow aS' \mid \varepsilon \\ L &\rightarrow ScL \mid Sd \mid a \end{aligned}$$

(b) Left factor G' . The result is G'' .

$$\begin{aligned} S &\rightarrow bLS' \\ S' &\rightarrow aS' \mid \varepsilon \\ L &\rightarrow SL' \mid a \\ L' &\rightarrow cL \mid d \end{aligned}$$

(c) Fill in the table below with the FIRST and FOLLOW sets for the non-terminals in grammar G'' (note that the number of rows may be more than you need):

Non-terminal	<i>First</i>	<i>Follow</i>
S	b	$c, d, \$$
S'	a, ε	$c, d, \$$
L	a, b	a
L'	c, d	a

(d) Fill in the LL(1) parse table for this grammar.

Non-terminal	a	b	c	d	$\$$
S		$S \rightarrow bLS'$			
S'	$S' \rightarrow aS'$		$S' \rightarrow \varepsilon$	$S' \rightarrow \varepsilon$	$S' \rightarrow \varepsilon$
L	$L \rightarrow a$	$L \rightarrow SL'$			
L'			$L' \rightarrow cL$	$L' \rightarrow d$	

(e) Using the parse table in (d), show a top-down parse of the string *bbacbada*\$.

Stack (with top at left)	Input	Action
S\$	bbacbada\$	$S \rightarrow bLS'$
bLS'\$	bbacbada\$	match b
LS'\$	bbacbada\$	$L \rightarrow SL'$
SL'S'\$	bbacbada\$	$S \rightarrow bLS'$
bLS'L'S'\$	bbacbada\$	match b
LS'L'S'\$	bbacbada\$	$L \rightarrow a$
aS'L'S'\$	bbacbada\$	match a
S'L'S'\$	bbacbada\$	$S' \rightarrow \epsilon$
L'S'\$	bbacbada\$	$L' \rightarrow cL$
cLS'\$	bbacbada\$	match c
LS'\$	bbacbada\$	$L \rightarrow SL'$
SL'S'\$	bbacbada\$	$S \rightarrow bLS'$
bLS'L'S'\$	bbacbada\$	match b
LS'L'S'\$	bbacbada\$	$L \rightarrow a$
aS'L'S'\$	bbacbada\$	match a
S'L'S'\$	bbacbada\$	$S' \rightarrow \epsilon$
L'S'\$	bbacbada\$	$L' \rightarrow d$
dS'\$	bbacbada\$	match d
S'\$	bbacbada\$	$S' \rightarrow aS'$
aS'\$	bbacbada\$	match a
S'\$	bbacbada\$	$S' \rightarrow \epsilon$
\$	bbacbada\$	ACCEPT

6. Consider the following simple context free grammars:

$$\begin{array}{ll}
 G_1: & S \rightarrow Aa \\
 & A \rightarrow \varepsilon \\
 & A \rightarrow bAb \\
 G_2: & S \rightarrow Aa \\
 & A \rightarrow \varepsilon \\
 & A \rightarrow Abb
 \end{array}$$

Note that the grammars generate the same language: strings consisting of even numbers of b 's (including 0 of them), followed by an a .

- (a) Attempt to show a shift-reduce parse of the string $bbbba$ for a parser for grammar G_1 . Show the contents of the stack, the input, and the actions (i.e., shift, reduce, error, accept). You don't need to create a parse table; just use your knowledge of the grammar and how the parser works. Be sure to indicate any conflicts and explain why they are conflicts. Is G_1 LR(1)? Is G_1 LR(0)?

Stack (with top at right)	Input	Action
\$	bbbba\$	shift b
\$b	bbba\$	shift b
\$bb	bba\$	CONFLICT: reduce $A \rightarrow \varepsilon$ or shift b?

There is always a conflict while b 's still remain in the input, because you need to reduce by $A \rightarrow \varepsilon$ only once, at the point where half of the b 's have been shifted onto the stack. However, as long as there are b 's in the input you have no idea when you've reached the halfway point. Therefore, the option to shift b or reduce by $A \rightarrow \varepsilon$ is always possible if a b is the next item in the input.

G_1 is neither LR(0) nor LR(1) since a lookahead of $(\text{input length}/2)+1$ is required to determine when to reduce by $A \rightarrow \varepsilon$.

- (b) Attempt to show a shift-reduce parse of the string $bbbba$ for a parser for grammar G_2 . Show the contents of the stack, the input, and the actions (i.e., shift, reduce, error, accept). You don't need to create a parse table; just use your knowledge of the grammar and how the parser works. Be sure to indicate any conflicts and explain why they are conflicts. Is G_2 LR(1)? Is G_2 LR(0)?

Stack (with top at right)	Input	Action
\$	bbbba\$	reduce $A \rightarrow \varepsilon$
\$A	bbba\$	shift b
\$b	bba\$	shift b
\$Abb	bba\$	reduce $A \rightarrow Abb$
\$A	bba\$	shift b
\$Ab	ba\$	shift b
\$Abb	a\$	reduce $A \rightarrow Abb$
\$A	a\$	shift a
\$Aa	\$	reduce $A \rightarrow Aa$
\$S	\$	ACCEPT

G_2 is both LR(0) and LR(1), since regardless of what is in the input, you reduce by $A \rightarrow Abb$ if Abb is on the stack, and reduce by $A \rightarrow Aa$ when Aa is on the stack.

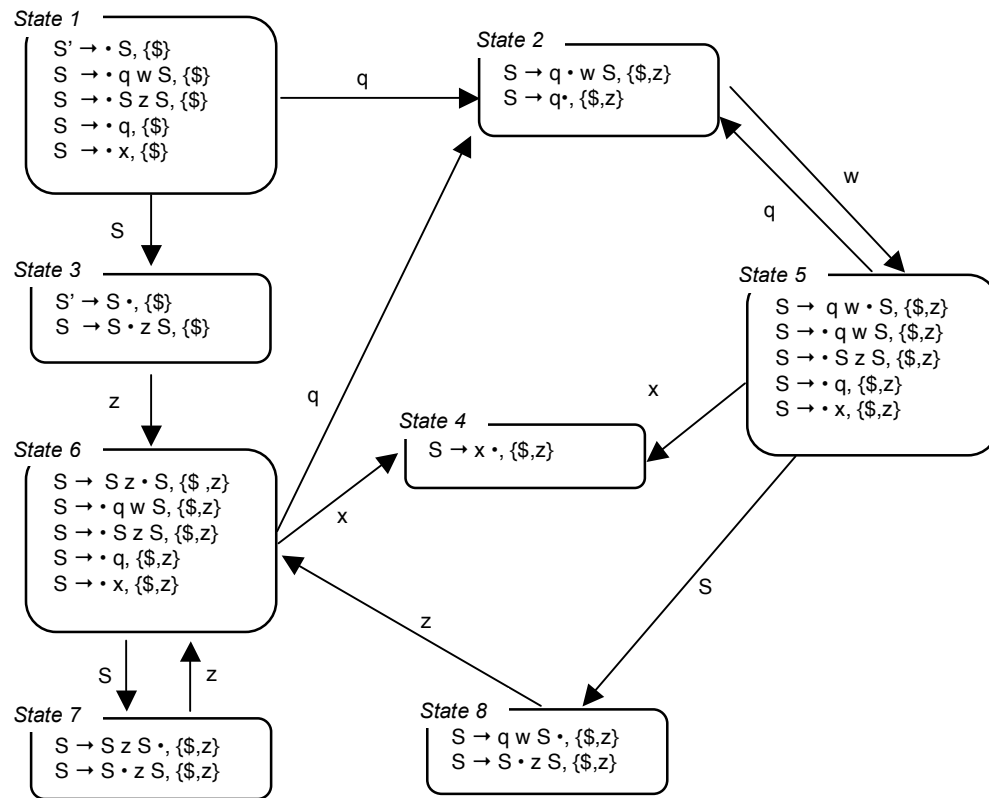
- (c) Indicate whether G_1 and G_2 are LL(1) and explain your answer. You don't need to construct the parse tables, but may argue from other properties.

G_2 is obviously LL(1) since it is LR(1), and LL(1) grammars are a subset of LR(1) grammars. G_1 is not LL(1) because with one character lookahead it cannot be decided whether to expand A to bAb or ϵ for the same reasons as given above in (a). That is, you have to know when you have processed half of the b 's, which is dependent on the length of the input string.

- (c) Of the language classes we have discussed, which is the *smallest* category into which $L(G_1)$ fits? Justify your answer.

$L(G_1)$ is a regular language, as it can be described with a regular expression: $(bb)^*a$.

7. Consider the following GOTO graph of an LALR(1) parser. Notice the state numbers and refer to them in the answers to the following questions.



- (a) Which states of the parser have conflicts? What kind of conflicts are they, and under what circumstances do they arise?

State 7 has a shift/reduce conflict because with z as the lookahead, we can either shift z (on the basis of $S \rightarrow S \cdot z$) or reduce by $S \rightarrow S z$ (on the basis of $S \rightarrow S z \cdot, \{\$,z\}$).

State 8 has a similar shift/reduce conflict because with z as the lookahead, we can either shift z (on the basis of $S \rightarrow S \cdot z S, \{\$,z\}$) or reduce by $S \rightarrow q w S$ (on the basis of $S \rightarrow q w S \cdot, \{\$,z\}$).

- (b) Describe the possible contents of the stack when the parser is in state 5.

The stack could contain 0 or more S 's followed by qw (transitions through states 1-2->5), or it could contain one or more Sz 's followed by qw (transitions through states 1-3->6->7->6 – repeating 6->7->6 any number of times – 2->5).

- (c) Give a string that could remain in the input when the parser is in state 5 that would lead to a successful parse.

Zero or more S 's followed by qw will lead to a successful parse.